





LEE HOLMES

Precision Computing - Software Design and Development





SEARCHING FOR CONTENT IN BASE-64 STRINGS

📅Thu, Sep 21, 2017 5-minute read

You might have run into situations in the past where you’re looking for some specific text or binary sequence, but that content is encoded with Base-64. Base-64 is an incredibly common encoding format in malware, and all kinds of binary obfuscation tools alike.

The basic idea behind Base-64 is that it takes arbitrary binary data and encodes it into 64 (naturally) ASCII characters that can be transmitted safely over any normal transmission channel. Wikipedia goes into the full details here: <https://en.wikipedia.org/wiki/Base64>.

Some tooling supports decoding of Base-64 automatically, but that requires some pretty detailed knowledge of where the Base-64 starts and stops.

The Problem

Pretend you’re looking for the string, “Hello World” in a log file or SIEM system, but you know that it’s been Base-64 encoded. You might use PowerShell’s handy Base-64 classes to tell you what to search for:

```
Windows PowerShell

2 [C:\Users\leeholm]
>> $text = "Hello world"

3 [C:\Users\leeholm]
>> $binary = [System.Text.Encoding]::ASCII.GetBytes($text)

4 [C:\Users\leeholm]
>> [Convert]::ToBase64String($binary)
SGVsbG8gV29ybGQ=

5 [C:\Users\leeholm]
```

That seems useful. But what if “Hello World” is in the middle of a longer string? Can you still use ‘SGVobG8gV29fbGQ=’? It turns out, no. Adding a single character to the beginning changes almost everything:

```
Windows PowerShell

6 [C:\Users\leeholm]
>> $text = " Hello world"

7 [C:\Users\leeholm]
>> $binary = [System.Text.Encoding]::ASCII.GetBytes($text)

8 [C:\Users\leeholm]
>> [Convert]::ToBase64String($binary)
IEh1bGxvIFdvcmxk

9 [C:\Users\leeholm]
```

Now, we’ve got ‘IEh1bGxvIFdvcmxk’.

The main problem here is the way that Base-64 works. When we’re encoding characters, Base-64 takes 3 characters (24 bits) and re-interprets them as 4 segments of 6 bits each. It then encodes each of those 6 bits into the 64 characters that you know and love. Here’s a graphical example from Wikipedia:



LEE HOLMES

Precision Computing - Software Design and Development



sc

Home

Projects

Archive

Writing

About

Bit pattern	0	1	0	0	1	1	0	1	0	1	1	0	0	0	0	1	0	1	1	0	1	1	1	0
Index	19						22						5						46					
Base64-encoded	T						W						F						u					
encoded octets	84 (0x54)						87 (0x57)						70 (0x46)						117 (0x75)					

So when we add a character to the beginning, we shift the whole bit pattern to the right and change the encoding of everything that follows!

Another feature of Base-64 is padding. If your content isn't evenly divisible by 24 bits, Base-64 encoding will pad the remainder with null bytes. It will use the "=" character to denote how many extra padding blocks were used:

Text content	M																							
ASCII	77 (0x4d)						0 (0x00)						0 (0x00)						0 (0x00)					
Bit pattern	0	1	0	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Index	19						16						0						0					
Base64-encoded	T						Q						=						=					

When final padding is added, you can't just remove those "=" characters. If additional content is added to the end of your string (i.e.: "Hello World!"), that additional content will influence both the padding bytes, as well as the character before them.

Another major challenge is when the content is Unicode rather than ASCII. All of these points still apply – but the bit patterns change. Unicode usually represents characters as two bytes (16 bits). This is why the Base-64 encoding of Unicode content representing ASCII text has so many of the 'A' character: that is the Base-64 representation of a NULL byte.

```
Windows PowerShell
2 [C:\Users\leeholm]
>> $sb = New-Object System.Text.StringBuilder
3 [C:\Users\leeholm]
>> 1..100 | % { $null = $sb.Append( ([char] (Get-Random -Min 0 -Max 256) )) }
4 [C:\Users\leeholm]
>> $text = $sb.ToString()
5 [C:\Users\leeholm]
>> $bytes = [System.Text.Encoding]::Unicode.GetBytes($text)
6 [C:\Users\leeholm]
>> $base64 = [Convert]::ToBase64String($bytes)
7 [C:\Users\leeholm]
>> $base64.ToCharArray() | group | sort -Desc Count | select -First 10

Count Name                                     Group
-----
97 A                                           {A, A, A, A...}
13 g                                           {g, g, g, g...}
12 w                                           {w, w, w, w...}
11 Q                                           {Q, Q, Q, Q...}
10 D                                           {D, D, D, D...}
10 C                                           {C, C, C, C...}
8 B                                           {B, B, B, B...}
7 O                                           {O, O, O, O...}
6 O                                           {O, O, O, O...}
5 O                                           {O, O, O, O...}
```

The Solution

When you need to search for content that's been Base-64 encoded, then, the solution is to generate the text at all possible three-byte offsets, and remove the characters that might be influenced by the context: content either preceding what you are looking for, or the content that follows. Additionally, you should do this for both the ASCII as well as Unicode representations of the string.



