NOTE: SCENARIO IS WHAT THE PUZZLE TAKER SEES.

SCENARIO:

You receive a file you've been expecting from your friend via email. After downloading the file, to make sure that the file is not tampered with, you calculate the SHA256 hash of the file and compare it with the already known valid hash (which you had received in advance over a secure channel). The following is a simple implementation of an auxiliary method that accepts a file path and an expected hash string, calculates the MD5 hash of the file, compares it to the expected hash, and returns `True` if the two hash values are equal and `False` otherwise. Consider the snippet of code below and answer the following questions, assuming that the code has all required permissions to execute.

```
01   # import whatever is needed
02   BLOCKSIZE = 65536
03
04   def is_valid_file(file_path, expected_hash):
05     hasher = hashlib.sha256()
06     with open(file_path, 'rb') as file:
07       buf = file.read(BLOCKSIZE)
08       while len(buf) > 0:
09         hasher.update(buf)
10         buf = file.read(BLOCKSIZE)
11     return hmac.compare_digest(expected_hash, hasher.hexdigest())
```

Questions:
1. What will the function `is_valid_file` do when it is called?

2. Which one of the following is correct if it is possible for you to precisely measure the execution time of the `is_valid_file` function, called with same-sized files and same expected hash value? (Note that there is no external factor affecting the execution time.)
a. The execution time is a constant value.
b. The execution time increases by the number of call to the function.
c. The execution time decreases by the number of call to the function.
d. None of the above

[Other statistical questions will be imported here while creating survey.]

NOTE: ANSWER IS TO BE SHOWN TO THE PUZZLE TAKER AT THE END OF THE SESSION.

ANSWER:

1. The code will calculate the hash of the given file and then compare it to the given expected hash value using string equality operator.

2. a
The `compare_digest` function performs in constant time to avoid timing attacks. Although Python documentation says: if the two arguments of the `compare_digest` function are of different lengths, or if an error occurs, a timing attack could theoretically reveal information about the types and lengths of a and b–but not their values, it does not apply to the above code.

TAGS:
python, cryptography, hash, timing-attack, flawless

CATEGORIES:
Blindspot - NO
Type - Crypto
Number of distinct functions - 8
Number of total functions - 8
Blindspot function - `N/A`
Function call omitted - N/A
Blindspot type - N/A
Number of Parameters in the blindspot function - N/A
Cyclomatic complexity - 5

NAME:
hmac.compare_digest(a, b)

DESCRIPTION:
Return `a == b`. This function uses an approach designed to prevent timing analysis by avoiding content-based short circuiting behaviour, making it appropriate for cryptography. `a` and `b` must both be of the same type: either unicode or a bytes-like object.

BLINDSPOT:
#N/A

CORRECT USE EXAMPLE:
#N/A

MORE INFORMATION:
#N/A

REFERENCES:
1. https://docs.python.org/2/library/hmac.html
2. https://codahale.com/a-lesson-in-timing-attacks/
3. https://security.stackexchange.com/questions/83660