New-Management Role
- Name ASCSD Recipient 1
- Parent "Mail Recipient Creation"

RFC4403

Get-Management Role Entry "ASCSD Recipient 1 \ ~~Remove-*~~ "

~~1 2 { $_.name -like "Remove-*" }~~

2 ~~Disable~~

| Remove-Management Role Entry -confirm:$false

New-Management Role
- Name ASCSD Recipient 2
- Parent "Mail Recipients"

Get-Management Role Entry "ASCSD Recipient 2 \ ~~Remove-*~~ "

(Disable)

~~1 2 { $_.name -like Remove Disable -*~~ + Remove }

| Remove-Management Role Entry -confirm:$false

Remove-Management Role Entry "ASCSD Recipient 1 \ New-LinkedUser "
New-RemoteMailbox"

---

1. New-ManagementRole -Name "ServiceDesk" -Parent "Mail Recipient Creation"

2. Get-ManagementRoleEntry "ServiceDesk*" | ?{$_.name -like "remove-*"}
| Remove-ManagementRoleEntry -Confirm:$false

3. New-ManagementScope -Name asd [RecipientRoot contoso.com/Employees
-RecipientRestrictionFilter {(RecipientType -eq "UserMailbox") -or
(RecipientType -eq "MailUser") -or (RecipientType -eq "MailContact")}

4. New-RoleGroup -Name Support -Roles "ServiceDesk"
-CustomRecipientWriteScope asd [Members ~~logons~~ logons who can use this role.]
→ Creates a Universal Group called Support

Mail Recipient Creation holds Remove-Mailbox
Mail Recipients holds Disable-Mailbox

New-ManagementRoleAssignment -SecurityGroup "GroupName" -Role "Unscoped Role Management"
New-ManagementRole -Name "x" -UnScopedTopLevel

Roles
Address-Lists
Distribution Groups
Mail Recipient Creation
Mail Recipients

New-ManagementRole ~~Assignment~~
-~~Role~~ Name -ModifyDistributionGroup
-Parent MyDistributionGroups

Remove-ManagementRoleEntry
-~~Modify~~Distribution Groups/New-Distribution Group
-Confirm:$false

Remove-Distribution Group

New-ManagementRoleAssignment
-Role ModifyDistribution Group
-Policy "Default Role Assignment Policy"

