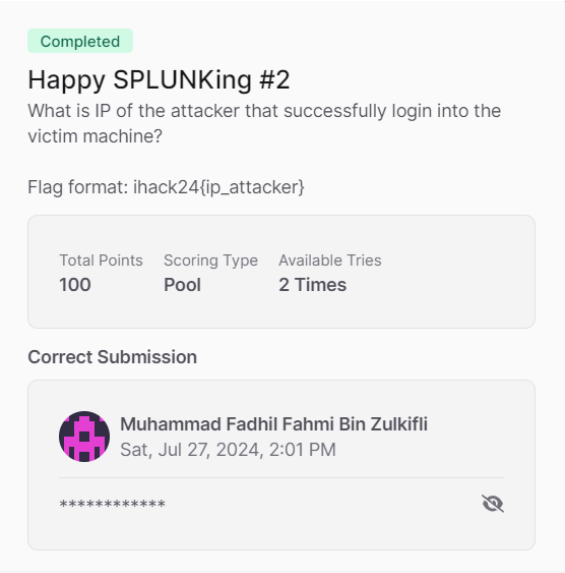
QUALIFICATION ROUND IHACK 2024

N3WBEES WRITEUP

Table of Contents

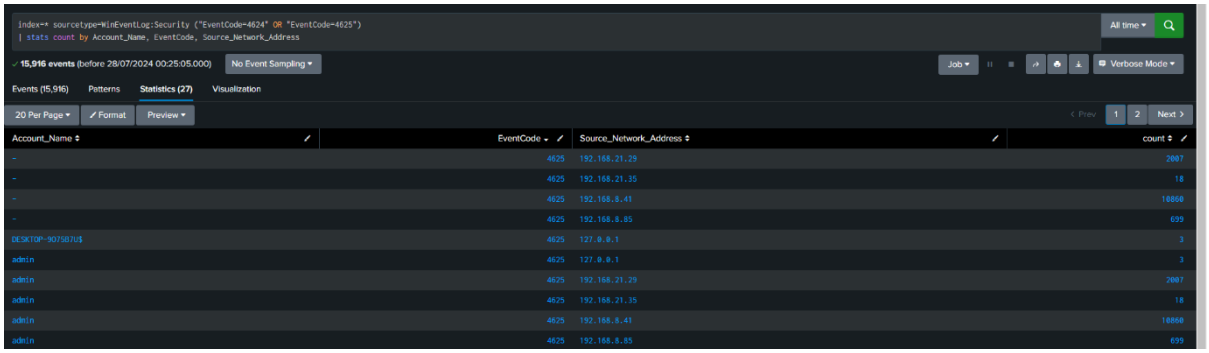
1.0 Digital Forensic & Incident Response (DFIR) - Happy SPLUNKing #2

🎽 Overview



✨ Walkthrough

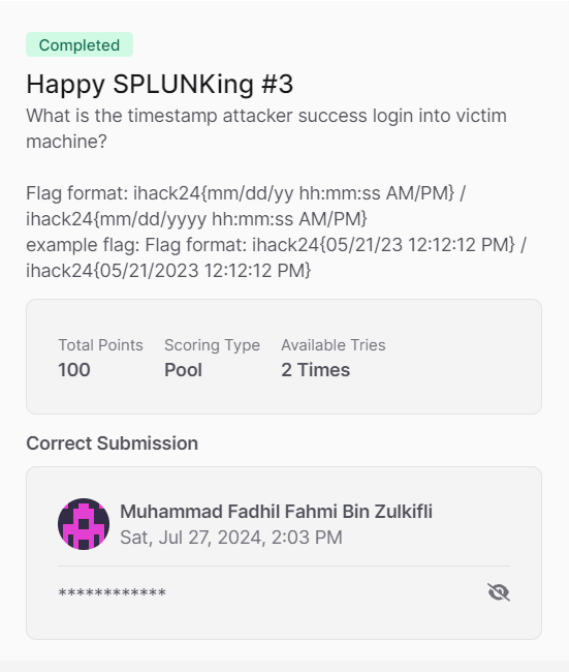
At first I don't know what to do since this is my first time doing splunk. So I just read at google and found out that you need to see the EventCode. Each code has a meaning. Then I'm just apply what I study in short term 💀. So, I see the EventCode=4625 is extremely high. Thats mean the attacker trying to bruteforce the account.



