

## Christmas Cheer Laser, part 8

### Answers to the Previous Question

9) What laser parameters do you recover from the XML document?

The last hint told us to look for a .xml file in the /etc directory.

```
PS /home/elf> gc /shall/see
Get the .xml children of /etc - an event log to be found. Group all .Id's and the last thing will be in the Properties of the lonely unique event Id.
PS /home/elf>
```

We've already done something like that, so it shouldn't be hard. It's nice that PowerShell will do recursive searches with a wild card in the middle, like /etc/\*.xml.

```
dir /etc/*.xml -Recurse -ErrorAction SilentlyContinue
```

```
PS /home/elf> dir /etc/*.xml -Recurse -ErrorAction SilentlyContinue

Directory: /etc/systemd/system/timers.target.wants

Mode                LastWriteTime         Length Name
----                -
--r---             11/18/19  7:53 PM      10006962 EventLog.xml

PS /home/elf>
```

Notice that the EventLog.xml file is over 10 MB. It will take a long time to scroll through this file looking for the correct id. In fact, if we look at the first few lines of the file, we see that each event is very long.

```
gc /etc/systemd/system/timers.target.wants/EventLog.xml | select -first 20
```

```
PS /home/elf> gc /etc/systemd/system/timers.target.wants/EventLog.xml | select -first 20
<Obj Version="1.1.0.1" xmlns="http://schemas.microsoft.com/powershell/2004/04">
  <Obj RefId="0">
    <TN RefId="0">
      <T>System.Diagnostics.Eventing.Reader.EventLogRecord</T>
      <T>System.Diagnostics.Eventing.Reader.EventRecord</T>
      <T>System.Object</T>
    </TN>
    <ToString>System.Diagnostics.Eventing.Reader.EventLogRecord</ToString>
    <Props>
      <I32 N="Id">3</I32>
      <By N="Version">5</By>
      <Nil N="Qualifiers" />
      <By N="Level">4</By>
      <I32 N="Task">3</I32>
      <I16 N="Opcode">0</I16>
      <I64 N="Keywords">-9223372036854775808</I64>
      <I64 N="RecordId">2194</I64>
      <S N="ProviderName">Microsoft-Windows-Sysmon</S>
      <G N="ProviderId">5770385f-c22a-43e0-bf4c-06f5698ffbd9</G>
      <S N="LogName">Microsoft-Windows-Sysmon/Operational</S>
    </Props>
  </Obj>
</Obj>
PS /home/elf>
```

It turns out that each event is about 210 lines long. A copy of one event is in the text file in the file LaserXML1event.txt. We are lucky though, as the Id we need is in the screenshot above. It is `<I32 N="Id">` The value of the `N="Id"` node in the screenshot is 3. (I32 stands for 32 bit integer.)

According to the hint, we need to group all the Id values. There will be one value that only occurs once, and it should contain the information we are seeking.

As always, there are several ways to solve this problem. We will do it two ways, as both are educational.

### PowerShell XML Data Type

There is a lot of XML data, so most languages have methods to handle it. We can load the contents of EventLog.xml file into a variable (PowerShell variable names start with \$) and then change its data type to XML. Many programming languages call this type casting, or casting. `Get-Content` is like `cat` in Linux and `type` in Windows, so it has the aliases `gc`, `type`, and `cat`. (This terminal doesn't have `cat`.)

```
[xml]$xdoc = gc /etc/systemd/system/timers.target.wants/EventLog.xml
```

If you pipe `$xdoc` into `Get-Member`, you will see that it is a `System.Xml.XmlDocument` and it has many methods for dealing with XML data.

Let's explore. The variable itself will tell us that it is a collection of objects called `Objs`.

```
PS /home/elf> $xdoc
Objs
----
Objs
PS /home/elf> █
```

```
PS /home/elf> $xdoc.Objs
Version xmlns                               Obj
-----
1.1.0.1 http://schemas.microsoft.com/powershell/2004/04 {Obj, Obj, Obj, Obj...}
```

Ok. The objects inside of the `Objs` collection are called `Obj`.

We are dealing with a 10MB file, we should limit our output.

```
PS /home/elf> $xdoc.Objs.Obj | select -First 2
RefId      : 0
TN         : TN
ToString   : static string XmlNode(psobject instance)
Props      : Props
MS         : MS

RefId      : 24
TNRef      : TNRef
ToString   : static string XmlNode(psobject instance)
Props      : Props
MS         : MS
```

Ugh. We can continue for another three pages, drilling down into the XML file, finding what we want and extracting it. (If you want to see what it looks like to drill down layer after layer, see “Christmas Cheer Laser part 8.hard way.pdf”.) However, consider two things:

1. This is a Windows Event Log file
2. PowerShell imports and exports data via XML all the time.

It turns out that there is a PowerShell cmdlet that understands Windows Event Logs in XML, and it is called `Import-Clixml`. First, use `Import-Clixml` instead of `Get-Content` to put the XML data into a variable.

```
$x = Import-Clixml /etc/systemd/system/timers.target.wants/EventLog.xml
```

