Web Server Core Design (Part 3)

BY EVAN OLDS

CPT S 422

Timeouts

- Q: What if a malicious (or just very slow) client connects to your server, sends a small amount of valid data, and then just halts?
- Recall: NetworkStream blocks on a read call, so it could halt that thread indefinitely if the socket connection remains active but no data is sent to complete the client request

Read Timeout

- NetworkStream.ReadTimeout property
- Gives us the ability to have a Read call terminate after a certain amount of time
  - Prevents indefinite wait
  - What happens when the timeout expires?
    - Read that returns 0? (would potentially violate the end of stream rule)
    - Returns partial data? (what if there is none in the network buffer?)
    - (investigate in class)
- By the way: default read timeout is infinite

Read Timeout

- What happens when the timeout expires?
- From the MSDN page:

"If the read operation does not complete within the time specified by this property, the read operation throws an IOException."

Read Timeout

- A nice feature, but doesn't do all the work for us
- Need to decide what to do when a read call times out?
  - Just terminate the connection?
  - Depending on what the timeout is, this may or may not be what you want
  - Fairly likely that it is. If you don't get ANY data within a few (seconds/milliseconds?) on a read call, the connection may be considered idle
  - Consider the buffer size on your read and how much data you could potentially get

Read Timeout

- Recall: you read in a loop to build a request

```
while (true) {
   Read from socket
   Append to memory stream
   Validate memory stream
}
```

Read Timeout

- If you have set a timeout but the read never times out, is everything ok? Proven to be a non-malicious request?

```
while (true) {
   Read from socket // might never time out
   Append to memory stream
   Validate memory stream
}
```

Read Timeout

- Consider scenario
  - Read timeout of 1 second
  - No read call times out, but each one takes just about 1 full second
  - Each read call delivers a byte
  - 1000 byte “valid” (according to HTTP format specifications at least) request
- Uses up 1000 seconds (16 minutes!) on that request handling thread
- What to do?

Read Timeout

- Implement in your code: Read timeout for individual reads and also read timeout for the whole request
- You have to implement that second part of the timeout logic “manually”
  - Start a timer when TCP client first connects
  - After each read call completes successfully, check total time
  - If it’s over a “reasonable” limit for one request, close the socket connection

Read Timeout

- With the timeout logic implemented, might you terminate non-malicious request that are just coming from very slow clients? Potentially yes
  - Need to choose a reasonable timeout and just accept this fact
  - Can be lenient enough to allow fairly slow clients
  - We're just talking about the data up to the body of the request
  - Should arrive in
    - 1 second
    - 10 seconds
    - 30 seconds
    - 1 minute
    - ?
  - Your choice, no golden rule

Other Malicious Client Scenarios?

- Suppose: Client sends request to server in a reasonable amount of time
  - Read calls don't timeout
  - Don't exceed total time for request up-to-body
- What else should we be looking out for?

Size Threshold

- Client could send a LOT of data in a short amount of time if connection is fast
- Could send 100 megabytes in a few seconds of bogus headers before reaching the body of the request
  - Timeout logic wouldn't catch this
  - Request format validation logic wouldn't catch it either
  - If we had a time-to-body total timeout of 30 seconds, we could build up hundreds of megabytes (maybe even into gigabytes) in memory
- We should have an additional threshold for the size of content before the body of the request
- Terminate connection if this size is exceeded

Size Threshold

- Again, no magic number for the size threshold
- 100 kilobytes maybe?
  - Remember this is just the stuff before the body, so request line and headers
- But wait, there's only a limited number of official headers defined by HTTP, can't we just verify against that? What would be the problem with this? What question do we need to go to the HTTP specification to answer?

HTTP Headers

- What question do we need to go to the HTTP specification to answer?
  - "Can there be more headers than what are included in the official list of HTTP headers at this time? In other words, can new headers be added?"
- Answer: Yes, in fact there is no official limit
  - Except from RFC 7230 on the next slide

HTTP Headers

- From RFC 7230, section 3.2.1. “Field Extensibility”:

Header fields are fully extensible: there is no limit on the introduction of new field names, each presumably defining new semantics, nor on the number of header fields used in a given message. Existing fields are defined in each part of this specification and in many other specifications outside this document set.

