Computer and Information Security
(ECE590-04, Fall 2018, Duke Univ., Prof. Tyler Bletsch)
Homework 2

Name: _Rijish_Ganguly

Duke NetID: rg239

Instructions - read all carefully:

- **DON'T SCREW UP:** Read each question carefully and be sure to answer <u>all</u> parts. Some questions are a mix of explanation and questions, so pay close attention to where you are being asked for something. It is recommended to answer the questions in the order they are asked, as they build on each other.
- **COMPUTERS YOU WILL NEED:**
    - The assignment will make use of the four computers described below.
    - VMs you already created using the Duke VCM service:
        - The **Ubuntu 18.04** VM; which we'll call your **Linux VM**.
        - The **Windows 10** VM; which we'll call your **Windows VM**.
    - A Duke VM pre-created for you running **Kali Linux**. This is your **Kali VM**.
    - Your own machine on Duke wifi: your **personal computer** (any OS).
- **WRITTEN PORTION DIRECTIONS:**
    - This assignment is designed to be copied into a new document so you can answer questions inline (either as a Google doc or in a local word processor).
    - This assignment should be submitted as a **<u>PDF</u> through Gradescope**. Other formats or methods of submission will not be accepted.
    - When you submit, the tool will ask you to mark which pages contain which questions. This is easiest if you avoid having two questions on one page and keep the large question headers intact. Be sure to mark your answer pages appropriately.
- **PROGRAMMING PORTION DIRECTIONS:**
    - There is a small programming project in this assignment; **your code for this will be submitted as a separate file** via the **Sakai assignment facility**. See the question itself for details.
- **CITE YOUR SOURCES:** Make sure you document any resources you may use when answering the questions, including classmates and the textbook. Please use authoritative sources like RFCs, ISOs, NIST SPs, man pages, etc. for your references.

This assignment is adapted from material by Samuel Carter (NCSU).

Question 0: Accessing the Homework (0 points, but necessary)

Homework 2 was encrypted with two layers of encryption, but if you’re reading this, you’ve already solved that.

Question 1: Show your work from Question 0 (2 points)

Below, **paste the commands needed to complete Question 0.**

```
ssh -i deployment_key.txt root@kali-vcm-18.vm.duke.edu
openssl rsautl -decrypt -inkey privkey.pem -in secret-rg239.key.enc -out key.txt
openssl enc -d -aes-256-cbc -in rg239.dat -out rg239 -pass file:key.txt
sudo nmap -sP -n 152.3.53.103
```

Give the MD5 hash of your decrypted secret key.

5d572a9b9b79187b2829ff2dbe2e94ca

Question 2: Chapters 2, 20, and 21 - Cryptography (18 points)

(Based in part on review questions from the textbook)

a. What are the inputs and output of a symmetric cipher's encryption function? Its decryption function?

   Symmetric cipher uses symmetric algorithms to encrypt and decrypt data. The inputs to a symmetric cipher encryption function are plaintext input and secret key shared by the sender and recipient. The output is the ciphertext which gets transmitted. The inputs to a symmetric cipher decryption function are the transmitted ciphertext and the secret key shared by the sender and recipient. The output is the plaintext.

b. How many keys are required for two people to communicate via a symmetric cipher?

   Only one key is required for two people to communicate via a symmetric cipher. The sender and receiver must obtain copies of the secret key and secure them in a responsible manner.

c. What is a message authentication code?

   A message authentication code is a piece of information that is used to authenticate a message, i.e. to confirm that the message came from the stated sender and has not been modified by an external source.

d. In your own words, describe how each of the three MAC schemes in textbook figure 2.5 work. (The figure is also in the slides; see slide 32 or so.)

   The three MAC schemes are:

   - Using symmetric encryption: For this scheme, we assume that only the sender and receiver share the encryption key. If the assumption is correct, the authenticity of the message can be assured.
   - Using public key encryption: This scheme used two separate keys known as the public key and private key. Public key is accessible to the users. The user encrypts the data using his or her private key. Anyone who possesses the corresponding public key will be able to decrypt the transmitted message.
   - Using secret value: In this technique, we use a hash function but no encryption for message authentication. This scheme assumes that the two communicating parties share a secret common key. The secret key is used to generate a hash code. The hash function is computed using the message and key on one end and transmitted to the other end along with the message. Since the other end already possesses the key it can recompute the hash function and verify the transmitted message.

e. What properties must a hash function have to be useful for message authentication?

