jmrozanec / cron-utils Public

Code | Issues 14 | Pull requests | Actions | Projects | Wiki

New issue

Jump to bottom

Fix security vulnerability regarding the Hibernate dependency. #461

Closed jmrozanec opened this issue on Nov 21, 2020 · 23 comments · Fixed by #494

Assignees

Labels done enhancement

Milestone 9.1.3

jmrozanec commented on Nov 21, 2020 Owner

No description provided.

jmrozanec added enhancement done labels on Nov 21, 2020

jmrozanec self-assigned this on Nov 21, 2020

jmrozanec added this to the **next** milestone on Nov 21, 2020

jmrozanec closed this as completed on Nov 21, 2020

jmrozanec modified the milestones: **next**, **9.1.3** on Nov 21, 2020

jmrozanec commented on Nov 22, 2020 • edited Owner Author

A Template Injection was identified in cron-utils enabling attackers to inject arbitrary Java EL expressions, leading to unauthenticated Remote Code Execution (RCE) vulnerability.

Remote Code Execution - JavaEL Injection
If developers use the **@Cron** annotation to validate a user-controlled Cron expression, attackers will be able to inject and run arbitrary Java Expression Language (EL) expressions.

Cron-Utils uses Java Bean Validation (JSR 380) custom constraint validators such as CronValidator. When building custom constraint violation error messages, it is important to understand that they support different types of interpolation, including Java EL expressions. Therefore if an attacker can inject arbitrary data in the error message template passed to ConstraintValidatorContext.buildConstraintViolationWithTemplate(), they will be able to run arbitrary Java code. Unfortunately, it is common that validated (and therefore, normally untrusted) bean properties flow into the custom error message. In this case CronValidator includes the Cron expression being validated in the custom constraint error validation message if an exception is thrown while parsing the expression:

```
public boolean isValid(String value, ConstraintValidatorContext context) {
    if (value == null) {
        return true;
    }

    CronDefinition cronDefinition = CronDefinitionBuilder.instanceDefinitionFor(type);
    CronParser cronParser = new CronParser(cronDefinition);
    try {
        cronParser.parse(value).validate();
        return true;
    } catch (IllegalArgumentException e) {
        context.disableDefaultConstraintViolation();
        context.buildConstraintViolationWithTemplate(e.getMessage()).addConstraintViolation();
        return false;
    }
}
```

gsmet commented on Nov 23, 2020 Contributor

Hi,

Hibernate Validator lead here.

So I have no idea if Hibernate Validator is the right thing for your usage, my personal opinion is that it might be too much infrastructure for a focused util library like yours.

But as far as security is concerned, I thought that **@pwntester** had added the important information I gave to him to the message he is sending. **@pwntester**, I already asked once, could you make sure to provide all relevant information to the people you contact? Thanks!

What you're doing is effectively unsafe but there are multiple ways to make it perfectly safe and still using HV:

- see an example here on how to include a properly escaped variable in your constraint violation message: https://in.relation.to/2020/05/07/hibernate-validator-615-6020-released/ . Basically, you have to consider user input as unsafe, exactly the same as for SQL injection and properly escape it.
- also you can get entirely rid of the javax.el/jakarta.el dependency by initializing your ValidatorFactory with:

```
ValidatorFactory validatorFactory = Validation.byDefaultProvider()
                                .configure()
                                .messageInterpolator( new ParameterMessageInterpolator() )
                                .buildValidatorFactory();
```

Note that the default messages of a couple of built-in constraints will be broken if you choose this option over the first one (typically the *Min, *Max constraints that are using EL for their default message). My advice would be to also properly escape things as recommended in the first option anyway.

I'm still thinking about a way to have a more secure default while keeping the *Min/*Max contraints working properly (and the Jakarta Bean Validation spec requires Jakarta EL support). It's not an easy one.

I hope this clarifies things for you.

pwntester commented on Nov 24, 2020 Contributor

Thanks for the feedback **@gsmet**
We are sending your recommendations as part of the initial report we send to affected projects and also as part of the blog post reference that we share with maintainers.

gsmet commented on Nov 24, 2020 Contributor

Also, I'm far from being sure that Apache BVal is safer if you have javax.el around, which you still have in your pom.