Get-ManagementRoleEntry "role*"
to list cmdlets on a role

Powershell -file <path>file.ps1
-noexit

Get-ExecutionPolicy
Set-Executionpolicy unrestricted

Variables start with $

technet.com → Exchange 2010 Cmdlets

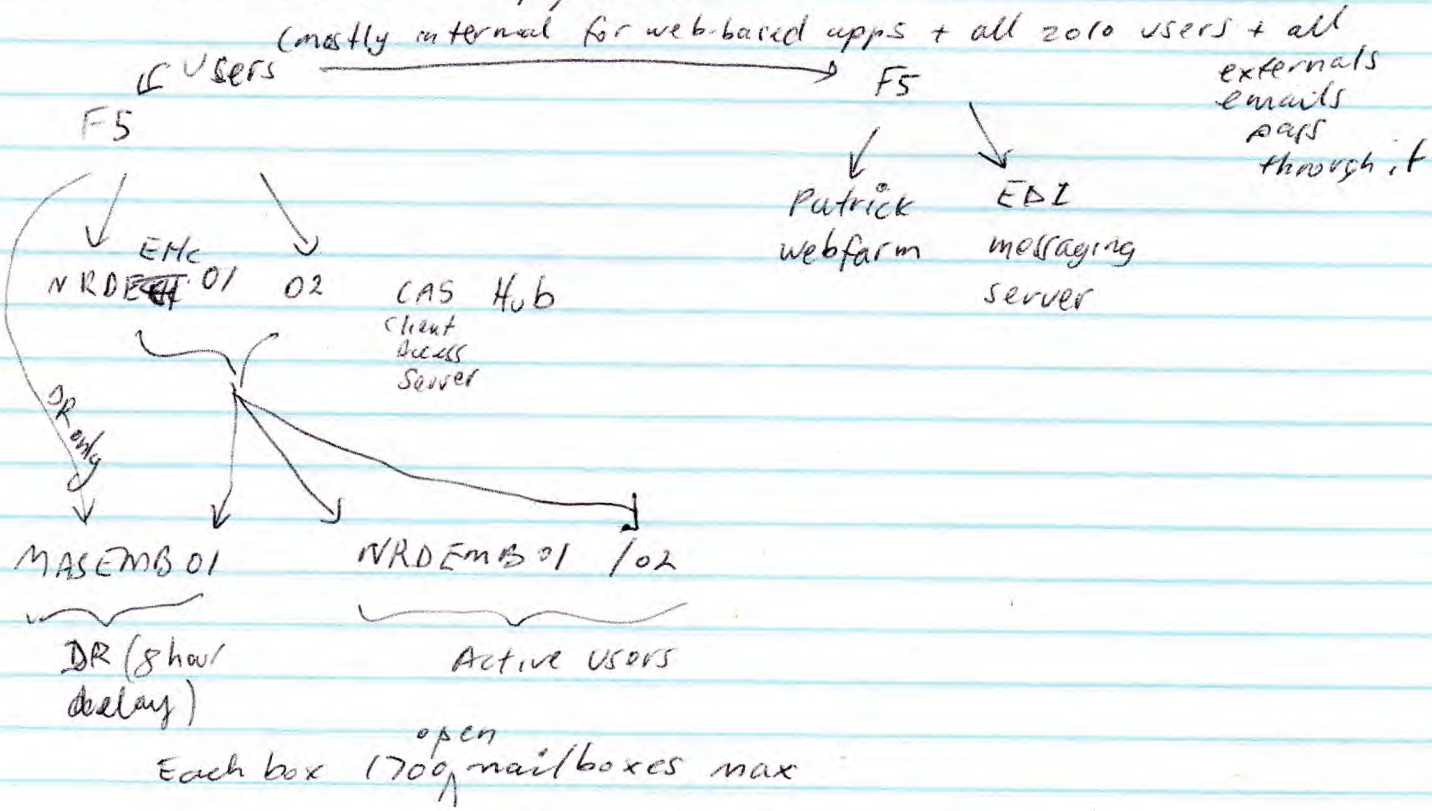
PowerGUI

Get-DisconnectedMailbox

Linked mailbox allows a user in one ~~domain~~ forest to have a mailbox in another
NOT RECOMMENDED

Equipment Mailbox + Room Mailbox are same thing - for resources and are always disabled.

Get-MailboxDatabaseCopyStatus



Ctrl-Alt-End = Ctrl Alt DEL
" - Break = Maximise

Pedro
Ali

ddls
technology | process | people

Richard Boxall 9299-3333
MS 10325 Automating Administration with Powershell 2.0
Ctrl-Alt-End Pa$$w0rd

Powershell can be put onto XP/2003 .NET Framework 3.5 SP1 is required

Get-Mailbox | Sort Size | Select -first 100 | Move-Mailbox server2

$host

verb-Noun
↓
Get - service

Get-psdrive Get-Command
Get-Help dir | get-member
Get-Alias Get-member

Get-Eventlog Security

New-PSDrive Test filesystem c:\test
Get-Alias -definition get-childitem

import-module activedirectory
cd AD:

format-table / wide / list

Get-process | select-object name, cpu, description

---

properties (file) size, attributes, owner
methods delete, copy, etc

$xyz is a variable
$myfile = dir
notepad $myfile

$myfile | foreach-object (notepad.exe $_)

Get-ComputerInventory | Get-Service

Get-Service -computerName NEWW302456

Get-xxx | Get-Member to get list of members in an object

Measure-Object -property VM -average -sum

tab = "`t"

bitwise 1 -band 2 AND
1 -bor 2 OR
1 -bxor 2 XOR

get-command | where-object {$_.definition -match "passthru"}

WMI Code Creator.exe

get-wmiobject win32-useraccount -computername "lon-dc1"

$cred get-credential "dom\id"

Input parameters

See page 6-25

New-aduser -Name moduleIIBtest1

Enable-Mailbox

13A

- Identity 'corp.asciano.ad/Divisional Groups/etc'
- Alias 'USERID' - Database 'Asciano Users AK'

Jennifer Levick : Users : Accounts : Coal : Divisional Groups, DC=pacnat

'pacnat.asciano.ad/Divisional Groups/Coal/Accounts/Users/Jennifer Levick'