LEE HOLMES

Precision Computing - Software Design and Development



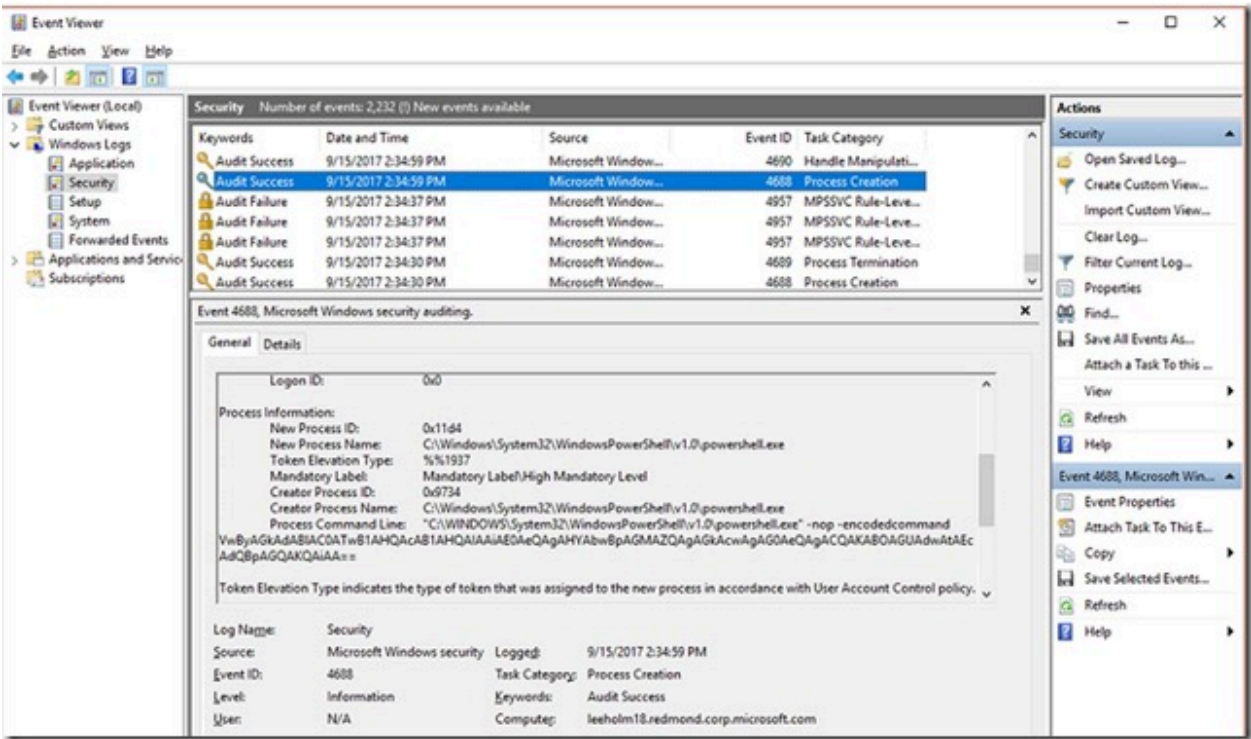
AI

Home Projects Archive Writing About

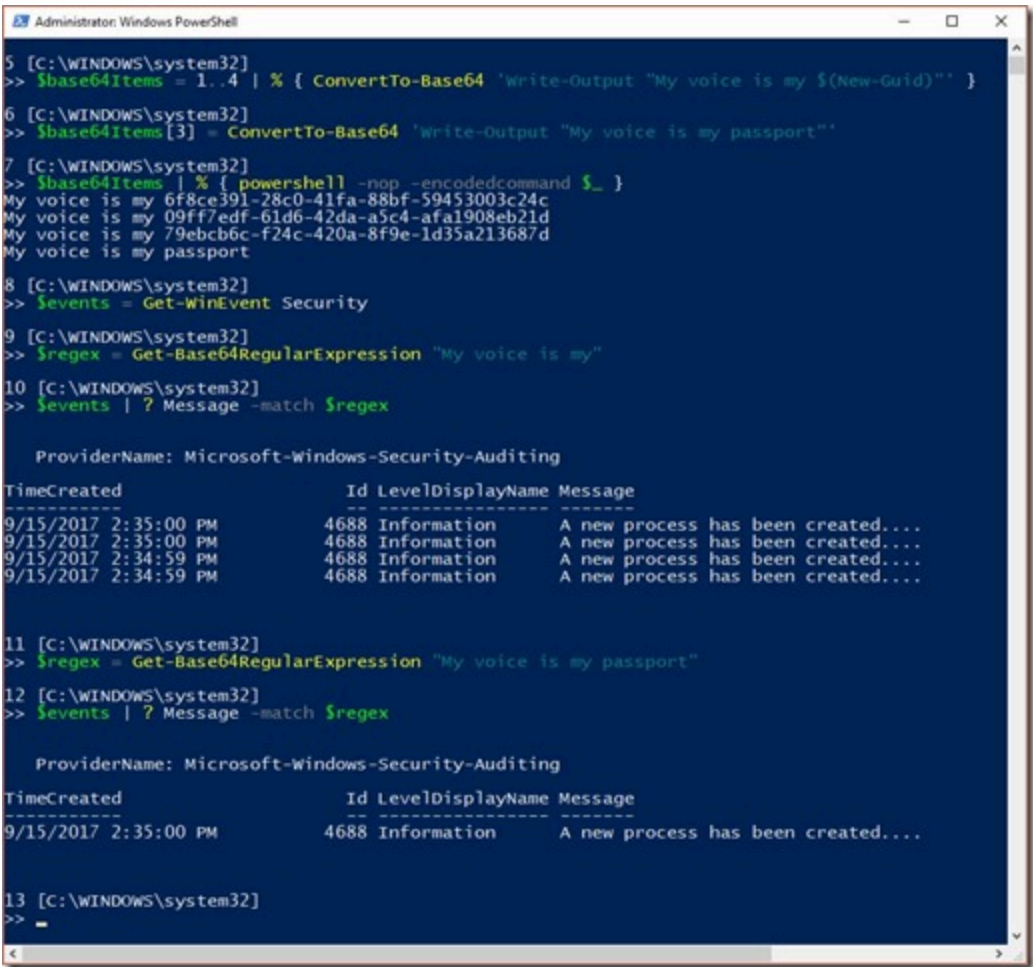


One example of Base-64 content is in PowerShell's EncodedCommand parameter. This shows up in Windows Event Logs if you have command-line logging enabled (and of course shows up directly if you have PowerShell logging enabled).

Here's an example of an event log like that:



Here's an example of launching a bunch of PowerShell instances with the - EncodedCommand parameter, as well as the magical **Get-Base64RegularExpression** command. That command will generate a regular expression that you can use to match against that content:



As you can see in this example, searching for the Base-64 content of "My voice is my" returned all four log entries, while the "My voice is my passport" search returned the single event log that contained the whole expression.

The Script

Get-Base64RegularExpression is a pretty simple script. You can use this in PowerShell, or any event log system that supports basic regular expression searches.

```
param(  
    ## The value that we would like to search for in Base64 encoded content  
    [Parameter(Mandatory, ValueFromPipeline)]  
    $Value,  
    [Parameter(Mandatory, ValueFromPipeline)]  
    $Regex
```



LEE HOLMES

Precision Computing - Software Design and Development



```
[Parameter()][Switch] $Unicode,

## True if we should emit the raw strings of each Base64 encoding
[Parameter()][Switch] $Raw
)

begin
{
    $base64sequences = @()
}

process
{
    ## Holds the various byte representations of what we're searching for
    $byteRepresentations = @()

    ## If we got a string, look for the representation of the string
    if($Value -is [String])
    {
        if($Unicode.IsPresent)
        {
            $byteRepresentations +=
