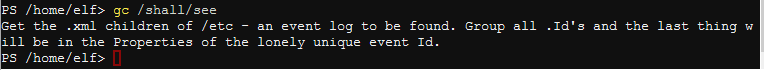
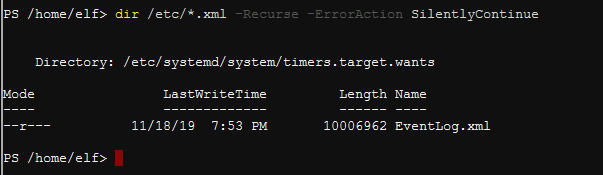
# Christmas Cheer Laser, part 8

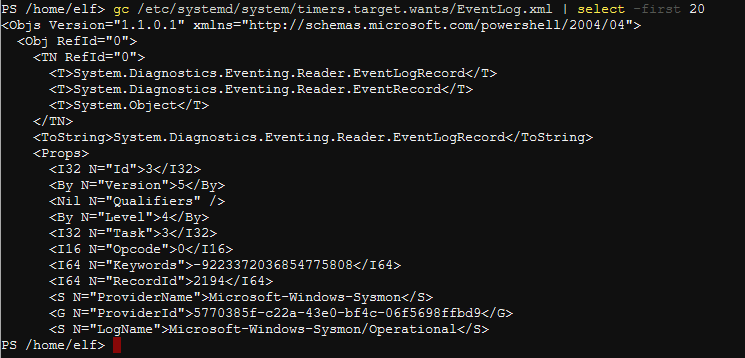
## Answers to the Previous Question

1. 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.  


We’ve already done something like that, so it shouldn’t be hard to find a .xml file. 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  


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  


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

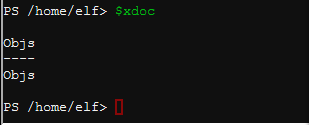
According to the hint, we need to group all the Id values. There will be one value that only occurs once, and it should contain the information we are seeking.

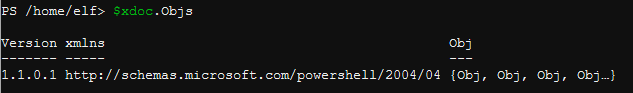
As always, there are several ways to solve this problem. We will do it two ways, as both are educational.

### PowerShell XML Data Type

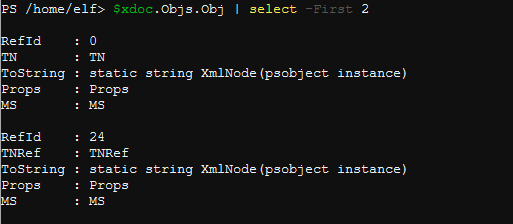
There is a lot of XML data, so most languages have methods to handle it. We can load the contents of EventLog.xml file into a variable (PowerShell variable names start with $) and then change its data type to XML. Many programming languages call this type casting, or casting. Get-Content is like cat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




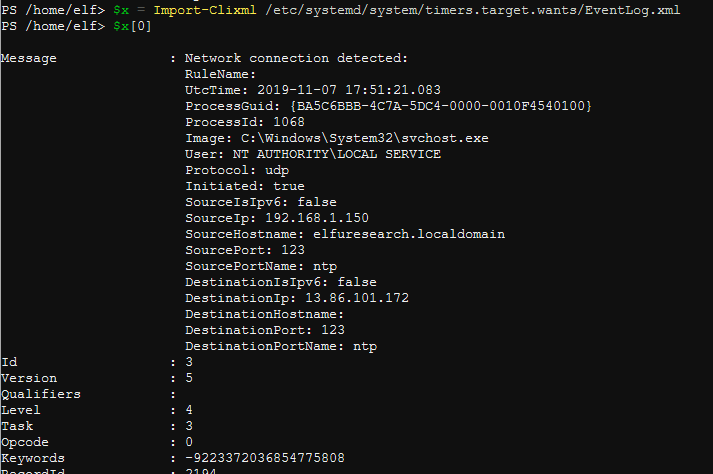
Ok. The objects inside of the Objs collection are called Obj.

