New issue

Jump to bottom

READY - Stop DOS attacks by making the lexer stop early on evil input. #2892

Merged **bbakerman** merged 11 commits into `master` from `ddos_stop_lexer_early` on Jul 26

bbakerman commented on Jul 21 • edited Member

This is related to #2888

The bug was that we do indeed have a max token counting mechanism in graphql-java BUt it was being enacted too late.

Testing showed that the max token code was indeed being hit BUT the ANTLR lexing and parsing code was taking proportionally longer to get to the max token state as the input size increased

This is cause bv the greedy nature of the ANTLR Lexer - it will look ahead and store tokens in memory under certain grammar conditions and butted directives like `@lol@lol` are one of them. This meant that the lexer was the code contributing to CPU time and not the parser - BUT the max token code check was in the parser.

This PR puts the same max token checks on the lexer as it does in the parser. In fact it debatable if the parser checks should still be retained (since the lexer will be the main way this will be hit) but the logic is common so I left it in place.

The current existing billion `@lols` test still parse with this change - The difference is where cancel parse exception is being thrown.

The use of the lexer as the place to count means the counting checks are done in constant time. As soon as the max tokens is encountered, the parse is stopped.

graphql-java also uses 3 channels for tokens (0 for the grammar and 2 and 3 for comments and whitespace). The lexing of whitespace and comments also take up CPU time so they counters have been put in place to watch tokens on those channels as well.

This will stop a "mostly whitespace" attack, even though whitespace costs less to aggregate in practice

👍 2

bbakerman added 3 commits 4 months ago

- This stops DOS attacks by making the lexer stop early. ✓ 772084a
- This stops DOS attacks by making the lexer stop early. Added BadSitua… ✓ 2caa273
- This stops DOS attacks by making the lexer stop early. Added BadSitua… ✓ 30ee65a

bbakerman changed the title ~~Stop DOS attacks by making the lexer stop early on evil input.~~ **WIP - Stop DOS attacks by making the lexer stop early on evil input.** on Jul 21

bbakerman commented on Jul 21 • edited — Member, Author

The questions to be answered here is whether to track the whitespace and line comment channels or not. They are fast BUT they still take some time BUT not like the grammar channel does.

when we run the ParserBadSituations program with unlimited tokens the numbers for whitespace and comments are something like

```
Whitespace Bad Payloads(run #2)(1 of 15) - | query length 50020 | bad payloads 5000 | duration 4ms
Whitespace Bad Payloads(run #2)(2 of 15) - | query length 100020 | bad payloads 10000 | duration
9ms
Whitespace Bad Payloads(run #2)(3 of 15) - | query length 150020 | bad payloads 15000 | duration
16ms
Whitespace Bad Payloads(run #2)(4 of 15) - | query length 200020 | bad payloads 20000 | duration
19ms
Whitespace Bad Payloads(run #2)(5 of 15) - | query length 250020 | bad payloads 25000 | duration
28ms
Whitespace Bad Payloads(run #2)(6 of 15) - | query length 300020 | bad payloads 30000 | duration
28ms
Whitespace Bad Payloads(run #2)(7 of 15) - | query length 350020 | bad payloads 35000 | duration
50ms
Whitespace Bad Payloads(run #2)(8 of 15) - | query length 400020 | bad payloads 40000 | duration
41ms
Whitespace Bad Payloads(run #2)(9 of 15) - | query length 450020 | bad payloads 45000 | duration
46ms
Whitespace Bad Payloads(run #2)(10 of 15) - | query length 500020 | bad payloads 50000 | duration
48ms
Whitespace Bad Payloads(run #2)(11 of 15) - | query length 550020 | bad payloads 55000 | duration
55ms
Whitespace Bad Payloads(run #2)(12 of 15) - | query length 600020 | bad payloads 60000 | duration
52ms
Whitespace Bad Payloads(run #2)(13 of 15) - | query length 650020 | bad payloads 65000 | duration
87ms
Whitespace Bad Payloads(run #2)(14 of 15) - | query length 700020 | bad payloads 70000 | duration
84ms
Whitespace Bad Payloads(run #2) - finished | max time was 87 ms
======================
```

```
Comment Bad Payloads(run #2)(1 of 15) - | query length 75022 | bad payloads 5000 | duration 14ms
Comment Bad Payloads(run #2)(2 of 15) - | query length 150022 | bad payloads 10000 | duration 25ms
Comment Bad Payloads(run #2)(3 of 15) - | query length 225022 | bad payloads 15000 | duration 31ms
Comment Bad Payloads(run #2)(4 of 15) - | query length 300022 | bad payloads 20000 | duration 34ms
Comment Bad Payloads(run #2)(5 of 15) - | query length 375022 | bad payloads 25000 | duration 26ms
Comment Bad Payloads(run #2)(6 of 15) - | query length 450022 | bad payloads 30000 | duration 25ms
Comment Bad Payloads(run #2)(7 of 15) - | query length 525022 | bad payloads 35000 | duration 32ms
Comment Bad Payloads(run #2)(8 of 15) - | query length 600022 | bad payloads 40000 | duration 37ms
Comment Bad Payloads(run #2)(9 of 15) - | query length 675022 | bad payloads 45000 | duration 48ms
Comment Bad Payloads(run #2)(10 of 15) - | query length 750022 | bad payloads 50000 | duration
51ms
Comment Bad Payloads(run #2)(11 of 15) - | query length 825022 | bad payloads 55000 | duration
48ms
Comment Bad Payloads(run #2)(12 of 15) - | query length 900022 | bad payloads 60000 | duration
48ms
Comment Bad Payloads(run #2)(13 of 15) - | query length 975022 | bad payloads 65000 | duration
52ms
Comment Bad Payloads(run #2)(14 of 15) - | query length 1050022 | bad payloads 70000 | duration
70ms
Comment Bad Payloads(run #2) - finished | max time was 70 ms
=======================
```

