SERIALIZACJA

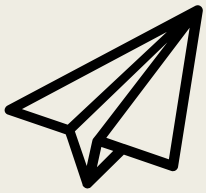
Czyli jak sobie radzić z przesyłaniem danych

Przygotował
Paweł Jan Tłusty

Co kryje się za słowem serializacja?

Koncept jest bardzo prosty :

- Bierzesz jakiś obiekt
- Wydobywasz jego dane
- Zamykasz to w tekstowej albo binarnej reprezentacji
- Zapisujesz żeby go w przyszłości użyć



Jeśli okaże się, ze jednak potrzebujesz użyć tego obiektu to wczytujesz te dane, „parsujesz" i na podstawie tego rekonstruujesz obiekt

Analizator składniowy (Parsing)

Analizator składniowy, parser – program dokonujący analizy składniowej danych wejściowych w celu określenia ich struktury gramatycznej w związku z określoną gramatyką formalną.

TYLKO NA CZYM POLEGA NASZ PROBLEM?

No bo wiecie, skoro jest serializacja to i istnieje deserializacja. Czyli te obiekty to tak jednak fajnie by było przesyłać pomiędzy maszynami. I tu jak na razie wszystko brzmi fajnie. Tylko jest taki jeden mały szkopuł, jak na razie mówimy o komunikacji dwóch maszyn. I teraz do tego układu wprowadzamy trzecią stronę – człowieka. I tutaj nadchodzi prawda objawiona, Deserializacja danych pochodzących od użytkownika potrafi być przyczyną problemów z bezpieczeństwem niemal w każdym języku programowania.(w zasadzie potrafi umożliwić wykonanie dowolnego kodu np. po stronie serwera, nie najlepiej brzmi, prawda?)

{JSON}	 python
▪ Jest popularniejszy ▪ Nie jest przypisany do jednego języka programowania ▪ W Pythonie może tworzyć tylko ,,proste” obiekty (tj.string,liczby,listy)	▪ Daje większe możliwości niż przeciętny format serializacja ▪ Jest bardzo prosty w obsłudze ▪ Umożliwia przesyłanie dowolnego obiektu

Tutaj pod obiektu kryje się klasa, metoda, referencja, praktycznie wszystko co znajdzie się w Pythonie

A i pamiętajmy o tym, że te rozważania dokonujemy patrząc przez pryzmat pracy w Pythonie. W innej implementacji JSON może okazać się najlepszym wyborem.

Co musimy wiedzieć o tym Pickle?

W zasadzie do użytku codziennego wystarczy nam poznać dwie jego metody:

- dumps – zwraca obiekt otrzymany jako argument w postaci zserializowanej
- loads- przyjmuje argument typu bytes i odtwarza obiekt

```
1   import pickle,datetime
2
3
4
5   now= datetime.datetime.now()
6
7   print(now)
8
9   pickled = pickle.dumps(now)
10
11  print(pickled)
12
13  print(pickle.loads(pickled))
14
```

```
2022-06-09 12:52:33.748390
b'\x80\x03cdatetime\ndatetime\nq\x00C\n\x07\xe6\x06\t\x0c4!\x0bkfq\x01\x85q\x02Rq\x03.'
2022-06-09 12:52:33.748390
    0: \x80 PROTO     3
    2: c    GLOBAL    'datetime datetime'
   21: C    SHORT_BINBYTES b'\x07\xe6\x06\t\x0c4!\x0bkf'
   33: \x85 TUPLE1
   34: R    REDUCE
   35: .    STOP
highest protocol among opcodes = 3
None
```

W sumie to wartałoby wspomnieć że trzymamy te dane w formie byte albo string, ale dla nas byte
Tutaj zrobim w ogóle o Pickle slajd co to to jest jak działa etc.

Co zwróciła nam analiza?