🏳️ **Flag > ihack24{192.168.8.41}**

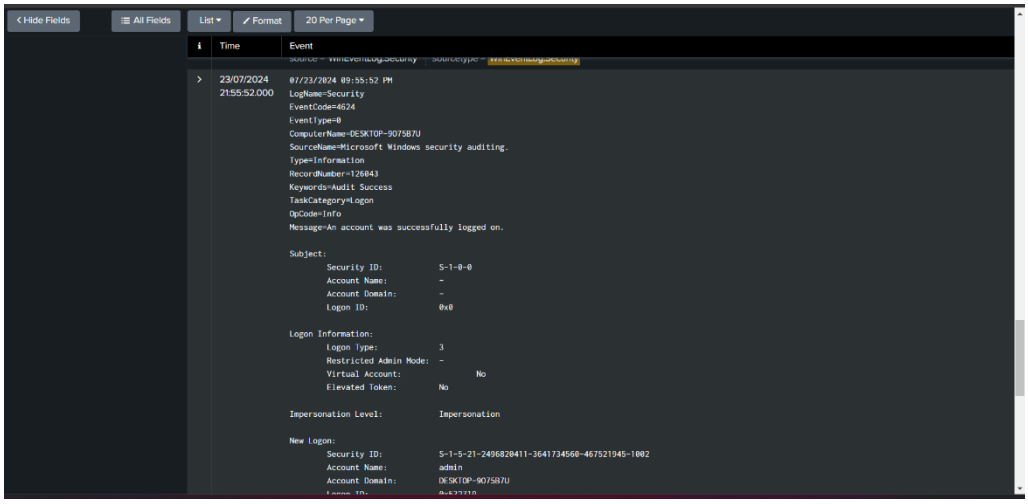
2.0 Digital Forensic & Incident Response (DFIR) - Happy SPLUNKing #3

🎏 Overview



✨ Walkthrough

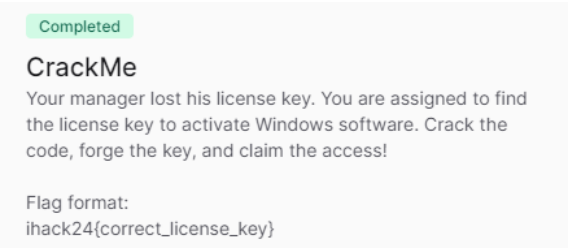
Since I know already the attacker Ip, I'm just straight filter the Ip and EventCode:4624. Why 4624? Because that's the EventCode for successful log in.



🏳️ Flag > ihack24{07/23/2024 09:55:52 PM}

3.0 Reverse Engineering – CrackMe

📚 Overview



✨ Walkthrough

Given an .EXE file written in .NET so let's decompile the file using dotPeek, which is *a free .NET decompiler and assembly browser by JetBrains*. After that, head over to the Assembly Explorer and then navigate to CrackMe (app host) > CrackMe > CrackMe > Form 1, who knows *something interesting* can be found in that section :D.

```
private bool ValidateLicenseKey(string key)
{
  string str = this.SecretKey("BRQFHF@WR_+6 ,N:$78", "secret");
  return key == str;
}

private string SecretKey(string hidden, string key)
{
  StringBuilder stringBuilder = new StringBuilder();
  for (int index = 0; index < hidden.Length; ++index)
    stringBuilder.Append((char) ((uint) hidden[index] ^ (uint) key[index % key.Length]));
  return stringBuilder.ToString();
}
```

There are two functions, `ValidateLicenseKey()` to validate whether our input is equal to the license key or not and `SecretKey()` to generate the key. In short, it will validate whether our input is equal to the generated key. The key was generated by *XORing* each character in obfuscated value with one character from the secret key using loop (*in this case, % operator was used to iterate over the 6 characters of 'secret' for XOR process*). That said, let's recreate the code using Python and run it to get the flag.

```
hidden = "BRQFHF@WR_+6 ,N:$78"
key = "secret"

for i in range(len(hidden)):
    print(chr(ord(hidden[i]) ^ ord(key[i % len(key)])), end='')
```

🏳️ **Flag > ihack24{1724-2321-NBSI-HACK}**

4.0 Reverse Engineering – Brute Force Frenzy

🎭 Overview

Completed

Brute Force Frenzy

Your manager's software has locked him out again. This time, the manager has lost the original key and urgently needs your help. Can you use your technical skill to brute-force the key and save the day once more?

Flag format:
ihack24{correct_license_key}

✨ Walkthrough

Given an .EXE file, so let's use Ghidra to analyse the file and after that, navigate to the check_license_key() function.