Split off at comma LOV, 9 v1 = v2, 1(field),

then

Split off at each colon

Reverse the order

Windows Security Blog

PowerShell Byte Array And Hex Functions 11 fe 20

Posted by Jason Fossen
Filed under PowerShell

1 comme

PowerShell can manipulate and convert binary byte arrays, which is important for malware analysis, interacting with TCP ports, parsing binary log data, and a myriad other tasks. This article is a collection of PowerShell functions and notes for manipulating byte data. All the code here is in the public domain. If you find an error or have a suggested addition, please p a comment! The intention is to create a one-stop shop for PowerShell functions related to byte arrays and binary file handli It's best to assume that the functions here all require PowerShell 2.0 or later.

Background: Byte Array Issues

Handling byte arrays can be a challenge for many reasons:

- Bytes can be arranged in big-endian or little-endian format, and the endianness may need to be switched by one's code on the fly, e.g., Intel x86 processors use little-endian format internally, but TCP/IP is big-endian.
- A raw byte can be represented in one's code as a .NET object of type *System.Byte*, as a hexadecimal string, in some other format, and this format may need to be changed as the bytes are saved to a file, passed in as argument to a function, or sent to a TCP port over the network.
- Hex strings can be delimited in different ways in text files ("0xA5,0xB6" vs. "\xA5\xB6" vs. "A5-B6") or not delimited at all ("A5B6").
- Some cmdlets inject unwanted newlines into byte streams when piping.
- The redirection operators (> and >>) mangle byte streams as they attempt on-the-fly Unicode conversion.
- Bytes which represent text strings can encode those strings using ASCII, Unicode, UTF, UCS, etc.
- Newline delimiters in text files can be one or more different bytes depending on the application and operating system which created the file.
- Some .NET classes have unexpected working directories when their methods are invoked, so paths must be resolved explicitly first.
- StdIn and StdOut in PowerShell on Windows are not the same as in other languages on other platforms, whi can lead to undesired surprises.
- Manipulating very large arrays can lead to performance problems if the arrays are mishandled, e.g., not usin [Ref] where appropriate, constantly recopying to new arrays under the hood, recasting to different types unnecessarily, using the wrong .NET class or cmdlet, etc.

System.Byte

All variables in PowerShell are .NET objects, including 8-bit unsigned integer bytes. A byte object is an object of type *System.Byte* in the .NET class library, hence, it has properties and methods accessible to PowerShell (you can pipe them i get-member). To create a single *Byte* object or an array of *Bytes*:

```
[Byte] $x = 0x4D
[Byte[]] $y = 0x4D,0x5A,0x90,0x00,0x03
```

If you don't cast as a [Byte] or [Byte[]] array, then you'll accidentally get integers. When in doubt, check the type with get-member.

If you want your array to span multiple lines and include comments, that's OK with PowerShell, and remember that you car simply paste the code into your shell, you don't always have to save it to a script file first, hence, you can simply copy the following code and paste it into your shell, it'll work as-is:

```
[Byte[]] $payload =
0x00,0x00,0x00,0x90, # NetBIOS Session
0xff,0x53,0x4d,0x42, # Server Component: SMB
0x72, # SMB Command: Negotiate Protocol
0x00,0x00,0x00,0x00  # NT Status: STATUS_SUCCESS
```

Bitwise Operators (AND, OR, XOR)

PowerShell includes support for some bit-level operators that can work with *Byte* objects: binary AND (-band), binary OR (-bor) and binary XOR (-bxor). See the about_Comparison_Operators help file for details, and also see the format operator (for handling hex and other formats. PowerShell 2.0 and later supports bitwise operators on 64-bit integers too.

Bit Shifting ()

There are no built-in operators for bit shifting yet, but you can get functions for that here at Joel Bennett's site.

Get/Set/Add-Content Cmdlets

The following PowerShell cmdlets will take an "-encoding byte" argument, which you will want to use whenever manipulatir raw bytes with cmdlets in order to avoid unwanted on-the-fly Unicode conversions:

- Get-Content
- Set-Content
- Add-Content

The Out-File cmdlet has an -Encoding parameter too, but it will not take "byte" as an argument. The redirection operators (and ">>") actually use Out-File under the hood, which is why you should not use redirection operators (or Out-File) when handling raw bytes. The problem is that PowerShell often tries to be helpful by chomping and converting piped string data i lines of Unicode, but this is not helpful when you wish to handle a raw array of bytes *as is*.