So it starts to creep up BUT nothing like the grammar file which is 2 orders of magnitude slower!

This is what is discovered debugging. Channel 0 (grammar tokens) get put into the parse tree and this is quite costly when you have 10s of 1000s of them. The cost is in the lexing AND in the parse tree building so it makes total sense for them to be strongly limited.

However there is 2 other channels - whitespace and line comments (`# comments`). Now it turns out that they are NOT anywhere near as costly. The whitespace (while lexed as single tokens per whitespace character) are accumulated (burns some CPU and memory since each becomes a Token) but they are not placed into the parse tree - but rather accumulated. As you can see above it happens relatively fast.

And then when the parser is called back - this line means we throw them away.

```
private void addIgnoredChars(ParserRuleContext ctx, NodeBuilder nodeBuilder) {
    if (!parserOptions.isCaptureIgnoredChars()) {
        return;
    }
```

So they accumulate fast enough and then are never used.

The line comments are also accumulated, not as fast because they are actually used in the parser tree.

```
protected List<Comment> getComments(ParserRuleContext ctx) {
    if (!parserOptions.isCaptureLineComments()) {
        return NO_COMMENTS;
    }
```

By default (even for queries) we keep the line comments. So an attack vector is to send in

```
# lots of comments
# lots of comments
# lots of comments
# lots of comments
query x { f }
```

However as the numbers above show even this is fast-ish.

My fear is around whitespace. That 15,000 whitespace characters will catch some queries out. I mean some queries might be say 512KB in size - they will likely be below 15,000 grammar tokens BUT could they be 1/3 whitespace - (170,000 whitespace chars say) - yeah why not.

So this PR as it is (counting whitespace the same as grammar tokens or line comments) will stop them sending in such a query.

I think this leads us towards having 2 `max` values in ParserOptions - the max tokens and a max whitespace number.

bbakerman added 5 commits 4 months ago

This stops DOS attacks by making the lexer stop early. Added per quer…	···	79b989c
This stops DOS attacks by making the lexer stop early. Added whitespa…	···	888d123
This stops DOS attacks by making the lexer stop early. Added whitespa…	···	1669ca7
This stops DOS attacks by making the lexer stop early. Added whitespa…	···	077e64a
This stops DOS attacks by making the lexer stop early. Added whitespa…	···	17ec04b

bbakerman changed the title ~~**WIP - Stop DOS attacks by making the lexer stop early on evil input.**~~ **READY - Stop DOS attacks by making the lexer stop early on evil input.** on Jul 22

bbakerman commented on Jul 22 — Member — Author

After chats with Andi and Donna, we decided to

- add a separate whitespace max token count - this is set at 200_000 whitespace tokens
- add a separate set of default parser options for "operation" parsing versus DSL
  - We don't want comment lines captured by default on operation AST elements

bbakerman mentioned this pull request on Jul 22

Denial of Service via Directive overloading #2888

Closed

bbakerman requested review from **andimarek** and **dondonz** 4 months ago

bbakerman added 2 commits 4 months ago

`This stops DOS attacks by making the lexer stop early.Use array inste…` ··· ✓ `a0a7cfd`

`This stops DOS attacks by making the lexer stop early.Use array inste…` ··· ✓ `50206dd`

andimarek requested changes on Jul 25

View changes

andimarek left a comment — Member

1: Small naming change

2: I think with introducing defaultOperationParserOptions we should rethink the `SchemaParser.parseImpl` where we overwrite the `maxTokens` . I think we don't need this anymore if we have two different ParserOptions.