Decompile: check_license_key - (another_key.exe)

```
1
2  bool check_license_key(char *param_1)
3
4  {
5    int enc_key;
6    size_t keyLength;
7    bool check;
8    int i;
9    int key;
10
11   keyLength = strlen(param_1);
12   if (keyLength == 8) {
13     key = 0;
14     for (i = 0; i < 8; i = i + 1) {
15       enc_key = ((int)param_1[i] * (i + 1) + 0xd) % 0x61;
16       key = key + enc_key;
17       if (enc_key != *(int *)(&DAT_140010000 + (longlong)i * 4)) {
18         return false;
19       }
20     }
21     check = key == target_sum;
22   }
23   else {
24     check = false;
25   }
26   return check;
27 }
28
```

This function will take an input (*license key*) from the user, obfuscate it, and then compare it with the key that has been encoded (*which can be found in DAT_140010000*). For the obfuscation process,

i. **The char will be multiplied with (i + 1).**
ii. **After that, added it with 0xD.**
iii. **Lastly, take the remainder of *its* division with 0x61 (*using % operator*).**

The problem is, reversing the modulo process would be troublesome because there are quite number of possible values for the character of the original key. Well, nevermind, that isn't a problem anymore 😼.

***"With this treasure, I summon bruteforce techniques!"*. Since the length of the key is only 8 characters and the number of printable ASCII characters are not really that big, bruteforce is quite feasible in this case :D (*I mean, its also mentioned in the challenge title*).**

```
ENC_KEY = [0x5B, 0x3E, 0x42, 0x13, 0x3B, 0x33, 0x48, 0x29]
ASCII_START = 0x21
ASCII_END = 0x7e

key = ''
for i in range(len(ENC_KEY)):
    for char in range(ASCII_START, ASCII_END):
        guess = (char * (i+1) + 0xd) % 0x61
        if guess == ENC_KEY[i]:
            key += chr(char)
            break

print("Key:", key)
```

ENC_KEY – Retrieved from the binary itself.
ASCII_START, ASCII_END – Range of printable ASCII character.

Short pseudocode to briefly explain the process ...

Loop through each position
Loop through each printable ASCII character
Obfuscate the character, and then compare. If matched, store in key variable.

🏳 **Flag > ihack24{NI220G24}**

5.0 Malware Analysis – Just a normal EXE

🧩 Overview

Completed

Just a normal EXE

One of our clients reported the presence of a file named "normal.exe" in their TEMP folder. They mentioned that this executable file sometimes runs automatically.
Can you help them discover what this executable file does?

md5 hash of file "file.tar.gz" :
933e754be6c8366d3b4e020080dbad00

✨ Walkthrough

When the programme executed in Windows command prompt, it will give an output which is similar to the command net user. Since the .EXE file is written in .NET, decompile it using dotPeek and after that, navigate to normal (msil) > ModuleNameSpace > MainApp.

```
foreach (string strA in args)
{
  if (string.Compare(strA, "-wait", true) == 0)
    flag = true;
  else if (strA.StartsWith("-extract", StringComparison.InvariantCultureIgnoreCase))
  {
    string[] strArray = strA.Split(new string[1]
    {
      ":"
    }, 2, StringSplitOptions.RemoveEmptyEntries);
    if (strArray.Length != 2)
    {
      Console.WriteLine("If you specify the -extract option you need to add a file for extraction in this way\r\n   -extract:\"<filename>\"");
      return 1;
    }
    path = strArray[1].Trim('"');
  }
```

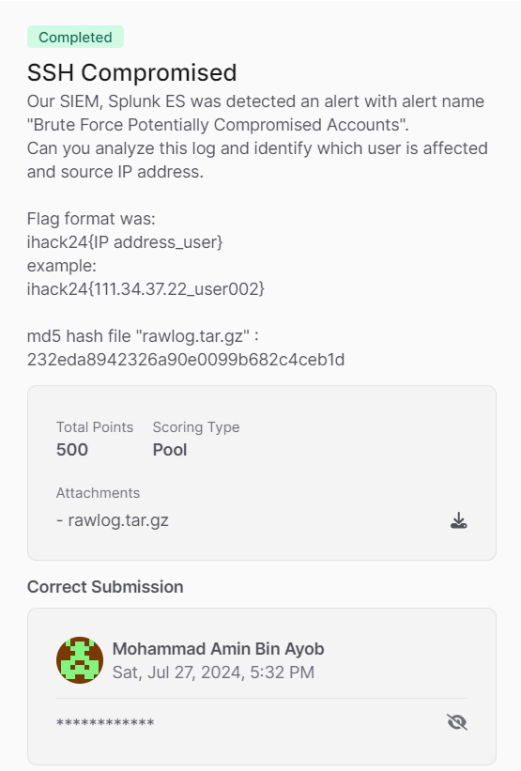
It can be seen that the programme also take arguments, allowing the users to extract the information into a file. So, let's create a file named output and then execute the programme using command normal -extract:"filename" in cmd.