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


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:

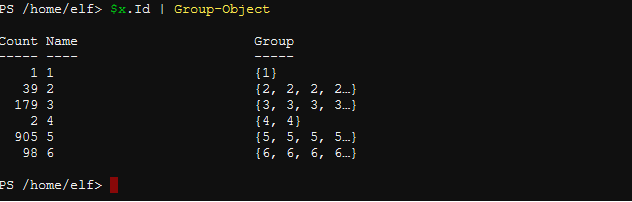
1. This is a Windows Event Log file
2. PowerShell imports and exports data via XML all the time.

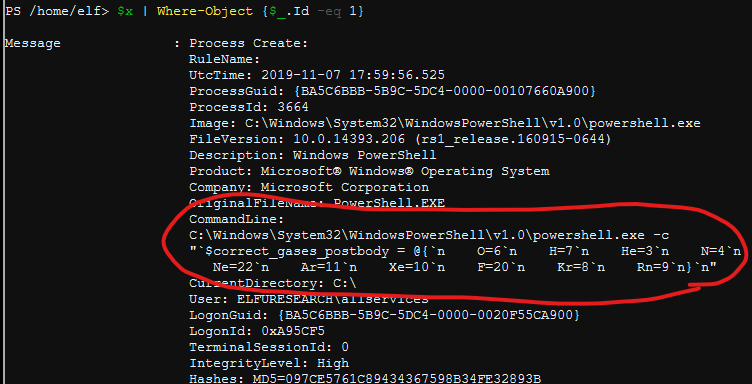
It turns out that there is a PowerShell cmdlet that understands Windows Event Logs in XML, and it is called Import-Clixml. First, use Import-Clixml instead of Get-Content to put the XML data into a variable.  
$x = Import-Clixml /etc/systemd/system/timers.target.wants/EventLog.xml

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


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


Nice! Let’s group all the Ids.

$x.Id | Group-Object  


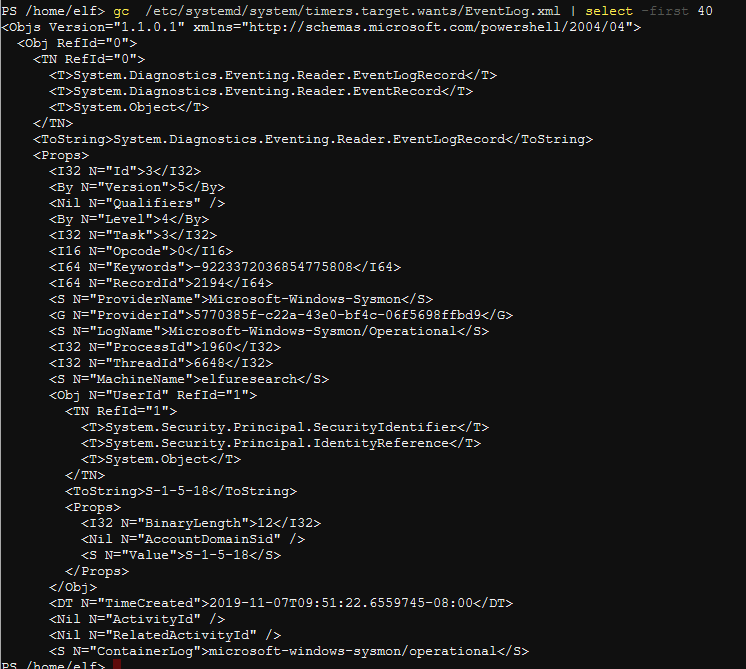
The event with Id = 1 is the only one that occurs once. (“One is the Loneliest Number” is a pre-historic song, circa 1969.) See what’s in the event with Id = 1.  
$x | Where-Object {$\_.Id -eq 1}  


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}

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.   