Then look at the first event with `$x[0]`.

```
PS /home/elf> $x = Import-Clixml /etc/systemd/system/timers.target.wants/EventLog.xml
PS /home/elf> $x[0]

Message           : Network connection detected:
RuleName          :
UtcTime           : 2019-11-07 17:51:21.083
ProcessGuid       : {BA5C6BBB-4C7A-5DC4-0000-0010F4540100}
ProcessId         : 1068
Image             : C:\Windows\System32\svchost.exe
User              : NT AUTHORITY\LOCAL SERVICE
Protocol          : udp
Initiated         : true
SourceIsIpv6      : false
SourceIp          : 192.168.1.150
SourceHostname    : elfuresearch.localdomain
SourcePort        : 123
SourcePortName    : ntp
DestinationIsIpv6 : false
DestinationIp     : 13.86.101.172
DestinationHostname:
DestinationPort   : 123
DestinationPortName: ntp
Id                : 3
Version           : 5
Qualifiers        :
Level             : 4
Task              : 3
Opcode            : 0
Keywords          : -9223372036854775808
RecordId          : 2184
```

Much better. How about `$x[0].Id`, since `Id` is the node we are looking for.

```
PS /home/elf> $x[0].Id
3
PS /home/elf> █
```

Nice! Let's group all the `Ids`.

```
$x.Id | Group-Object
```

```
PS /home/elf> $x.Id | Group-Object
```

Count	Name	Group
1	1	{1}
39	2	{2, 2, 2, 2...}
179	3	{3, 3, 3, 3...}
2	4	{4, 4}
905	5	{5, 5, 5, 5...}
98	6	{6, 6, 6, 6...}

```
PS /home/elf>
```

The event with Id = 1 is the only one that occurs once. (“One is the Loneliest Number” is a pre-historic song, circa 1969.) See what’s in the event with Id = 1.

```
$x | Where-Object {$_.Id -eq 1}
```

```
PS /home/elf> $x | Where-Object {$_.Id -eq 1}
```

```
Message          : Process Create:
                  RuleName:
                  UtcTime: 2019-11-07 17:59:56.525
                  ProcessGuid: {BA5C6BBB-5B9C-5DC4-0000-00107660A900}
                  ProcessId: 3664
                  Image: C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
                  FileVersion: 10.0.14393.206 (rs1_release.160915-0644)
                  Description: Windows PowerShell
                  Product: Microsoft® Windows® Operating System
                  Company: Microsoft Corporation
                  OriginalFileName: PowerShell.EXE
                  CommandLine:
                  C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe -c
                  "`$correct_gases_postbody = @{'n      O=6`n      H=7`n      He=3`n      N=4`n
                  Ne=22`n      Ar=11`n      Xe=10`n      F=20`n      Kr=8`n      Rn=9`n}'`n"
                  CurrentDirectory: C:\
                  User: ELFURESEARCH\allservices
                  LogonGuid: {BA5C6BBB-5B9C-5DC4-0000-0020F55CA900}
                  LogonId: 0xA95CF5
                  TerminalSessionId: 0
                  IntegrityLevel: High
                  Hashes: MD5=097CE5761C89434367598B34FE32893B
```

Done!

```
$correct_gases_postbody = @{'n      O=6`n      H=7`n      He=3`n      N=4`n
Ne=22`n      Ar=11`n      Xe=10`n      F=20`n      Kr=8`n      Rn=9`n}'`n
```

Note: the backtick ( ` ) is the PowerShell escape character, so `n is the same as \n in Linux, or newline. The challenge designers probably used the format above to keep us from copying and pasting it later, as it doesn’t work with Invoke-WebRequest the way it is written. It is correctly written on one line as

```
$correct_gases_postbody =
@{O=6;H=7;He=3;N=4;Ne=22;Ar=11;Xe=10;F=20;Kr=8;Rn=9}
```