To read the bytes of a file into an array:

```
[byte[]] $x = get-content -encoding byte -path .\file.exe
```

To read only the first 1000 bytes of a file into an array:

```
[byte[]] $x = get-content -encoding byte -path .\file.exe -totalcount 1000
```

However, the performance of get-content is *horrible* with large files. Unless you are reading less than 200KB, don't use get content, use the get-filebyte function below.

The same is not true, though, for set-content and add-content; their performance is probably adequate for even 5MB of dat but the write-filebyte function below is still much faster when working with more than 5MB.

Remember that you can't pipe just anything into set-content or add-content when using byte encoding, you must pipe objec of type *System.Byte* specifically. You will often need to convert your input data into a *System.Byte[]* array first (and there ar functions below for this).

To overwrite or create a file with raw bytes, avoiding any hidden string conversion, where $x is a *Byte[]* array:

```
set-content -value $x -encoding byte -path .\outfile.exe
```

To append to or create a file with raw bytes, avoiding any hidden string conversion, where $x is a *Byte[]* array:

```
add-content -value $x -encoding byte -path .\outfile.exe
```

Reading The Bytes Of A File Into A Byte Array

To quickly read the bytes of a file into a *Byte[]* array, even if the file is megabytes in size:

```
function Read-FileByte {
###################################################################
#.Synopsis
# Returns an array of System.Byte[] of the file contents.
#.Parameter Path
# Path to the file as a string or as System.IO.FileInfo object.
# FileInfo object can be piped into the function. Path as a
# string can be relative or absolute, but cannot be piped.
###################################################################
[CmdletBinding()] Param (
 [Parameter(Mandatory = $True, ValueFromPipelineByPropertyName = $True)]
 [Alias("FullName","FilePath")]
 $Path
)
[System.IO.File]::ReadAllBytes( $(resolve-path $Path) )
}
```

Writing Byte Array To File

To write a *Byte[]* array in memory to a new file or to overwrite an existing file (use the Add-Content cmdlet if you simply wa to append):

```
function Write-FileByte {
###################################################################
#.Synopsis
# Overwrites or creates a file with an array of raw bytes.
#.Parameter ByteArray
```

```
# System.Byte[] array of bytes to put into the file. If you
# pipe this array in, you must pipe the [Ref] to the array.
#.Parameter Path
# Path to the file as a string or as System.IO.FileInfo object.
# Path as a string can be relative, absolute, or a simple file
# name if the file is in the present working directory.
#.Example
# write-filebyte -bytearray $bytes -path outfile.bin
#.Example
# [Ref] $bytes | write-filebyte -path c:\temp\outfile.bin
########################################################################
[CmdletBinding()] Param (
 [Parameter(Mandatory = $True, ValueFromPipeline = $True)] [System.Byte[]] $ByteArray,
 [Parameter(Mandatory = $True)] $Path
)



if ($Path -is [System.IO.FileInfo])
 { $Path = $Path.FullName }
elseif ($Path -notlike "*\*") #Simple file name.
 { $Path = "$pwd" + "\" + "$Path" }
elseif ($Path -like ".\*") #pwd of script
 { $Path = $Path -replace "^\.",$pwd.Path }
elseif ($Path -like "..\*") #parent directory of pwd of script
 { $Path = $Path -replace "^\.\.",$(get-item $pwd).Parent.FullName }
else
 { throw "Cannot resolve path!" }
[System.IO.File]::WriteAllBytes($Path, $ByteArray)
}
```

Convert Hex String To Byte Array

You will often need to work with bytes represented in different formats and to convert between these formats. And sometim you'll have source code written in other programming languages that contain the equivalent of *Byte[]* arrays which you wan extract and convert into something more PowerShell-malleable, which is not hard if you paste that code into a here-string i your shell and then convert that string into a *Byte[]* array.

