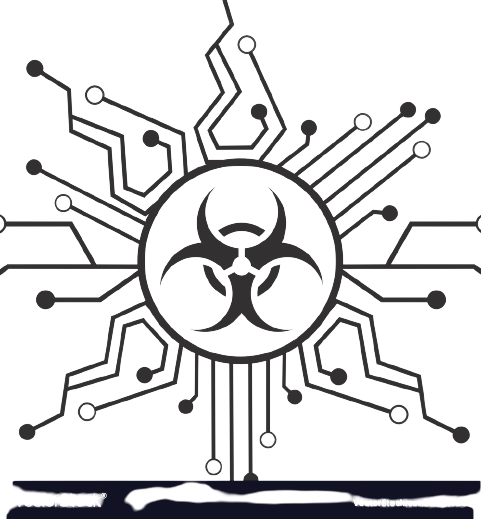
****

**Log4j POC Shell Exploit.**

September 2022

# 

## **For**

## Client

## 

## **REPRESENTATIVE**

Muhammad Sarwar

**Demonstration of POC:**

Basically, PoC exploits are done by a vendor working for the company. Simulating an actual attack, allows the company to patch the security hole without systems or data being compromised. The code developed for the test (proof of concept code) will likely be used in the future to test the software and make sure the new security measures work.

### **PoC code and zero-day exploits**

PoC code is a term used to describe a code that was developed to demonstrate security flaws in software or networks during a PoC exploit. IT departments use it to simulate attacks to identify vulnerabilities and patch them. PoC code can also be used to determine a threat level. When PoC code is published before the security hole is patched, a [zero-day](https://www.techtarget.com/searchsecurity/definition/zero-day-vulnerability) exploit can occur. Zero-day exploits are malicious attacks that occur after a security risk is discovered but before it is patched. Publishing PoC code has become controversial because it can lead to zero-day exploits when it is released too quickly. When a PoC code is shared before a weakness is patched, it leaves software and networks vulnerable to hackers. Many large companies have recently fallen victim to security breaches due to shared PoC codes before issues can be fixed.

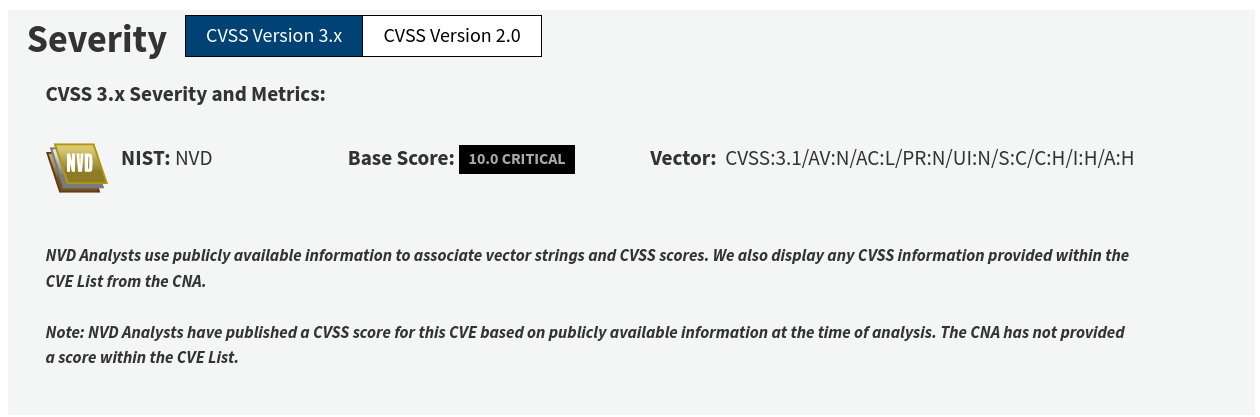
**Log4j Shell POC exploits:**

On December 9, 2021, security researchers discovered a flaw in the code of a software library used for logging. The software library, **Log4j**, is built on a popular coding language, Java, that has widespread use in other software and applications used worldwide. This flaw in Log4j is estimated to be present in over 100 million instances globally.

