# Ruby 1.8 - Bug #2644

## memory over-allocation with regexp

01/25/2010 07:48 PM - ghazel (Greg Hazel)

| | |
|---|---|
| **Status:** | Open |
| **Priority:** | Normal |
| **Assignee:** | |
| **Target version:** | |
| **ruby -v:** | ruby 1.8.7 (2009-06-12 patchlevel 174) [x86_64-linux] |

**Description**

=begin
Using a simple regular expression ruby allocates far too much memory, and can stack overflow.

Code:
```
p 1
s = "2" + (" " * 84149170)
p 2
s.match(/(\d) (.*)/)
p 3
```

Output:
```
1
2
hmm.rb:4:in `match': Stack overflow in regexp matcher: /(\d) (.*)/ (RegexpError)
from hmm.rb:4
```

Stack overflow is not the worst of it. It's actually trying to allocate very large amounts of memory. Here is the output of REE, which prints when malloc tries to grab a lot:

```
1
2
tcmalloc: large alloc 1090519040 bytes == 0x49867000 @
tcmalloc: large alloc 2181038080 bytes == 0x8aa67000 @
tcmalloc: large alloc 18446744072140881920 bytes == (nil) @
tcmalloc: large alloc 4362076160 bytes == (nil) @
hmm.rb:4:in `match': Stack overflow in regexp matcher: /(\d) (.*)/ (RegexpError)
from hmm.rb:4
```

External observation of processes show that this is memory over-allocation occurs across normal builds of 1.8.6, 1.8.7 and even 1.9.1

(Before you say "this is just a problem with regexp in general!", I tested the same thing on python and perl. Both work satisfactorily with even larger strings.)
=end

### History

**#1 - 01/25/2010 11:04 PM - rklemme (Robert Klemme)**

=begin
2010/1/25 Greg Hazel redmine@ruby-lang.org:

> Bug #2644: memory over-allocation with regexp
> http://redmine.ruby-lang.org/issues/show/2644
>
> Author: Greg Hazel
> Status: Open, Priority: Normal
> ruby -v: ruby 1.8.7 (2009-06-12 patchlevel 174) [x86_64-linux]
>
> Using a simple regular expression ruby allocates far too much memory, and can stack overflow.
>
> Code:
> p 1

```
s = "2" + (" " * 84149170)
p 2
s.match(/(\d) (.*)/)
p 3
```

Output:

```
1
2
hmm.rb:4:in `match': Stack overflow in regexp matcher: /(\d) (.*)/ (RegexpError)
    from hmm.rb:4
```

Stack overflow is not the worst of it. It's actually trying to allocate very large amounts of memory. Here is the output of REE, which prints when malloc tries to grab a lot:

```
1
2
tcmalloc: large alloc 1090519040 bytes == 0x49867000 @
tcmalloc: large alloc 2181038080 bytes == 0x8aa67000 @
tcmalloc: large alloc 18446744072140881920 bytes == (nil) @
tcmalloc: large alloc 4362076160 bytes == (nil) @
hmm.rb:4:in `match': Stack overflow in regexp matcher: /(\d) (.*)/ (RegexpError)
    from hmm.rb:4
```

External observation of processes show that this is memory over-allocation occurs across normal builds of 1.8.6, 1.8.7 and even 1.9.1

(Before you say "this is just a problem with regexp in general!", I tested the same thing on python and perl. Both work satisfactorily with even larger strings.)

Question is how much the fix costs and how realistic this is. Even in case there is not a fix, here's an easy workaround for the regular expression you presented: avoid backtracking through greediness (works on 1.9 and JRuby only):

```
15:00:28 Temp$ allruby match-break.rb
CYGWIN_NT-5.1 padrklemme1 1.7.1(0.218/5/3) 2009-12-07 11:48 i686 Cygwin
========================================
ruby 1.8.7 (2008-08-11 patchlevel 72) [i386-cygwin]
match-break.rb:4: nested ?+ in regexp: /(\d) (.+)/
========================================
ruby 1.9.1p376 (2009-12-07 revision 26041) [i386-cygwin]
1
2
3
========================================
jruby 1.4.0 (ruby 1.8.7 patchlevel 174) (2009-11-02 69fbfa3) (Java
HotSpot(TM) Client VM 1.6.0_17) [x86-java]
1
2
3
15:01:13 Temp$ cat match-break.rb
p 1
s = "2" + (" " * 84149170)
p 2
s.match(/(\d) (.*+)/)
p 3
15:01:18 Temp$
```

Kind regards

robert

--
remember.guy do |as, often| as.you_can - without end
http://blog.rubybestpractices.com/

=end

**#2 - 01/27/2010 02:56 PM - naruse (Yui NARUSE)**

*- Category set to core*

*- Priority changed from Normal to 3*

=begin

I think, WONTFIX.
=end

#### #3 - 02/08/2010 05:35 PM - ghazel (Greg Hazel)

=begin
Well, considering other regexp implementations do not have this bug, my guess would be this is either easy or a matter of switching to a better regexp parser, which probably has other benefits.

