### Description

Need a global memcap, maybe per client, per server and/or per flow as well. Question is how to implement it. I see several options:

- We could possibly do it in libhtp itself, tracking and limit enforcing. Might not be feasible to do anything but per flow (htp_conn)
- We could somehow use our own alloc/free functions in libhtp, which would have to modified to support this
- Maybe libhtp can track allocs, with us getting access to the trackers. Might not be very precise.

Need to discuss with libhtp upstream.

### Subtasks

<table>
<thead>
<tr>
<th>Feature # 1067: memcap for http inside suricata</th>
<th>Closed</th>
</tr>
</thead>
</table>

### History

**#1 - 07/23/2013 07:38 AM - Ivan Ristic**

Victor Julien wrote:

> Need a global memcap, maybe per client, per server and/or per flow as well. Question is how to implement it. I see several options:

> - We could possibly do it in libhtp itself, tracking and limit enforcing. Might not be feasible to do anything but per flow (htp_conn)
> - We could somehow use our own alloc/free functions in libhtp, which would have to modified to support this
> - Maybe libhtp can track allocs, with us getting access to the trackers. Might not be very precise.

> Need to discuss with libhtp upstream.

Are you seeing excessive memory usage, or is this something you’d like to have?

Unrelated to memory consumption, I think it’s a good idea to support custom memory allocators, per HTTP transaction. This could be achieved with 2 additional functions (alloc and free) and the existing transaction hooks (to support memory pools; clear memory pool when the transaction is destroyed).

That said, if memory allocation fails the entire stream will fail. It's better to make LibHTTP aware of allocation limits so that it can fall back to different behaviours. For example, it could process request parameters and headers by default, but fall back to streaming-only (non copying) processing under constrained memory conditions. That would allow you to process request/response bodies. In parallel, you should improve Suricata to do most of its processing in a streaming fashion, as data becomes available.

As for the limit itself, I think it should be dynamic. For example, there could be an upper per-transaction (and, possibly per-connection) limit, but there could also be a global limit for all HTTP processing. A transaction that needs a lot of memory could use it if available, but if many concurrent transactions are processed, the per-transaction limit would go down.

**#2 - 07/23/2013 08:18 AM - Victor Julien**

Ivan Ristic wrote:

> Are you seeing excessive memory usage, or is this something you’d like to have?

Both actually. We see Suricata using a lot more memory than our limits allow when processing HTTP traffic. This goes to the order of Suricata giving 3GB in various settings, but using 13GB of real memory. We need to be able to control it better to our memory use is more predictable.

Unrelated to memory consumption, I think it’s a good idea to support custom memory allocators, per HTTP transaction. This could be achieved with 2 additional functions (alloc and free) and the existing transaction hooks (to support memory pools; clear memory pool when the transaction is destroyed).
That said, if memory allocation fails the entire stream will fail. It's better to make LibHTP aware of allocation limits so that it can fall back to
different behaviours. For example, it could process request parameters and headers by default, but fall back to streaming-only (non copying)
processing under constrained memory conditions. That would allow you to process request/response bodies. In parallel, you should improve
Suricata to do most of its processing in a streaming fashion, as data becomes available.

As for the limit itself, I think it should be dynamic. For example, there could be an upper per-transaction (and, possibly per-connection) limit, but
there could also be a global limit for all HTTP processing. A transaction that needs a lot of memory could use it if available, but if many
concurrent transactions are processed, the per-transaction limit would go down.

Agreed, although I think the dynamic thing might be more of a "nice to have".

#3 - 07/23/2013 08:34 AM - Ivan Ristic
Victor Julien wrote:

Ivan Ristic wrote:

Are you seeing excessive memory usage, or is this something you'd like to have?

Both actually. We see Suricata using a lot more memory than our limits allow when processing HTTP traffic.

Is this with 1.4.x or 2.x?

#4 - 07/23/2013 08:49 AM - Victor Julien
This number was from 2.0dev (so our current git) with libhtp git as well. But I've seen this pattern since the beginning of our project. I'm not saying it's
bug or a memleak, just that we need toggles to control how much libhtp uses per flow.

Btw, in all fairness the libhtp glue code we have isn't using memcaps either, so there may be some memory there as well.

#5 - 07/23/2013 09:11 AM - Ivan Ristic
Victor Julien wrote:

This number was from 2.0dev (so our current git) with libhtp git as well. But I've seen this pattern since the beginning of our project. I'm not
saying it's bug or a memleak, just that we need toggles to control how much libhtp uses per flow.

With LibHTP 0.5.x you should see a substantial decrease in memory consumption because the large (per-connection) buffers are now gone.

As for memory leakage in LibHTP, I am running all my tests though Valgrind and there's nothing there, but I don't have a good production site to test
with.

Btw, in all fairness the libhtp glue code we have isn't using memcaps either, so there may be some memory there as well.

IIRC, you are not consuming transactions as they are parsed by LibHTP, which means that you're holding some allocated data for longer than
necessary.

I think a good next step is to build an optional memory profiler into LibHTP so that we can at least trace what the consumption is.

#6 - 07/23/2013 12:32 PM - Victor Julien
All fair points, but even if all that is done/fixed/changed/addressed we'll still want the memcap settings. I'm not suspecting memleaks at this time, as
memory use stabilizes at some point.

#7 - 10/25/2013 06:27 AM - Victor Julien
- Assignee set to Victor Julien
- Target version set to 3.0RC2

#8 - 11/05/2014 04:57 AM - Victor Julien
- Target version changed from 3.0RC2 to 70

#9 - 08/07/2020 01:55 PM - Victor Julien
- Target version changed from 70 to TBD