The flaw, also known as a *vulnerability* by the security community, was rated a 10 out of 10 on the Common Vulnerability Scoring System, or CVSS, due to the potential impact that it can have if leveraged by attackers. Details of the vulnerability can be found in the National Vulnerability Database (NVD) under the heading [CVE-2021–44228](https://nvd.nist.gov/vuln/detail/CVE-2021-44228).

**CVE-2021-44228 Detail**

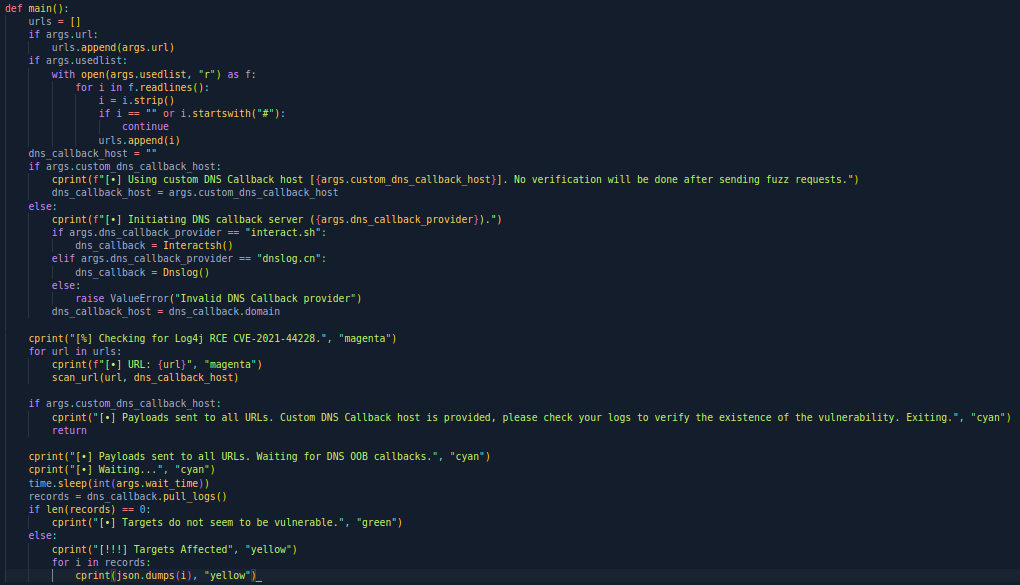
Apache Log4j2 2.0-beta9 through 2.15.0 (excluding security releases 2.12.2, 2.12.3, and 2.3.1) JNDI features used in the configuration, log messages, and parameters do not protect against attacker-controlled LDAP and other JNDI related endpoints. An attacker who can control log messages or log message parameters can execute arbitrary code loaded from LDAP servers when message lookup substitution is enabled. From log4j 2.15.0, this behavior has been disabled by default. From version 2.16.0 (along with 2.12.2, 2.12.3, and 2.3.1), this functionality has been completely removed. Note that this vulnerability is specific to log4j-core and does not affect log4net, log4cxx, or other Apache Logging Services projects.



**Introduction of Script:**

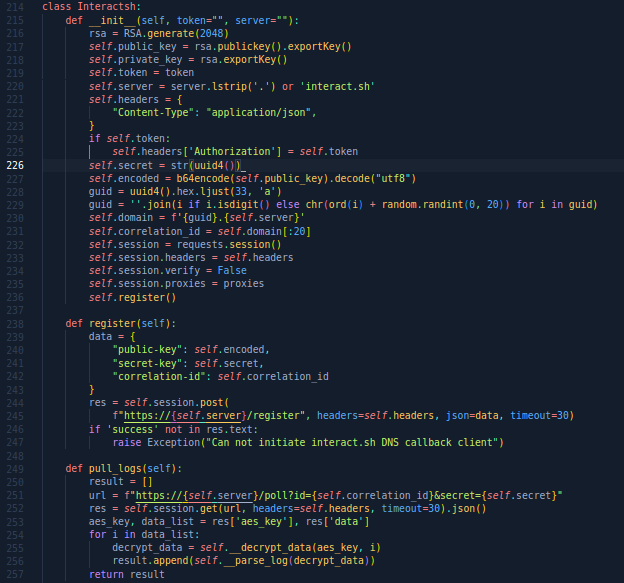
Our Python script is performing threading, sub-processing, and server requests in order to perform log4j Shell POC, Also this program is only coded in python using different libraries. This program consists of two different CVEs the CVE-2022-42889 which is able to exploit the RCE (Remote Code Execution) and CVE-2021-45046 which is Apache Log4j 2.15.0 vulnerability. Since the script is written in python it can work on multiple platforms with any error.