I had a quick look at the code and I think they automatically enable EL if it's in the classpath.

pwntester commented on Nov 25, 2020 Contributor

as far as i know, their code to decide if EL expressions should be evaluated is this which seems to check only for the configuration property being enabled or the message being the default one

gsmet commented on Nov 25, 2020 • edited Contributor

I fail to see how changing the test impl makes `CronValidator` safer.

Because first, HV is the most widely used implementation and second, even in Apache BVal, you can enable EL for custom constraint violation message, which can be done for very good reasons and a user might think all your constraints are safe because carefully audited.

And in this case, having your `CronValidator` around will make the application vulnerable.

What you need to do is escape `e.getMessage()` with an approach similar to what is done here: https://github.com/hibernate/hibernate-validator/blob/master/engine/src/main/java/org/hibernate/validator/internal/engine/messageinterpolation/util/InterpolationHelper.java .

There are better Hibernate Validator-specific way to do this but if you want to be compatible with all the implementations and be entirely safe, that's the only way to do it right now.

dsuresh-ap commented on Dec 10, 2020

@jmrozanec Do you know if this vulnerability is an issue in 6.0.6? Fossa filed this vulnerability but I don't see any related code in v6.0.6 so we are assuming that this is not an issue in that version. Would be great if you can confirm.

NielsDoucet commented on Oct 21, 2021 Contributor

As the original contributor of the validator, I'm confused about what was done to mitigate the problem, as well as how the code was deemed vulnerable in the first place.

As far as I can tell, the change to swap out the test dependency does nothing for the runtime behaviour of this library. So I fail to see how this mitigates the risk.

On the validity of the report: there is no user input for this constraint violation message. It's simply taking the message from the thrown exception and using it as the violation message. So the `CronParser` implementation class has to be swapped out completely and then a maliciously crafted exception message can be thrown. This is technically possible, but at that point this validator is the least of your worries, imho.

@pwntester @jmrozanec @gsmet am I wrong in my assessment here?

pwntester commented on Oct 21, 2021 Contributor

> On the validity of the report: there is no user input for this constraint violation message. It's simply taking the message from the thrown exception and using it as the violation message.

`CronParser` reflects the cron job string in the exception message, so controlling the cron string, an attacker can control the exception message and trigger an EL evaluation. You can find a PoC in the advisory

👀 1

NielsDoucet commented on Oct 21, 2021 Contributor

I see, I completely missed that part (obviously). In that case, shouldn't this library still be considered vulnerable to the problem? Was the fix validated?

pwntester commented on Oct 21, 2021 Contributor

I think the change to bval was considered a secure fix back then and we missed that it was a test dependency. I think its worth verifying it with the latest version. In the meantime, Hibernate has disabled EL evaluation by default on custom constraint error messages

jmrozanec commented on Oct 21, 2021 Owner Author

@pwntester is the current version considered safe, or there are any actions required on our side?

gsmet commented on Oct 21, 2021 Contributor

Hibernate Validator lead here, 6.2 (`javax.` packages) and 7.0 (`jakarta.` packages) are safe.

pwntester commented on Oct 21, 2021 Contributor

I just verified that if I upgrade my testcase to use 9.1.5 and keep the rest of the dependencies the same (hibernate-validator 6.1.6.Final), the application is still vulnerable:

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>mygroupid</groupId>
  <artifactId>myartifactid</artifactId>
  <version>0.0-SNAPSHOT</version>