Anyway, I didn't just think up this edge case. I ran in to it when a developer on a project I'm on wrote a simple little log parser and this parser couldn't ever finish because it kept hanging. So obviously people hit this bug in the wild, and are not aware of it. Familiarity with perl or other regexp environments will not give you the knowledge to know what to avoid, since they do not have this problem.

So, I think it should be fixed.
=end

#### #4 - 02/08/2010 06:15 PM - matz (Yukihiro Matsumoto)

=begin
Hi,

In message "Re: [ruby-core:28104] [Bug #2644] memory over-allocation with regexp"
on Mon, 8 Feb 2010 17:36:07 +0900, Greg Hazel redmine@ruby-lang.org writes:

|Well, considering other regexp implementations do not have this bug, my guess would be this is either easy or a matter of switching to a better regexp parser, which probably has other benefits.

I don't know any other regex implementation that allows the current
feature set with M17N available.  Do you?

We just cannot abandon the current implementation to fix this
relatively minor issue.  Besides that, I'm afraid most DFA regex
matcher would have this issue anyway.

```
                          matz.
```

=end

#### #5 - 02/08/2010 09:36 PM - murphy (Kornelius Kalnbach)

=begin
On 08.02.10 09:36, Greg Hazel wrote:

> Issue #2644 has been updated by Greg Hazel.
> I have a nicer workaround, see below.
>
> Well, considering other regexp implementations do not have this bug,
> my guess would be this is either easy or a matter of switching to a
> better regexp parser, which probably has other benefits.
> I think Oniguruma (1.9) is a great Regexp engine. We could redirect the
> issue to the author...but that won't fix 1.8. Maybe the problem is on
> the Ruby side? JRuby seems to be affected, too:

```
1
2
Error: Your application used more memory than the safety cap of 500m.
Specify -J-Xmx####m to increase it (#### = cap size in MB).
Specify -w for full OutOfMemoryError stack trace
```

(The string is only 84 MB.)

On 25.01.10 15:03, Robert Klemme wrote:

> Question is how much the fix costs and how realistic this is.  Even
> in case there is not a fix, here's an easy workaround for the regular
> expression you presented: avoid backtracking through greediness
> (works on 1.9 and JRuby only): /(\d) (.*+)/
> Question is: Why would it backtrack? It doesn't need to. The expressions
> are equivalent for this string.

Also, both these expressions need to backtrack, and they work even for
larger strings:

p ' '.*(200_000_000)[/.?$/]*.size  #=> 200000000
p ' '.*(200_000_000)[/^?$/]*.size  #=> 200000000

Even this one works:

p ' '.*(200_000_000)[/(.)/,1]*.size  #=> 1

All of the above scripts only need about 200 MB of RAM.

Just this breaks it, trying to allocate over 2GiB:

p ' '.*(200_000_000)[/./]*.size
ruby(81051) malloc: *** mmap(size=2348810240) failed (error code=12)
*** error: can't allocate region
*** set a breakpoint in malloc_error_break to debug
test.rb:1:in `[]': Stack overflow in regexp matcher: /.*/ (RegexpError)
from test.rb:1

Same with + and {0,}. This seems to be a workaround (all Ruby versions):

p ' '.*(200_000_000)[/(?:.)/]*.size  #=> 200000000

Ruby is definitely doing something strange here in the background.

> So, I think it should be fixed.
> So do I. Actually, how can it be a *Stack overflow*? Quirks like this
> tend to hint at bugs or possible optimizations.

Thanks for reading.
[murphy]

=end

#### #6 - 02/08/2010 09:37 PM - murphy (Kornelius Kalnbach)

=begin
On 08.02.10 13:26, Kornelius Kalnbach wrote:

> This seems to be a workaround (all Ruby versions):
> p ' '.*(200_000_000)[/(?:.)/]*.size  #=> 200000000
> Sorry, this one doesn't work in JRuby and Ruby 1.9.

[murphy]

=end

#### #7 - 02/08/2010 09:56 PM - naruse (Yui NARUSE)

=begin
Current Ruby 1.9 can run the script,

(2010/02/08 18:15), Yukihiro Matsumoto wrote:

> In message "Re: [ruby-core:28104] [Bug #2644] memory over-allocation with regexp"
> on Mon, 8 Feb 2010 17:36:07 +0900, Greg Hazel redmine@ruby-lang.org writes:
>
> |Well, considering other regexp implementations do not have this bug,
> my guess would be this is either easy or a matter of switching to a better regexp parser,
> which probably has other benefits.
>
> I don't know any other regex implementation that allows the current
> feature set with M17N available.  Do you?
>
> We just cannot abandon the current implementation to fix this
> relatively minor issue.  Besides that, I'm afraid most DFA regex
> matcher would have this issue anyway.