## Regular Expressions and a Short Script

It is worthwhile showing this method because it provides exposure to Regular Expressions and loops. Refer back to the beginning of an event.

```
PS /home/elf> gc /etc/systemd/system/timers.target.wants/EventLog.xml | select -first 40
<Obj Version="1.1.0.1" xmlns="http://schemas.microsoft.com/powershell/2004/04">
  <Obj RefId="0">
    <TN RefId="0">
      <T>System.Diagnostics.Eventing.Reader.EventLogRecord</T>
      <T>System.Diagnostics.Eventing.Reader.EventRecord</T>
      <T>System.Object</T>
    </TN>
  </TN>
  <ToString>System.Diagnostics.Eventing.Reader.EventLogRecord</ToString>
  <Props>
    <I32 N="Id">3</I32>
    <By N="Version">5</By>
    <Nil N="Qualifiers" />
    <By N="Level">4</By>
    <I32 N="Task">3</I32>
    <I16 N="Opcode">0</I16>
    <I64 N="Keywords">-9223372036854775808</I64>
    <I64 N="RecordId">2194</I64>
    <S N="ProviderName">Microsoft-Windows-Sysmon</S>
    <G N="ProviderId">5770385f-c22a-43e0-bf4c-06f5698ffbd9</G>
    <S N="LogName">Microsoft-Windows-Sysmon/Operational</S>
    <I32 N="ProcessId">1960</I32>
    <I32 N="ThreadId">6648</I32>
    <S N="MachineName">elfuresearch</S>
    <Obj N="UserId" RefId="1">
      <TN RefId="1">
        <T>System.Security.Principal.SecurityIdentifier</T>
        <T>System.Security.Principal.IdentityReference</T>
        <T>System.Object</T>
      </TN>
      <ToString>S-1-5-18</ToString>
      <Props>
        <I32 N="BinaryLength">12</I32>
        <Nil N="AccountDomainSid" />
        <S N="Value">S-1-5-18</S>
      </Props>
    </Obj>
    <DT N="TimeCreated">2019-11-07T09:51:22.6559745-08:00</DT>
    <Nil N="ActivityId" />
    <Nil N="RelatedActivityId" />
    <S N="ContainerLog">microsoft-windows-sysmon/operational</S>
  </Props>
</Obj>
PS /home/elf>
```

The `.Id` the hint refers to is in the line `<I32 N="Id">3</I32>`. We want the `'3'` that's between the tags. Search for that with a regular expression (regex) like this.

```
$regex = '<I32 N=.Id.>(\d)</I32>'
```

Here I've replaced the quotes with the single character wild card (`.`) because I didn't want to bother to escape the quotes. (Lazy, I know.)

The number 3 will change from event to event so it is replaced by `\d`. Note that this will only grab one digit. In our case the `Id` is only one digit long so we can get away with it. If we needed to grab multiple digits, we could use `\d+`. In a regex, the `'+'` means "one or more of the previous character."

In a regex, parenthesis may be used for grouping or to capture output. In this case we want to capture the number that is in between the tags. The captured digits are returned in the variable \$Matches. \$Matches[0] contains the entire line that matched, and \$Matches[1] contains the match for the first set of parenthesis.

```
$ids = ''
$regex = '<I32 N=.Id.>(\d)</I32>'
Get-Content /etc/systemd/system/timers.target.wants/EventLog.xml |
    ForEach-Object {
        if($_ -match $regex){
            $ids += $Matches[1]
        }
    }
$ids.ToCharArray() | group
```

This code just pipes the content of EventLog.xml into a ForEach-Object loop, which captures the number we want and accumulates them in \$ids. To be able to group by the numbers we recover, we have to change the string \$ids to an array of characters. (Good thing that the Id's are only one-digit numbers, or else this wouldn't work.)

```
PS /home/elf> $ids = ''
PS /home/elf> $regex = '<I32 N=.Id.>(\d)</I32>'
PS /home/elf> Get-Content /etc/systemd/system/timers.target.wants/EventLog.xml | ForEach-Object {
>>     if($_ -match $regex){
>>         $ids += $Matches[1]
>>     }
>> }
PS /home/elf> $ids.ToCharArray() | group
```

Count	Name	Group
1	1	{1}
39	2	{2, 2, 2, 2...}
179	3	{3, 3, 3, 3...}
2	4	{4, 4}
905	5	{5, 5, 5, 5...}
98	6	{6, 6, 6, 6...}

The lonely unique Id referenced in the hint is the number 1 .

We can find the clue by selecting the .Id we want by looking at lines before and after the match to get the entire event (-Context 8,141). (I played with those numbers until I had the entire event, nothing magic about them.)

```
PS /home/elf> gc /etc/systemd/system/timers.target.wants/EventLog.xml | Select-String -i
'<I32 N=.Id.>1</I32>' -context 8,141
```

```
<Obj RefId="1800">
  <TN RefId="1800">
    <T>System.Diagnostics.Eventing.Reader.EventLogRecord</T>
    <T>System.Diagnostics.Eventing.Reader.EventRecord</T>
    <T>System.Object</T>
  </TN>
  <ToString>System.Diagnostics.Eventing.Reader.EventLogRecord</ToString>
  <Props>
>    <I32 N="Id">1</I32>
    <By N="Version">1</By>
    <Nil N="Qualifiers" />
    <By N="Level">4</By>
  </I32 N="Task">1</I32>
```



This hint trail has grown cold. To find our fourth parameter we need to back up where we expanded the archive and found the runme.elf file. Obviously we need to execute runme.elf. When solving this challenge, it pays to know what OS we are running on. PowerShell Core is available for many OSs. Find the OS, and then ask yourself, "What do I need to do to execute a script/executable I've written on this OS?"

### Questions

10) What OS is this terminal running?

11) How do you run a brand new script/executable in this OS? (If you need a hint look in /bin. The permissions are in Windows format instead of rwx format, but there is still useful information.)

12) What is the value of the last parameter?