To extract hex data out of a string and convert that data into a *Byte[]* array while ignoring whitespaces, formatting or other n -hex characters in the string:

```
function Convert-HexStringToByteArray {
########################################################################
#.Synopsis
# Convert a string of hex data into a System.Byte[] array. An
# array is always returned, even if it contains only one byte.
#.Parameter String
# A string containing hex data in any of a variety of formats,
# including strings like the following, with or without extra
# tabs, spaces, quotes or other non-hex characters:
# 0x41,0x42,0x43,0x44
# \x41\x42\x43\x44
# 41-42-43-44
# 41424344
```

```
# The string can be piped into the function too.
#########################################################################
[CmdletBinding()]
Param ( [Parameter(Mandatory = $True, ValueFromPipeline = $True)] [String] $String )


#Clean out whitespaces and any other non-hex crud.
$String = $String.ToLower() -replace '[^a-f0-9\\\,x\-\:]',''


#Try to put into canonical colon-delimited format.
$String = $String -replace '0x|\\x|\-|,',':'


#Remove beginning and ending colons, and other detritus.
$String = $String -replace '^:+|:+$|x|\\',''


#Maybe there's nothing left over to convert...
if ($String.Length -eq 0) { ,@() ; return }


#Split string with or without colon delimiters.
if ($String.Length -eq 1)
{ ,@([System.Convert]::ToByte($String,16)) }
elseif (($String.Length % 2 -eq 0) -and ($String.IndexOf(":") -eq -1))
{ ,@($String -split '([a-f0-9]{2})' | foreach-object { if ($_) {[System.Convert]::ToByte($
elseif ($String.IndexOf(":") -ne -1)
{ ,@($String -split ':+' | foreach-object {[System.Convert]::ToByte($_,16)}) }
else
{ ,@() }
#The strange ",@(...)" syntax is needed to force the output into an
#array even if there is only one element in the output (or none).
}
```

Convert Byte Array To Hex

To convert a *Byte[]* array into a string containing hex characters in a variety of formats:

```
function Convert-ByteArrayToHexString {
#########################################################################
#.Synopsis
# Returns a hex representation of a System.Byte[] array as
# one or more strings. Hex format can be changed.
#.Parameter ByteArray
# System.Byte[] array of bytes to put into the file. If you
# pipe this array in, you must pipe the [Ref] to the array.
# Also accepts a single Byte object instead of Byte[].
#.Parameter Width
# Number of hex characters per line of output.
#.Parameter Delimiter
```

```
# How each pair of hex characters (each byte of input) will be
# delimited from the next pair in the output. The default
# looks like "0x41,0xFF,0xB9" but you could specify "\x" if
# you want the output like "\x41\xFF\xB9" instead. You do
# not have to worry about an extra comma, semicolon, colon
# or tab appearing before each line of output. The default
# value is ",0x".
#.Parameter Prepend
# An optional string you can prepend to each line of hex
# output, perhaps like '$x += ' to paste into another
# script, hence the single quotes.
#.Parameter AddQuotes
# An switch which will enclose each line in double-quotes.
#.Example
# [Byte[]] $x = 0x41,0x42,0x43,0x44
# Convert-ByteArrayToHexString $x
#
# 0x41,0x42,0x43,0x44
#.Example
# [Byte[]] $x = 0x41,0x42,0x43,0x44
# Convert-ByteArrayToHexString $x -width 2 -delimiter "\x" -addquotes
#
# "\x41\x42"
# "\x43\x44"
##########################################################################
[CmdletBinding()] Param (
 [Parameter(Mandatory = $True, ValueFromPipeline = $True)] [System.Byte[]] $ByteArray,
 [Parameter()] [Int] $Width = 10,
 [Parameter()] [String] $Delimiter = ",0x",
 [Parameter()] [String] $Prepend = "",
 [Parameter()] [Switch] $AddQuotes
)


if ($Width -lt 1) { $Width = 1 }
if ($ByteArray.Length -eq 0) { Return }
$FirstDelimiter = $Delimiter -Replace "^[\,\\:\t]",""
$From = 0
$To = $Width - 1
Do
{
 $String = [System.BitConverter]::ToString($ByteArray[$From..$To])
 $String = $FirstDelimiter + ($String -replace "\-",$Delimiter)
 if ($AddQuotes) { $String = '"' + $String + '"' }
 if ($Prepend -ne "") { $String = $Prepend + $String }
 $String
 $From += $Width
 $To += $Width
} While ($From -lt $ByteArray.Length)
}
```