Once the execution done, a *suspicious* array of char 👀 (*and also other bunch of stuffs*) appeared inside the file. That said, copy all the values and then convert it into ASCII characters. Turns out it is a link leading to the s3cr3t5.txt, which is the flag.

🏳 **Flag > ihack24{obFusCat!on_Alw4ys_w0rK}**

6.0 Incident Handling – SSH Compromised

🧩 Overview



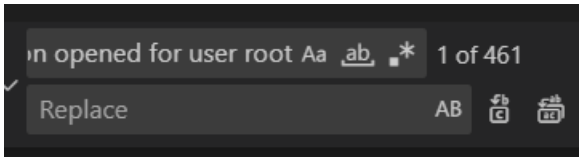
✨ Walkthrough

For this challenge, given the auth log file that contains much information about SSH login attempts, I used Visual Studio to open the log so I could see how many records matched my search. At first, I saw multiple failed password attempts for the user "root". I thought this was the answer, so I tried to submit it, but it was wrong.

Then, I analyzed the log again and noticed that there were many users in the log file. I searched each user one by one to see how many records existed for each user. For "root", there were many records. I searched for the next potential username I saw in the log, which was "sysadmin". This user had fewer records than "root". At this point, I still thought the answer was "root", but after scrolling through the log, I found a line indicating that a session was opened for the user "root", meaning the user "root" successfully logged in.

```
78   Jul 21 00:09:01 vmprod-uat-01 CRON[60148]: pam_unix(cron:session): session opened for user root(uid=0) by (uid=0)
79   Jul 21 00:09:01 vmprod-uat-01 CRON[60148]: pam_unix(cron:session): session closed for user root
```

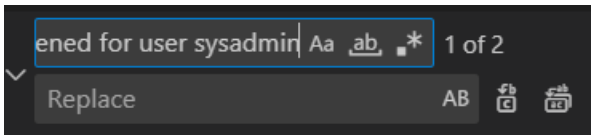
The records for the session opened for the user "root" were quite numerous. So, I knew user "root" had not been compromised. After searching for some time, I found what I was looking for.



There was another user that successfully logged in, which was "sysadmin". The line before this successful login looked suspicious, as user "sysadmin" had failed the password multiple times. This made me think that this might be a brute force attack by an attacker.

```
136265   Jul 27 05:02:24 vmprod-uat-01 sshd[153861]: Failed password for sysadmin from 149.102.244.68 port 24888 ssh2
136266   Jul 27 05:02:24 vmprod-uat-01 sshd[153863]: Failed password for sysadmin from 149.102.244.68 port 7153 ssh2
136267   Jul 27 05:02:24 vmprod-uat-01 sshd[153865]: Failed password for sysadmin from 149.102.244.68 port 53842 ssh2
136268   Jul 27 05:02:24 vmprod-uat-01 sshd[153869]: Failed password for sysadmin from 149.102.244.68 port 52336 ssh2
136269   Jul 27 05:02:24 vmprod-uat-01 sshd[153867]: Failed password for sysadmin from 149.102.244.68 port 57047 ssh2
136270   Jul 27 05:02:24 vmprod-uat-01 sshd[153845]: Failed password for sysadmin from 149.102.244.68 port 49249 ssh2
136271   Jul 27 05:02:25 vmprod-uat-01 sshd[153871]: Failed password for sysadmin from 149.102.244.68 port 24445 ssh2
136272   Jul 27 05:02:25 vmprod-uat-01 sshd[153873]: Failed password for sysadmin from 149.102.244.68 port 16503 ssh2
136273   Jul 27 05:02:25 vmprod-uat-01 sshd[153875]: Failed password for sysadmin from 149.102.244.68 port 46547 ssh2
136274   Jul 27 05:02:26 vmprod-uat-01 sshd[153863]: Accepted password for sysadmin from 149.102.244.68 port 7153 ssh2
136275   Jul 27 05:02:26 vmprod-uat-01 sshd[153863]: pam_unix(sshd:session): session opened for user sysadmin(uid=1000) by (uid=0)
136276   Jul 27 05:02:26 vmprod-uat-01 systemd-logind[712]: New session 735 of user sysadmin.
136277   Jul 27 05:02:26 vmprod-uat-01 systemd: pam_unix(systemd-user:session): session opened for user sysadmin(uid=1000) by (uid=0)
```

Also, by searching the line that I found, it showed only two records, indicating that the user had more failed logins than successful logins. I took the IP address that failed the password and the user "sysadmin", submitted the flag, and it worked. That's how I managed to solve this challenge.



Flag > ihack24{149.102.244.68_sysadmin}