  <properties>
      <maven.compiler.target>1.8</maven.compiler.target>
      <maven.compiler.source>1.8</maven.compiler.source>
  </properties>
  <dependencies>
    <dependency>
      <groupId>com.cronutils</groupId>
      <artifactId>cron-utils</artifactId>
      <version>9.1.5</version>
    </dependency>
    <dependency>
      <groupId>javax.validation</groupId>
      <artifactId>validation-api</artifactId>
      <version>2.0.1.Final</version>
    </dependency>
    <dependency>
      <groupId>org.hibernate</groupId>
      <artifactId>hibernate-validator</artifactId>
      <version>6.1.6.Final</version>
    </dependency>
    <dependency>
      <groupId>org.glassfish</groupId>
      <artifactId>javax.el</artifactId>
      <version>3.0.0</version>
    </dependency>
  </dependencies>
</project>
```

So the vulnerability is now dependent on what validator version is used at runtime which is not ideal.

@jmrozanec see the remediation section in this blog post

NielsDoucet commented on Oct 21, 2021 Contributor

From my perspective, there are 2 possible routes:

- do not reflect the cron expression back through the exception message at all (this makes the usage of the message safe in all possible contexts)
- escape the exception message before passing it to the violation as described here:

> What you need to do is escape `e.getMessage()` with an approach similar to what is done here: https://github.com/hibernate/hibernate-validator/blob/master/engine/src/main/java/org/hibernate/validator/internal/engine/messageinterpolation/util/InterpolationHelper.java .

This library has no direct influence on the validator framework used, so we can only make sure the message is safe.

jmrozanec commented on Oct 21, 2021 Owner Author

@NielsDoucet perhaps the first option (not reflecting the expression back in the exception message) is a good choice? We would be grateful for a PR providing that fix. Is that possible?

pwntester commented on Oct 22, 2021 • edited Contributor

@jmrozanec This should fix it

jmrozanec commented on Oct 22, 2021 Owner Author

@pwntester thanks! 😄 That was fast! 🚀

pwntester commented on Oct 22, 2021 Contributor

that was easy too 😄 I guess we should issue a new advisory and CVE to let user know about the security issue in <9.1.5

jmrozanec commented on Oct 22, 2021 Owner Author

@pwntester yes, please issue an advisory. We will release a new version over the weekend, so we ensure the fix can be incorporated ASAP by anyone. Thanks! 😄

pwntester commented on Oct 22, 2021 Contributor

I dont have permissions on this repo to create a security advisory. Please create one and invite me in and I will fill in the details

NielsDoucet commented on Oct 23, 2021 Contributor

> **@jmrozanec** This should fix it

I was thinking more about fixing it in the actual parser: https://github.com/jmrozanec/cron-utils/blob/master/src/main/java/com/cronutils/parser/CronParser.java#L131
Technically reflecting the input back in the exception could lead to other vulnerabilities by anyone using this exception message in a context where it's evaluated. We would avoid this problem altogether by removing the reflection at the source.

```
throw new IllegalArgumentException(String.format("Failed to parse '%s'. %s", expression, e.getMessage()), e);
```

could become

```
throw new IllegalArgumentException(String.format("Failed to parse the cron expression. %s", e.getMessage()), e);
```

It would at least provide more feedback from the validator than a generic "something went wrong".
@jmrozanec does that sound good?

1

NielsDoucet added a commit to NielsDoucet/cron-utils that referenced this issue on Oct 23, 2021

`Resolve RCE vulnerability.` ··· `9c93c17`

NielsDoucet mentioned this issue on Oct 23, 2021

Resolve RCE vulnerability. #494

Merged

jmrozanec commented on Nov 14, 2021 Owner Author

@pwntester just created the security advisory and added you as a collaborator. I used the same description as for the previous one, only updating the library version for the mitigation. Can you check what else is required to finalize this?
@NielsDoucet I updated the code and released a new version of the library. The update removed the change introduced by **@pwntester** and no other change was made. Should be this ok?

Thank you both for the help, and my apologies for the delay 😄 Thanks!

1

igalbk mentioned this issue on Nov 30, 2021

Critical CVSS 9.8 CVE-2021-41269 Vulnerability in cron-utils dependency kagkarlsson/db-scheduler#250

Closed

Assignees

jmrozanec

Labels

done enhancement

Projects

None yet

Milestone

9.1.3

Development

Successfully merging a pull request may close this issue.

Resolve RCE vulnerability.
NielsDoucet/cron-utils

5 participants