Convert Byte Array To String (ASCII, Unicode or UTF)

To convert a *Byte[]* array which encodes ASCII, Unicode or UTF characters into a regular string:

```
function Convert-ByteArrayToString {
################################################################
#.Synopsis
# Returns the string representation of a System.Byte[] array.
# ASCII string is the default, but Unicode, UTF7, UTF8 and
# UTF32 are available too.
#.Parameter ByteArray
# System.Byte[] array of bytes to put into the file. If you
# pipe this array in, you must pipe the [Ref] to the array.
# Also accepts a single Byte object instead of Byte[].
#.Parameter Encoding
# Encoding of the string: ASCII, Unicode, UTF7, UTF8 or UTF32.
# ASCII is the default.
################################################################
[CmdletBinding()] Param (
 [Parameter(Mandatory = $True, ValueFromPipeline = $True)] [System.Byte[]] $ByteArray,
 [Parameter()] [String] $Encoding = "ASCII"
)


switch ( $Encoding.ToUpper() )
{
 "ASCII" { $EncodingType = "System.Text.ASCIIEncoding" }
 "UNICODE" { $EncodingType = "System.Text.UnicodeEncoding" }
 "UTF7" { $EncodingType = "System.Text.UTF7Encoding" }
 "UTF8" { $EncodingType = "System.Text.UTF8Encoding" }
 "UTF32" { $EncodingType = "System.Text.UTF32Encoding" }
 Default { $EncodingType = "System.Text.ASCIIEncoding" }
}
$Encode = new-object $EncodingType
$Encode.GetString($ByteArray)
}
```

Display A File's Hex Dump

There are many hex dumpers, but PowerShell dumpers can be copied into one's profile script, accept piped file objects, an support parameters like -Width, -Count, -NoOffset and -NoText to hopefully make it more flexible. You can get this as a sta alone script (Get-FileHex.ps1) from the SEC505 zip file if you wish, along with lots of other scripts.

```
function Get-FileHex {
###################################################################
#.Synopsis
# Display the hex dump of a file.
#.Parameter Path
# Path to file as a string or as a System.IO.FileInfo object;
# object can be piped into the function, string cannot.
#.Parameter Width
# Number of hex bytes shown per line (default = 16).
```

```
#.Parameter Count
# Number of bytes in the file to process (default = all).
#.Parameter NoOffset
# Switch to suppress offset line numbers in output.
#.Parameter NoText
# Switch to suppress ASCII translation of bytes in output.
###############################################################
[CmdletBinding()] Param (
 [Parameter(Mandatory = $True, ValueFromPipelineByPropertyName = $True)]
 [Alias("FullName","FilePath")] $Path,
 [Parameter()] [Int] $Width = 16,
 [Parameter()] [Int] $Count = -1,
 [Parameter()] [Switch] $NoOffset,
 [Parameter()] [Switch] $NoText
)


$linecounter = 0 # Offset from beginning of file in hex.
$placeholder = "." # What to print when byte is not a letter or digit.


get-content $path -encoding byte -readcount $width -totalcount $count |
foreach-object {
 $paddedhex = $asciitext = $null
 $bytes = $_ # Array of [Byte] objects that is $width items in length.


 foreach ($byte in $bytes) {
 $byteinhex = [String]::Format("{0:X}", $byte) # Convert byte to hex, e.g., "F".
 $paddedhex += $byteinhex.PadLeft(2,"0") + " " # Pad with zero to force 2-digit length, e.
 }


 # Total bytes in file unlikely to be evenly divisible by $width, so fix last line.
 # Hex output width is '$width * 3' because of the extra spaces added around hex character
 if ($paddedhex.length -lt $width * 3)
 { $paddedhex = $paddedhex.PadRight($width * 3," ") }


 foreach ($byte in $bytes) {
 if ( [Char]::IsLetterOrDigit($byte) -or
 [Char]::IsPunctuation($byte) -or
 [Char]::IsSymbol($byte) )
 { $asciitext += [Char] $byte }
 else
 { $asciitext += $placeholder }
 }


 $offsettext = [String]::Format("{0:X}", $linecounter) # Linecounter in hex too.
 $offsettext = $offsettext.PadLeft(8,"0") + "h:" # Pad linecounter with left zeros.
 $linecounter += $width # Increment linecounter, each line representing $width bytes.
```

```
 if (-not $NoOffset) { $paddedhex = "$offsettext $paddedhex" }
 if (-not $NoText) { $paddedhex = $paddedhex + $asciitext }
 $paddedhex
}
}
```