**Demonstration of Coding:**

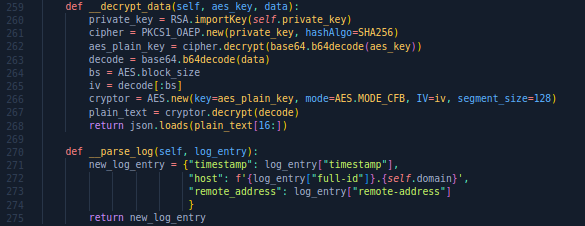
****

First of all, we are going to learn about argument parse which is Python Library so users are able to enter flexible input in order to perform operations, So this main function is defining empty variable name urlswith the list data type. Then with the argument parse, we are checking if the user uses the single url or a list of different urls then we will append the given url in the urls list variable. Then we declare the dns\_callback\_hos string variable which will be updated if the uses the costume DNS callback otherwise the interact.sh will be the default DNS callback or the user has the option to choose the dnslog.cn callback. If the user chooses to interact.sh DNS callback options the Interactsh() class object will be saved in the dns\_callback variable otherwise Dnslog() class object will be saved in the dns\_callback variable. Then the for loop will iterate over the urls variable and send the url and callback variable as arguments to scan\_url() function.

* **Interactsh():** This is a class defined in our script this class object is saved in the dns\_callback variable by default. This class has a total of 5 functions in it.
  + \_\_init\_\_()
  + register()
  + pull\_logs()
  + \_\_decrypt\_data()
  + \_\_parse\_log()



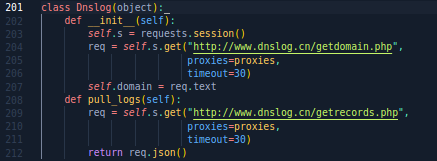
We are exporting the public and private from the interact.sh DNS callback in order to communicate with the server. Then we register with a public key which is encoded in base64 and stored in self.encoded and secret key and with correlation\_id with the help of this data we make a request to the server port if and will wait for 30 seconds, if the server didn’t respond we will print the error message on the screen. If the server responds in time and makes a session then by using dns\_callback variable which holds the object of interactsh() class will call pull\_logs in the main function of our script. By combining the server name, correlation\_id, and secret key we get the data from the server but the data is encrypted, so that’s why we send the data in the protected function name \_\_decrypt\_data() with the data variable and with AES key.



The \_\_decrpyt\_data() functions receive the two arguments aes\_key and data from pull\_logs() functions. This function import the private key from the \_\_init\_\_() function from the RSA library and saves it in the variable, we import the new private key from PKCS1\_OAEP with the SHA256 hashing algorithm and save it in the variable name cipher. Now using the cipher variable as an object will call the decrypt() function by sending aes\_key as an argument which will convert into the base64 plaintext and store it in the aes\_plain\_text variable. Now we covert the data variable into base64 but it will not be readable for humans, So using the AES library we will call the new() built-in function we will provide the plain text AES key store in aes\_plain\_key variable, set the mode to AES and segment size to 128 and store this configuration in the cryptor variable which will decode the data variable content to plain text. And return the plain\_text variable which holds the data variable content.

Now coming back to the pull\_logs() function we will send the decrypt() function data to the protected \_\_parse\_log() function as an argument. This function will make a dictionary variable name new\_log\_entry and we will store only the Hostname, Address, and timestamp which was obtained from the server and saved in the data variable. Which will be printed on the screen.

