Mess - Operation M.E.M.

First of all, I used volatility to extract the mess.exe executable from the memory dump image. The executable is a tool used to encrypt a given input file.

By reverse engineering the executable, I found out that it creates a key by concatenating the result of the *GetUserNameExA* function with a password read from the standard input, and the result is XOR'ed with a 16-bytes buffer.

```
snprintf(key, 100, "%-30s%s", username, password);
i = 0;
while (i < 100) {
  key[i] ^= buffer[i % 16];
  i++;
}
```

The resulting key is used as a symmetric key for the AES algorithm to encrypt the input file. The result of the encryption is written in the output file.
The key is also written in the output file, but after it is encrypted using RSA with a key that we don't know.

The output file has the following format:

- header - 4 bytes: MESS
- the 16 bytes buffer
- RSA ciphertext length - 4 bytes
- AES ciphertext length - 4 bytes
- RSA ciphertext - the key encrypted using RSA
- AES ciphertext - the input file AES encrypted using the key generated as described above

As we can see, the output file contains the 16 bytes buffer, so to generate the key using the algorithm above we need the username and the password.
The *GetUserNameExA* function returns the name using the following format:

```
PC-Name\Username
```

After inspecting the memory dump, I found the following string "Agent191-PC\Agent191", which is exactly the string returned by the function.

So now the only thing I didn't know to decrypt the AES ciphertext is the password. A key aspect is that the username is padded in the key to 30 characters. As an AES key can have at most 32 bytes, it means that I only needed the first 2 bytes of the password, so I bruteforce them.

To solve the task, I created the python script called *mess.py*.