Toggle Big/Little Endian (With Sub-Widths)

Different platforms, languages, protocols and file formats may represent data in big-endian, little-endian, middle-endian or some other format (this is also called the "NUXI problem" or the "byte order problem"). If the relevant unit of data within an array is the single byte, then reversing the order of the array is sufficient to toggle endianness, but if the unit to be swappec two or more bytes (within a larger array) then a simple reversing might not be desired because then the ordering is change *within* that unit too. Ideally, a single function could be called multiple times, if necessary, with different unit lengths on a chopped up array of bytes to achieve the right endianness both within and across units in the original array. However, usuz you'll probably have just an array of bytes (1 unit = 1 byte) that simply needs to be reversed to toggle the endianness.

```
function Toggle-Endian {
#####################################################################
#.Synopsis
# Swaps the ordering of bytes in an array where each swappable
# unit can be one or more bytes, and, if more than one, the
# ordering of the bytes within that unit is NOT swapped. Can
# be used to toggle between little- and big-endian formats.
# Cannot be used to swap nibbles or bits within a single byte.
#.Parameter ByteArray
# System.Byte[] array of bytes to be rearranged. If you
# pipe this array in, you must pipe the [Ref] to the array, but
# a new array will be returned (originally array untouched).
#.Parameter SubWidthInBytes
# Defaults to 1 byte. Defines the number of bytes in each unit
# (or atomic element) which is swapped, but no swapping occurs
# within that unit. The number of bytes in the ByteArray must
# be evenly divisible by SubWidthInBytes.
#.Example
# $bytearray = toggle-endian $bytearray
#.Example
# [Ref] $bytearray | toggle-endian -SubWidthInBytes 2
#####################################################################
[CmdletBinding()] Param (
 [Parameter(Mandatory = $True, ValueFromPipeline = $True)] [System.Byte[]] $ByteArray,
 [Parameter()] [Int] $SubWidthInBytes = 1
)


if ($ByteArray.count -eq 1 -or $ByteArray.count -eq 0) { $ByteArray ; return }


if ($SubWidthInBytes -eq 1) { [System.Array]::Reverse($ByteArray); $ByteArray ; return }
```

```
if ($ByteArray.count % $SubWidthInBytes -ne 0)
{ throw "ByteArray size must be an even multiple of SubWidthInBytes!" ; return }


$newarray = new-object System.Byte[] $ByteArray.count


# $i tracks ByteArray from head, $j tracks NewArray from end.
for ($($i = 0; $j = $newarray.count - 1) ;
 $i -lt $ByteArray.count ;
 $($i += $SubWidthInBytes; $j -= $SubWidthInBytes))
{
 for ($k = 0 ; $k -lt $SubWidthInBytes ; $k++)
 { $newarray[$j - ($SubWidthInBytes - 1) + $k] = $ByteArray[$i + $k] }
}
$newarray
}
```