Od lewej:

- Numer bajtu z wejściowego ciągu bajtów
- Bajt reprezentujący dany opcod
- Nazwa danego opcodu.
- Argumenty opcodu

```
    0: \x80 PROTO      3
    2: c    GLOBAL     'datetime datet
ime'
   21: C    SHORT_BINBYTES b'\x07\xe6\
x06\t\x02\x0e\x00\x00/\x01'
   33: \x85 TUPLE1
   34: R    REDUCE
   35: .    STOP
highest protocol among opcodes = 3
None
```

PROTO – werjsa protokołu uużytego przezpickle
GLOBAL – wrzuca na stos referencję do konstruktora klasy datetime.datetime, czy czego my tam uzyjem
SHORT_BINBYTES wrzuca na stos ciąg bajtów
TUPLE1 nam to krotkuje(dzieli każdy bajt na osobną ktrotkę) – tu jest fajny moment żeby podpytać o neimutowalnosć krotki i jej pochodzenie
REDUCE – pobiera ze stosu dwa elementyL krtokę która jest lisą argumentów i referncję do funkcji, którą po chwili wykonouje, a jej wartość wrzuca na stos.
STOP – musi się pojawić i odkrywczo oznacza koniec przetwarzania obiektu

Tak w ogóle wartałoby wspomnieć, że do analizowania obiektów służy moduł pickletools i metoda dis.

A wynik tego polecenia powstał przy pomocy:
Print(pickletools.dis(pickletools.optimize(pickled)))

A teraz jak możemy to wykorzystać?

Zbudujemy własny obiekt

Odpowiedź: zbudujemy sobie własny obiekt

RTFM stands oczywiście for:
Read
The
Fajny
Manual

Jak nic chodziło im o fajny. Bo to dokładnie to słowo na f przychodzi nam na myśl jako pierwsze.

```
Get Started | text_reminder.py | pickle_show.py | pickle_tools.py | picke_live_inj.py
pickle_module > picke_live_inj.py
1  import pickletools, pickle
2
3
4  print(pickletools.dis(b'\x80\x03cos\nsystem\nS"id"\n\x85R.'))
5
6  print(pickle.loads(b'\x80\x03cos\nsystem\nS"touch mnie_tu_nie_powinno_byc.txt"\n\x85R.'))
```

```
[kotmin@localhost pickle_module]$ /bin/python3 /home/kotmin/git_sem/pickle_module/picke_live_inj.py
    0: \x80 PROTO      3
    2: c    GLOBAL     'os system'
   13: S    STRING     'id'
   19: \x85 TUPLE1
   20: R    REDUCE
   21: .    STOP
highest protocol among opcodes = 2
```

Wynik polecenia z linii 4.

Tutaj akurat wrzuciłem wynik wykonania 4 linii kodu. Widzimy, że w sekcji global oraz sekcji string umieściliśmy odwołanie do metody os_system(która pozwala na używanie terminala, przy pomocy pythona, oraz polecenia konsolowego id.

W 7 linii przekazaliśmy touch z argumentami, jaki wniosek? Tak możemy przekazać wszystko, zyskujemy dodatkową możliwość łączenia się z urządzeniem.

JAK SIĘ PRZED TYM BRONIĆ?

Rozwiązaniem jest … nie używać deserializacji. Co prawda nie jedynym, ale niwelującym te problemy.

Nawet twórcy Pickle w swojej oficjalnej dokumentacji piszą otwarcie, że ichj moduł nie jest bezpieczny. I żeby nie odpakowywać danych z nieznanych źródeł.

Mamy związane ręce?