New header fields can be defined such that, when they are understood by a recipient, they might override or enhance the interpretation of previously defined header fields, define preconditions on request evaluation, or refine the meaning of responses.

Size Threshold

- Back to the problem, how much header data (and request line data) do we allow?
  - Have to chose
  - Specification doesn't set a fixed number
- Need to choose some limit. Can't let it be unlimited or our server is vulnerable to attacks. You can't just say "oh well the server is going to be a strong machine, it will have enough memory to hold whatever data can come in before the timeout."
  - Why not? (discuss)

Additional Size Thresholds

- May want to have multiple thresholds
  - Maybe 5 kb max for first line, 100 kb for all content up to body
- It's not those exact numbers that are the point, rather it may be known that your server doesn't support any URI longer than a certain value, and the HTTP method + spaces + HTTP version all combined make for a limit on how long the first line can be
  - Perhaps if you haven't seen CRLF (that's the first SINGLE line break, not the double) after some number of bytes then terminate connection

One Last “Unblock”-Oriented Thing

- TcpListener.AcceptTcpClient blocks
  - Should accept connections while the server is active, so we really only need to “unblock” when terminating the server
  - (Future versions may have additional reasons/scenarios to unblock from this which DON’T involve terminating the app, but we’ll discuss those later)

Back to Functionality

- Once the request parser/builder is robust enough, we can go back to thinking about functionality
- Reminder: Header fields give us info
  - Already talked some about Content-Length header
- The WebRequest class has info about the request and has member function to look through the collection for certain known header fields
- Consider the function on the next slide

GetContentLengthOrDefault

```
public static long GetContentLengthOrDefault(ConcurrentDictionary<string, string> headers,
 long defaultValue)
{
    if (headers.ContainsKey("Content-Length")) {
        string val = headers["Content-Length"];
        long len;
        if (long.TryParse (val, out len)) {
            return len;
        }
        return defaultValue;
    }
    return defaultValue;
}
```

- Assume the headers are all in the dictionary and have been correctly parsed from the request
- Why the default value parameter?
- What's wrong with the implementation?
- (discuss in class)

GetContentLengthOrDefault

```
public static long GetContentLengthOrDefault(ConcurrentDictionary<string, string> headers,
 long defaultValue)
{
    if (headers.ContainsKey("Content-Length")) {
        string val = headers["Content-Length"];
        long len;
        if (long.TryParse (val, out len)) {
            return len;
        }
        return defaultValue;
    }
    return defaultValue;
}
```

- Why the default value parameter?
  - Content-Length header is not required to be in the request headers
- What's wrong with the implementation?
  - Although it would work for the majority of requests from modern browsers, it's neglecting the fact that the header names are case-insensitive?
  - How to fix? (question clarified on next slide)

Searching Dictionary of Headers (Properly)

- The problem statement is as follows:

We have a dictionary of key,value pairs, both strings, for the headers from a request. Request building function removed whitespace from the ends of the field values, but left the rest as it was in the request. The HTTP specification states (RFC7230 section 3.2) that field names are case insensitive. If we want to search for a particular header (like Content-Length) how do we write a function to do so?

(implement on whiteboard in class)

WebService Class

- From the homework:

```
internal abstract class WebService
{

    public abstract void Handler(WebRequest req);

    /// <summary>
    /// Gets the service URI. This is a string of the form:
    /// /MyServiceName.whatever
    /// If a request hits the server and the request target starts with this
    /// string then it will be routed to this service to handle.
    /// </summary>
    public abstract string ServiceURI
    {
        get;
    }
}
```

WebService Class

- Once we have the ability to add web services to our server, we can start making actual “web apps” that do something interesting
- Future lectures will discuss different types of apps that we might want, how to implement them and potentially how to extend the server core code in the process