Inject Byte Array Into Listening TCP or UDP Port

As long as we're on the subject of manipulating bytes, sending bytes to a listening TCP or UDP port is easy (but processin responses requires more effort -- Lee Holmes has a nice script for it). Since it seems every miscellaneous language has be used to demonstrate how to run a particular DoS attack against SMBv2 on some Windows versions, here it is in PowerShe too for the bandwagon (there's nothing new here, the attack is well-known, it's just a demo for the function):

```
function PushToTcpPort
{
 param ([Byte[]] $bytearray, [String] $ipaddress, [Int32] $port)
 $tcpclient = new-object System.Net.Sockets.TcpClient($ipaddress, $port) -ErrorAction "Sil
 trap { "Failed to connect to $ipaddress`:$port" ; return }
 $networkstream = $tcpclient.getstream()
 #write(payload,starting offset,number of bytes to send)
 $networkstream.write($bytearray,0,$bytearray.length)
 $networkstream.close(1) #Wait 1 second before closing TCP session.
 $tcpclient.close()
}


[System.Byte[]] $payload =
0x00,0x00,0x00,0x90, # NetBIOS Session (these are fields as shown in Wireshark)
0xff,0x53,0x4d,0x42, # Server Component: SMB
0x72, # SMB Command: Negotiate Protocol
0x00,0x00,0x00,0x00, # NT Status: STATUS_SUCCESS
0x18, # Flags: Operation 0x18
0x53,0xc8, # Flags2: Sub 0xc853
0x00,0x26, # Process ID High (normal value should be 0x00,0x00)
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, # Signature
0x00,0x00, # Reserved
0xff,0xff, # Tree ID
0xff,0xfe, # Process ID
0x00,0x00, # User ID
```

```
0x00,0x00, # Multiplex ID
0x00, # Negotiate Protocol Request: Word Count (WCT)
0x6d,0x00, # Byte Count (BCC)
0x02,0x50,0x43,0x20,0x4e,0x45,0x54,0x57,0x4f,0x52,0x4b,0x20,0x50,0x52,0x4f,0x47,0x52,0x41,
0x02,0x4c,0x41,0x4e,0x4d,0x41,0x4e,0x31,0x2e,0x30,0x00, # Requested Dialects: LANMAN1.0
0x02,0x57,0x69,0x6e,0x64,0x6f,0x77,0x73,0x20,0x66,0x6f,0x72,0x20,0x57,0x6f,0x72,0x6b,0x67,
0x02,0x4c,0x4d,0x31,0x2e,0x32,0x58,0x30,0x30,0x32,0x00, # Requested Dialects: LM1.2X002
0x02,0x4c,0x41,0x4e,0x4d,0x41,0x4e,0x32,0x2e,0x31,0x00, # Requested Dialects: LANMAN2.1
0x02,0x4e,0x54,0x20,0x4c,0x4d,0x20,0x30,0x2e,0x31,0x32,0x00, # Requested Dialects: NT LM 0
0x02,0x53,0x4d,0x42,0x20,0x32,0x2e,0x30,0x30,0x32,0x00 # Requested Dialects: SMB 2.002


PushToTcpPort -bytearray $payload -ipaddress "127.0.0.1" -port 445
```

And here's a function for pushing a Byte[] array to a UDP port.

```
function PushToUdpPort {
#####################################################################
#.Synopsis
# Send byte array over UDP to IP address and port number.
#.Parameter ByteArray
# Array of [Byte] objects for the UDP payload.
#.Parameter IP
# IP address or FQDN of the destination host.
#.Parameter Port
# UDP port number at destination host.
#.Example
#
# [byte[]] $payload = 0x41, 0x42, 0x43, 0x44, 0x45
# PushToUdpPort $payload -ip "www.sans.org" -port 1531
#
#####################################################################


[CmdletBinding()]
Param ( [Parameter(Mandatory = $True)] [Byte[]] $ByteArray,
 [Parameter(Mandatory = $True)] [String] $IP,
 [Parameter(Mandatory = $True)] [Int] $Port
 )


$UdpClient = New-Object System.Net.Sockets.UdpClient
$UdpClient.Connect($IP,$Port)
$UdpClient.Send($ByteArray, $ByteArray.length) | out-null
}
```

Misc Notes

The "0xFF,0xFE" bytes at the beginning of a Unicode text file are byte order marks to indicate the use of little-endian UTF-

"0x0D" and "0x0A" are the ASCII carriage-return and linefeed ASCII bytes, respectively, which together represent a Windo style newline delimiter. This Windows-style newline delimiter in Unicode is "0x00,0x0D,0x00,0x0A". But in Unix-like system the ASCII newline is just "0x0A", and older Macs use "0x0D", so you will see these formats too; but be aware that many cmdlets will do on-the-fly conversion to Windows-style newlines (and possibly Unicode conversion too) when saving back t disk. When hashing text files, be aware of how the newlines and encoding (ASCII vs. Unicode) may have changed, since " same" text will hash to different thumbprints if the newlines or encoding have unexpectedly changed.

What Else?

I'm sure there are other byte array functions missing here, so what else should be added? And if you find a bug, please let know!

 Permalink | Comments RSS Feed - Post a comment | Trackback URL