Current trend, Irregexp doesn't be suited for M17N.

TRE seems M17N and can be compiled with VC++.
http://laurikari.net/tre/
But its internal presentation is wide character,

so conversion is needed on matching.

--
NARUSE, Yui naruse@airemix.jp

=end

#### #8 - 02/08/2010 11:48 PM - mame (Yusuke Endoh)

=begin
Hi,

2010/2/8 Greg Hazel redmine@ruby-lang.org:

> So, I think it should be fixed.

I agree. Could you please give us a patch? :-P

I guess the last .* node can be optimized by translating into (?>.*),
but I'm not at all sure that this patch has no bug. Do anyone review
this?

diff --git a/regcomp.c b/regcomp.c
index e1ac7ee..16c816b 100644
--- a/regcomp.c
+++ b/regcomp.c
@@ -3643,6 +3643,7 @@ setup_comb_exp_check(Node* node, int state, ScanEnv* env)
#define IN_NOT (1<<1)
#define IN_REPEAT (1<<2)
#define IN_VAR_REPEAT (1<<3)
+#define IN_LAST (1<<4)

/* setup_tree does the following work.

1. check empty loop. (set qn->target_empty_info) @@ -3664,7 +3665,8 @@ setup_tree(Node* node, regex_t* reg, int state, ScanEnv* env) { Node* prev = NULL_NODE; do {
   - r = setup_tree(NCAR(node), reg, state, env);
   - int s = IS_NOT_NULL(NCDR(node)) ? (state & ~IN_LAST) : state;
   - r = setup_tree(NCAR(node), reg, s, env); if (IS_NOT_NULL(prev) && r == 0) { r = next_setup(prev, NCAR(node), reg); } @@ -3795,6 +3797,20 @@ setup_tree(Node* node, regex_t* reg, int state, ScanEnv* env) } } #endif +
   - if ((state & IN_LAST) != 0 && qn->greedy && IS_REPEAT_INFINITE(qn->upper)) {
   - /* automatic posseivation a* (at last) ==> (?>a*) */
   - if (qn->lower <= 1) {
   - int ttype = NTYPE(qn->target);
   - if (IS_NODE_TYPE_SIMPLE(ttype)) {
   - Node* en = onig_node_new_enclose(ENCLOSE_STOP_BACKTRACK);
   - CHECK_NULL_RETURN_MEMERR(en);
   - SET_ENCLOSE_STATUS(en, NST_STOP_BT_SIMPLE_REPEAT);
   - swap_node(node, en);
   - NENCLOSE(node)->target = en;
   - }
   - }
   - } } break;

@@ -5423,7 +5439,7 @@ onig_compile(regex_t* reg, const UChar* pattern,
const UChar* pattern_end,
reg->num_call = 0;
#endif

- r = setup_tree(root, reg, 0, &scan_env);
- r = setup_tree(root, reg, IN_LAST, &scan_env); if (r != 0) goto err_unset;

#ifdef ONIG_DEBUG_PARSE_TREE

--
Yusuke ENDOH mame@tsg.ne.jp

=end

#### #9 - 02/09/2010 11:18 PM - matz (Yukihiro Matsumoto)

=begin
Hi,

In message "Re: [ruby-core:28111] Re: [Bug #2644] memory over-allocation with regexp"
on Mon, 8 Feb 2010 23:48:03 +0900, Yusuke ENDOH mame@tsg.ne.jp writes:

|I guess the last .* node can be optimized by translating into (?>.*),
|but I'm not at all sure that this patch has no bug.  Do anyone review
|this?

Can you check in?  If something bad happens, we can always revert the
modify.

```
                          matz.
```

=end

#### #10 - 11/25/2010 01:48 PM - shyouhei (Shyouhei Urabe)

=begin
Hi, while checking the ITS I noticed this issue is remain open.  What's its situation?  Can I close this?
=end

#### #11 - 11/25/2010 08:28 PM - mame (Yusuke Endoh)

=begin
Hi,

2010/11/25 Shyouhei Urabe redmine@ruby-lang.org:

> Hi, while checking the ITS I noticed this issue is remain open.  What's its situation?  Can I close this?

- My patch was once applied at r26701.
- It was turned out to be buggy (Bug #3681).
- It was reverted at r29023.  <- here now

--
Yusuke Endoh mame@tsg.ne.jp

=end