Challenge 3 ("aray")

Description

> And now for something completely different. I'm pretty sure you know how to write Yara rules, but can you reverse them?

Writeup

The third challenge, "aray", was a favourite of mine. Everything about this challenge was enjoyable — the idea, the problem, and maybe especially the name.

The provided archive contains a single file — `aray.yara`. Indeed, this is a valid YARA file containing a single rule, `aray`. The condition is a giant AND clause with all kinds of terms — constraints on the values of individual bytes, doublewords, or hashes (MD5, SHA-256, CRC32) of specific parts of the file.

To improve readability, I first searched and replaced every occurence of `and<space>` with `and<newline>`. The first line of the condition asserts that the size of the file is 85 bytes. The following line provides us with the hash of the whole file contents. The rest of the conditions operate on a specific chunk of the file.

It was apparent that a lot of the conditions were superfluous or irrelevant - for example, a lot of them put constraints on the filesize, even though we already knew that from the first line. Next, many lines were of the form `<some part of the file> % x < x`, which is a tautology.

After going through a couple of lines and trying to reconstruct the file contents by hand, I thought a Python script would be faster, less error-prone and, above all else, the most satisfying. So I saved the newline-separated list of conditions into a separate file and wrote the following solver.

```
import binasciiimport hashlibimport refrom sys import argvdef strings_of_length(strlen):    if
strlen == 0:        yield ''    else:        for s in strings_of_length(strlen - 1):
for c in [chr(i) for i in range(32, 127)]:                yield c + sdef reverse_crc32(h,
strlen):    for s in strings_of_length(strlen):        if binascii.crc32(s.encode()) &
0xFFFFFFFF == int(h, 16):            return s    return Nonedef reverse_md5(h, strlen):    for
s in strings_of_length(strlen):        if hashlib.md5(s.encode()).hexdigest() == h:
return s    return Nonedef reverse_sha256(h, strlen):    for s in strings_of_length(strlen):
if hashlib.sha256(s.encode()).hexdigest() == h:            return s    return Nonedef main():
with open(argv[1], 'r') as f:        lines = f.readlines()        buffer   = None    file_md5
= None    for line in lines:        if re.match(r'filesize == \d+ and', line):
filesize = int(line.split(' ')[2])            buffer = [None for _ in range(filesize)]
elif re.match(r'hash.md5\(0, filesize\) ==', line):            file_md5 = re.match(r'hash.md5\
(0, filesize\) == "([0-9a-f]+)"', line).group(1)                elif re.match(r'hash.
(md5|sha256)\(\d+, \d+\) ==', line):            eq = re.match(r'hash.(md5|sha256)\((\d+),
(\d+)\) == "([0-9a-f]+)" and', line)            which = eq.group(1)            idx, length =
int(eq.group(2)), int(eq.group(3))            digest = eq.group(4)            plaintext =
(reverse_md5(digest, length)                        if which == 'md5'
```

```
else reverse_sha256(digest, length))            assert buffer[idx:idx+length].count(None) ==
length            buffer[idx:idx+length] = plaintext            print(f'[{idx}..{idx+length}]
= {plaintext}')                elif re.match(r'hash.crc32\(\d+, \d+\) ==', line):
eq = re.match(r'hash.crc32\((\d+), (\d+)\) == 0x([0-9a-f]+)', line)            idx, length,
crc = int(eq.group(1)), int(eq.group(2)), eq.group(3)            plaintext =
reverse_crc32(crc, length)            assert buffer[idx:idx+length].count(None) == length
buffer[idx:idx+length] = plaintext            print(f'[{idx}..{idx+length}] = {plaintext}')
elif re.match(r'uint(8|32)\(\d+\) [+-^] \d+ == \d+', line):            eq =
re.match(r'uint(8|32)\((\d+)\) ([+-^]) (\d+) == (\d+)', line)            size =
int(eq.group(1)) // 8            idx = int(eq.group(2))            op = eq.group(3)
a = int(eq.group(4))            b = int(eq.group(5))            if op == '+':
result = (b - a).to_bytes(size, 'little').decode()            elif op == '-':
result = (b + a).to_bytes(size, 'little').decode()            elif op == '^':
result = (b ^ a).to_bytes(size, 'little').decode()            assert
buffer[idx:idx+size].count(None) == size            buffer[idx:idx+size] = result
print(f'[{idx}..{idx+size}] = {result}')    assert buffer.count(None) == 0
print(''.join(buffer))    print(f'Expected MD5: {file_md5}')    print(f'Actual MD5:
{hashlib.md5("".join(buffer).encode()).hexdigest()}')main()
```

(It's not my proudest creation, but it gets the job done. Also, at least some of the ugliness can be blamed on my faithful servant, GitHub Copilot.)

At first, I assumed the `uint32`s were big endian, but after correcting my mistake, the MD5 hash finally matched and I got the flag:

```
...rule flareon { strings: $f = "1RuleADayK33p$Malw4r3Aw4y@flare-on.com" condition: $f
}Expected MD5: b7dc94ca98aa58dabb5404541c812db2Actual MD5:   b7dc94ca98aa58dabb5404541c812db2
```

Revision #2
★ Created 23 December 2024 21:03:14 by Annatar
Updated 23 December 2024 21:13:47 by Annatar