The .Id the hint refers to is in the line <I32 N=”Id”>3</I32>. We want the ‘3’ that’s between the tags. Seach 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. Ok, here’s the way you’re supposed to do it  
$regex = '<I32 N=\"Id\">(\d)</I32>' )

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, paraenthesis 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 {

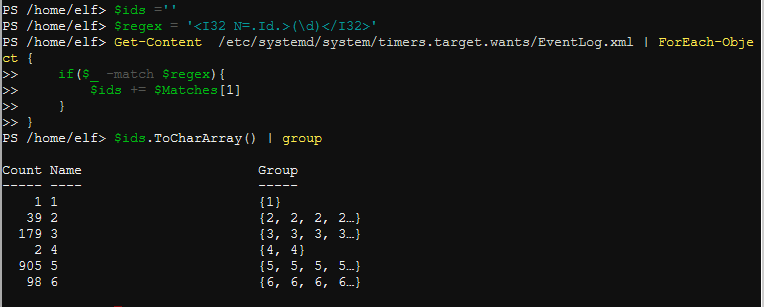
if($\_ -match $regex){

$ids += $Matches[1]

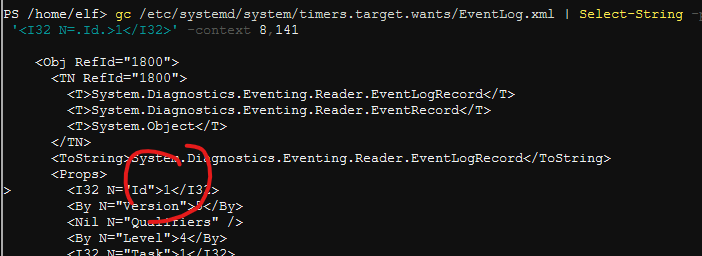
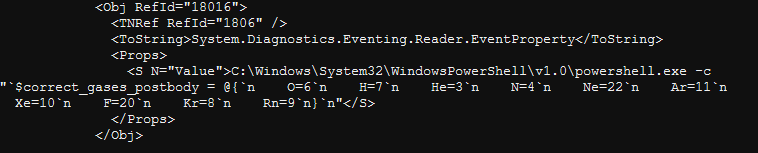
}

}

$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.)



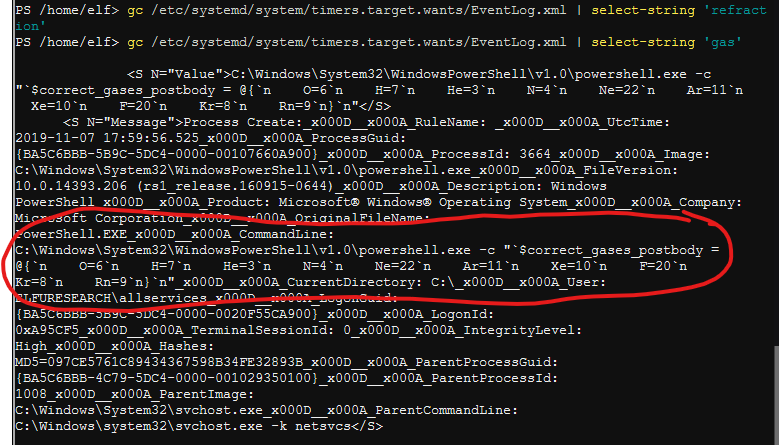
The lonely unique Id referenced in the hint is the number 1 .

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.)  
  
<snip>  
  
gc /etc/systemd/system/timers.target.wants/EventLog.xml | Select-String -pattern '<I32 N=.Id.>1</I32>' -context 8,141

So, we have our third parameter.  
$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}

### A Third, Lazy (smarter, if it works) Way

We know the parameter is either refraction or gas because we’ve found the other two. We’ll use Select-String just like we would use grep in Linux.  
gc /etc/systemd/system/timers.target.wants/EventLog.xml | select-string 'refraction'  
gc /etc/systemd/system/timers.target.wants/EventLog.xml | select-string 'gas'

Aha! The search for ‘gas’ did the trick!  


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

1. What OS is this terminal runnning?
2. 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.)
3. What is the value of the last parameter?