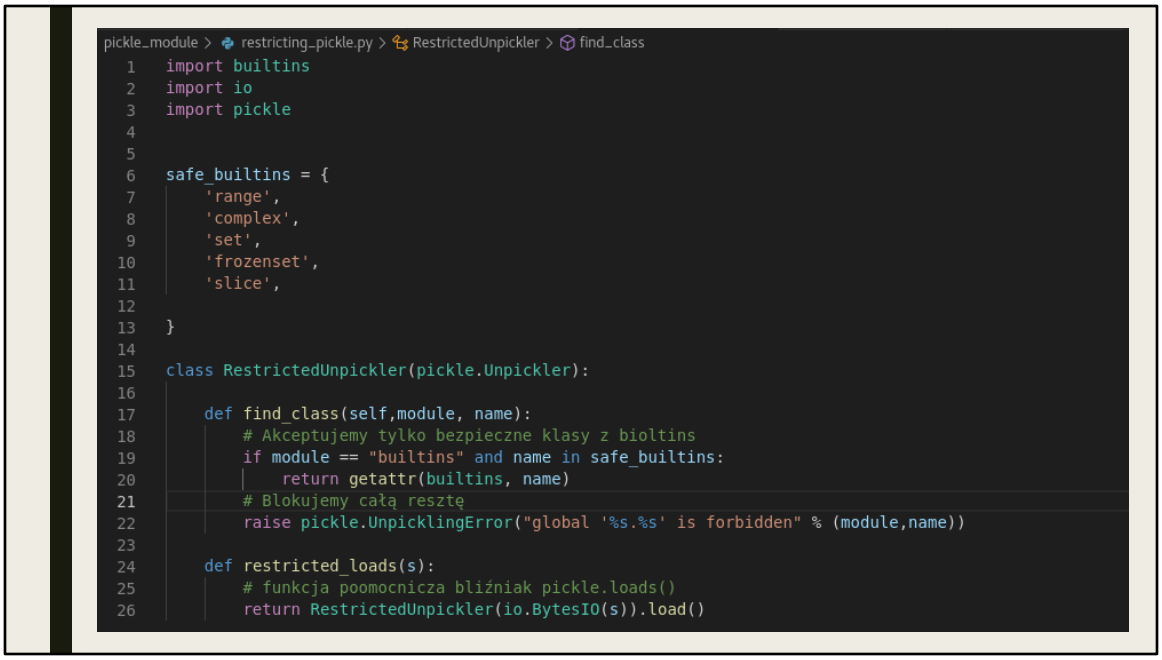
No nie do końca

```
15  # ,,Bezpieczniejsze przesyłanie
16  print(hmac.new(b'LOSOWY_KLUCZ',pickled).hexdigest())
```

```
2022-06-09 13:19:49.723693
1f83223c41800293b0b470302d477bbf
```

Wygląda znajomo nieprawdaż?

Jednym ze sposobów jest zabezpieczanie drogi przekazywania tych obiektów, niweluje nam to problem nieznanych źródeł. Hash upewnia nas ze nic się nie zmieniło w tej transmisji.



Przez ten przykład chcę pokazać, że możemy sami zabezpieczać sobie ten proces. Czy jest to dobra praktyka? Well i tak i nie. W branży CS pojawiło się takie zdanie, aby nie pisać swoich zabezpieczeń tylko korzystać z cudzych najlepiej opensoure'owych. Dlaczego? Bo nad nimi często czuwają tysiące programistów i szansa na to że oni wszyscy przeoczyli jakiś błąd/ podatność jest o wiele niższa, niż że my jesteśmy nieomylni i wszechwiedzący. Jakby nie chcę omawiać nazbyt tego przykładu, moim zdaniem warta uwagi jest tutaj idea dopuszczania danych wejściowych. Wszystko to z zamkniętej listy wpuszczamy, całej reszty nie. Wiem, że niektórzy na początku piszą swoje walidacje/ zabezpieczenia w ten sposób, że o zablokuje to to to i tamto i jestem bezpieczny. Nooo nie, bo co jeśli zapomnisz o czymś, albo ktoś doda customową funkcję której te warunki nie będą obejmowały?

PODSUMOWUJĄC

Pierwszym pytaniem, na które musimy sobie odpowiedzieć to czy w ogóle potrzebujemy tej deserializacji. W cyberbezpieczeństwie tak naprawdę chodzi o utrzmyanie odpowieniego balansu stosunku bezpieczeństwa to funkcjonalności. Jeśli odpowiemy sobie, że tak chcemy przesyłać obiekty należy przycupnąć i się zastanowić jak. W przypadu Pythona JSON ma dosyć ograniczoną funkcjonalność, sooo ktoś może nam mniej nabroić. A tak w ogóle co ja chce przesyłać, czy potrzebuje czegoś złożonego? Taką metoda date ma około 10 konstruktorów, czy nie ma pośród nich jakiegoś, który w zasadzie pozwolił nam na bezpieczne przerzucanie się czystymi typami „string”?

KONIEC

Dziękuję za uwagę

Przygotował
Paweł Jan Tłusty