`src/main/java/graphql/parser/Parser.java`

```
...  ...   @@ -46,6 +49,11 @@
46   49    @PublicApi
47   50    public class Parser {
48   51
     52  +     @Internal
     53  +     public static final int CHANNEL_COMMENTS = 2;
     54  +     @Internal
```

andimarek on Jul 25 — Member

lets call this also whitespace channel and not ignored ones to make it consistent with the options.

`src/main/java/graphql/parser/ParserOptions.java`

```
     30  +      * If you want to allow more, then {@link #setDefaultParserOptions(ParserOption
     31  +      * JVM wide.
     32  +      */
```

```
25  34
    33 +     public static final int MAX_WHITESPACE_TOKENS = 200_000;
```

andimarek on Jul 25 — Member

I think we should name this `SDL` parser options or so to make clear it will be used for Schema Parsing. See also my general review comment.

andimarek added this to the **19.0** milestone on Jul 26

`PR feedback - renamed options and added SDL options` ✓ `0bd81e7`

bbakerman merged commit **226aabd** into `master` on Jul 26

1 check passed

View details

---

bbakerman added a commit that referenced this pull request on Jul 26

`READY - Stop DOS attacks by making the lexer stop early on evil input. (` ··· ✓ `1511839`

This was referenced on Jul 26

18.x port - Stop DOS attacks by making the lexer stop early on evil input #2897

Merged

17.x port - Stop DOS attacks by making the lexer stop early on evil input #2898

Closed

17.x port - Stop DOS attacks by making the lexer stop early on evil input #2901

Closed

17.x port - Stop DOS attacks by making the lexer stop early on evil input #2902

Merged

bbakerman added a commit that referenced this pull request on Jul 26

`18.x port - Stop DOS attacks by making the lexer stop early on evil i…` ··· `0256fd0`

This was referenced on Jul 27

Bump com.graphql-java:graphql-java from 18.2 to 19.0 vert-x3/vertx-web#2244

Merged

Bump com.graphql-java:graphql-java from 18.2 to 18.3 vert-x3/vertx-web#2245

Closed

yeikel commented on Jul 28 • edited

Do you have(or are planning to create) a CVE for this?

bbakerman commented on Jul 29 — Member — Author

> Do you have(or are planning to create) a CVE for this?

We don't have concrete plans - mainly because we are unsure of the process and the work involved.

If you know more about this works and could coach us on process that would great because we aren't against the idea, we just know very little about the how of it

act1on3 commented on Aug 3 • edited

UPD. CVE-2022-37734 is assigned.

Hi **@bbakerman** ,
I've requested a CVE with the form: https://cveform.mitre.org/

❤️ 1

dondonz commented on Sep 14 — Member

@act1on3 I'd like to update the CVE with all versions containing the fix: v19.0/19.1/19.2, v18.3, and v17.4.

I've never had to update a CVE before - do you have any way to edit the text from your end?

👍 1

act1on3 commented on Sep 14 • edited

Hi **@dondonz** ,
I believe an analyst from MITRE updated it already. Thanks for info. I also have no idea how to update the description :)

yeikel commented on Sep 14

Considering the different packports, is the range correct? Ie: are users with 17.x versions that have the backport vulnerable?

dondonz commented on Sep 14 Member

Thanks **@act1on3**, I ended up contacting MITRE to fix it.

@yeikel Yes version v17.4 contains the backport, see the release notes https://github.com/graphql-java/graphql-java/releases/tag/v17.4

❤️ 1

yeikel commented on Sep 14

> Thanks **@act1on3**, I ended up contacting MITRE to fix it.
>
> **@yeikel** Yes version v17.4 contains the backport, see the release notes https://github.com/graphql-java/graphql-java/releases/tag/v17.4

If that's the case, then we should explicitly state that in the CVE. Otherwise, scanning systems will flag versions that are safe

dondonz commented on Sep 14 Member

@yeikel I agree, I'm getting automated tickets myself at work. The CVE text has been updated to reference the backported versions.

```
graphql-java before19.0 is vulnerable to Denial of Service. An attacker can send a malicious
GraphQL query that consumes CPU resources. The fixed versions are 19.0 and later, 18.3, and 17.4.
```

https://www.cve.org/CVERecord?id=CVE-2022-37734

I am also speaking to Snyk to fix their recommendation.

❤️ 1

 **hzariv** mentioned this pull request on Sep 21

CVE-2022-37734: Denial of Service (DoS) in graphql-java Netflix/dgs-framework#1238

Closed

Reviewers

 **andimarek**

 **dondonz**

Assignees

No one assigned

Labels

None yet

Projects

None yet

Milestone

19.0

Development

Successfully merging this pull request may close these issues.

None yet

5 participants