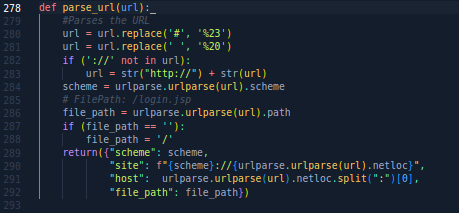
* **Dnslog():** This class has only two functions init.
  + \_\_init\_\_()
  + pull\_logs()



The \_\_init\_\_() function makes a request to build a session with the given domain and will wait 30 seconds for the response and for the response in the res variable and then converting the content of res into the text we will save the data in the new variable call domain. Then in the pull\_logs() function we will send the get request to the given domain and save the data in the req variable and then return the req variable data by converting it into JSON data type.

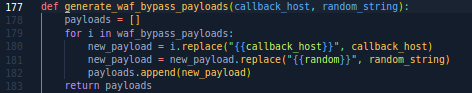
Before moving toward the scan\_url() function let's have a look at the different small functions listed below. These functions are used inside the scan\_url():

* parse\_url()
* generate\_waf\_bypass\_payload()
* get\_cve\_2021\_45046\_payloads()
* get\_cve\_2022\_42889\_payloads()
* **parse\_url():**

****

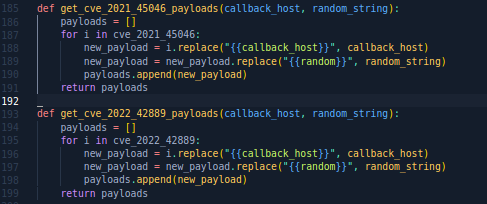
This function is taking the url and replacing the ‘#’ and empty space with ‘%23’ and ‘%20’ and checking the ‘http://’ is available in the given url or not if not then it will add the ‘http://’ at the start of url. After replacing the sting in the url this function is changing the path to ‘/’ and returns the scheme it gets the form url, manipulated site url, hostname, and file path.

* **generate\_waf\_bypass\_payload():**

****

This function takes two arguments and creates an empty list variable name payloads and starts a for loop and staring replacing the call host and random string and then appends both in the payloads variable and returns it.

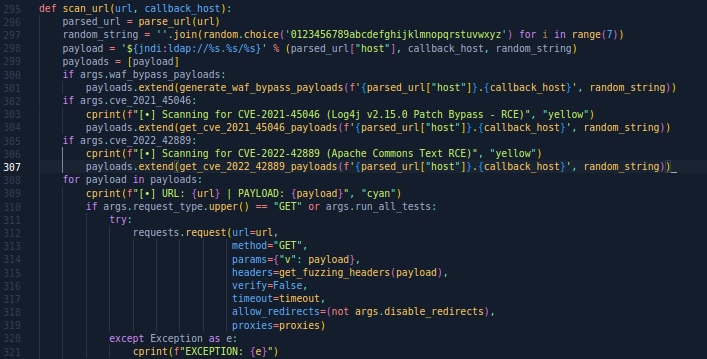
* **get\_cve\_2021\_45046\_payloads() and get\_cve\_2022\_42889\_payloads():**

****

Both CVE functions are doing the same thing as we discuss in the generate\_waf\_bypass\_payload() functions and returning the result to the function as it can use further in the script.

* **scan\_url():**

This function is accepting two arguments url and callback\_host which come from the main() function url and the url of the target or victim and callback\_host has the data of either interactsh or Dnslog class.



The url send to the parse\_url() function which is explained above then a random string is joined with the empty string with a random built-in function and makes 7 character random string and stored in the random\_string variable.

Now the random string, callback\_host data, and parsed\_url are joined together in the payload variable then pay is sent to the generate\_waf\_bypass\_payload() function to generate the payload for bypass and a similar thing is done if the user chooses a different option in the argument parser. After that, we run a for loop we see the ‘GET’ or ‘POST’ in the request and update the payload if the word is found. Otherwise, an expectation(error) will raise(shown on the terminal Screen).

If the pull\_log() function establishes a session with either DNS or User Custom DNS list it will print on the screen whether the vulnerability is found or not.

This script is completely written in python. Which means it’s a 100% Python Language Project. There is no other language is used in the project. This script is not saving any kind of data or load any kind of data from a file to read. Yes, Script does some data conversion in JSON but does not save that in the file. The script is fairly simple and does add any kind of hard concepts to it. I’m using my script against a vulnerable web app that runs on localhost and this vulnerable machine is open-source and available on GitHub to use. I have tested my script only on linux and it is running without any errors.