                , [Byte[]] [System.Text.Encoding]::Unicode.GetBytes($Value)
        }
        else
        {
            $byteRepresentations +=
                , [Byte[]] [System.Text.Encoding]::Default.GetBytes($Value)
        }
    }

    ## If it was a byte array directly, look for the byte representations
    if($Value -is [byte[]])
    {
        $byteRepresentations += , $Value
    }

    ## Find the safe searchable sequences for each Base64 representation of in
    $base64sequences += foreach($bytes in $byteRepresentations)
    {
        ## Offset 0. Sits on a 3-byte boundary so we can trust the leading cha
        $offset0 = [Convert]::ToBase64String($bytes)

        ## Offset 1. Has one byte from preceeding content, so we need to throw
        ## first 2 leading characters
        $offset1 = [Convert]::ToBase64String( (New-Object 'Byte[]' 1) + $bytes

        ## Offset 2. Has two bytes from preceeding content, so we need to thro
        ## first 3 leading characters
        $offset2 = [Convert]::ToBase64String( (New-Object 'Byte[]' 2) + $bytes

        ## If there is any terminating padding, we must remove the characters
        ## ends up being the number of equals signs, plus one.
        $base64matches = $offset0,$offset1,$offset2 | % {
            if($_ -match '(=+)$')
            {
                $_.Substring(0, $_.Length - ($matches[0].Length + 1))
            }
            else
            {
                $_
            }
        }

        $base64matches | ? { $_ }
    }
}

end
{
    if($Raw.IsPresent)
    {
        $base64sequences | Sort-Object -Unique
    }
    else
    {

```



LEE HOLMES

Precision Computing - Software Design and Development



You can also find this on the [PowerShell Gallery](#):

Install-Script Get-Base64RegularExpression.ps1

[#PowerShell](#) [#security](#) [#technique](#)

comments

Click to Load Comments