   To be used for message authentication, a hash function H must have the following properties:

   - H can be applied to a block of data of any size

- H should produce a fixed length output
- The hash should be easy to compute
- It should be impossible to calculate x such that $H(x) = h$
- It should be computationally infeasible to find $y \neq x$ such that $H(y) = H(x)$
- H should have strong collision resistance i.e. it should be impossible to find a pair (x, y) for which $H(x) = H(y)$

f. What problem with symmetric ciphers is solved by asymmetric ciphers?

For symmetric ciphers, the key needs to be transported. Every means of electronic communication is insecure as it is impossible to guarantee that no one will be able to tap communication channels. For asymmetric ciphers, there is no need for exchanging keys, thus eliminating the problem of key exchange. Symmetric ciphers cannot provide digital signatures that cannot be repudiated. Asymmetric ciphers can provide digital signatures which can be repudiated.

g. What are the inputs and output of an asymmetric cipher's encryption function? Its decryption function?

The inputs to an asymmetric cipher's encryption function are plain text input and the public key and the output is the encrypted ciphertext. The inputs to the decryption function are the corresponding private key and the ciphertext. The output of the decryption function is the plaintext.

h. How many keys are required for two people to communicate via an asymmetric cipher?

Two keys are required for two people to communicate via an asymmetric cipher- the public key and the corresponding private key.

i. Describe the process of using an asymmetric cipher to send a message confidentially. Describe a threat model that this use of cryptography defeats.

The user encrypts the plaintext input using RSA algorithm and the receiver's public key. Only the intended recipient should be able to decrypt the ciphertext because only the intended recipient is in possession of the required private key. The threat model that this use of cryptography defeats is unauthorized disclosure which is the threat consequence and exposure which is the threat action.

j. Describe the process of using an asymmetric cipher to send a message with provable authenticity. Describe a threat model that this use of cryptography defeats.

The user encrypts the plaintext input using RSA algorithm and the sender's private key. If a user is able to successfully recover the plaintext from the senders' ciphertext using the sender's public key, this indicates that only the sender could have encrypted the plaintext, thus providing authentication. The threat model that this use of cryptography defeats is deception which is the threat consequence and falsification which is the threat action.

k. What is the difference between a private key and a secret key?

Secret key encryption uses a single key to both encrypt and decrypt messages which is known as the secret key. It must be present at both source and destination of transmission to allow the message to be transmitted securely and recovered. If the key fell into the hands of an attacker, they would be able to intercept and decrypt messages. In the public key encryption method, the key which is kept secret and secured is known as the private key. The private key should be present only at the receiving end of the transmission.

l. What is a digital signature?

Digital signatures are the most advanced and secure type of electronic signature. They provide the highest levels of assurance about each signer's identity and the authenticity of the documents they sign. A digital signature is hence a data dependent bit pattern, generated by an agent as a function of a file, message, or other form of data block.

m. What is a public-key certificate?

A public key certificate is a digitally signed document that serves to validate the sender's authorization and name. The document consists of a specially formatted block of data that contains the name of the certificate holder and the holder's public key, as well as the digital signature of a certification authority for authentication.

n. How can public-key encryption be used to distribute a secret key?

The sender prepares a message and generates a random key that will be used one time only. He encrypts the message using symmetric key encryption using the one-time key. Then he encrypts the one-time key using public key encryption using the receiver's public key. He then attaches the encrypted one-time key to the encrypted message and sends it to the receiver.

o. What are the two general approaches to attacking cryptography?

The two general approaches for attacking cryptography are

- Cryptanalysis: Relies on the nature of the algorithm plus some knowledge on the general characteristics of the plaintext. This type of attack exploits the characteristics of the algorithm to attempt to deduce a specific plaintext or deduce the key being used.
- Brute Force Attack: The attacker tries every possible key on a piece of cipher text until an intelligible translation into plaintext is obtained. It's a computationally heavy process.

p. For general-purpose symmetric encryption, what's a common, good algorithm to use?

For general purpose symmetric encryption, AES is a good algorithm to use.

q. For general-purpose asymmetric encryption, what's a common, good algorithm to use?

For general purpose asymmetric encryption, RSA is a good algorithm to use.

r. What does Diffie-Hellman key exchange accomplish? Describe a threat model that this use of cryptography defeats.

   Diffie–Hellman key exchange establishes a shared secret between two parties that can be used for secret communication for exchanging data over a public network. The Diffie–Hellman key exchange method allows two parties that have no prior knowledge of each other to jointly establish a shared secret key over an insecure channel. This key can then be used to encrypt subsequent communications using a symmetric key cipher. The protocol is considered secure against eavesdroppers if G and g are chosen properly. This use of cryptography prevents unauthorized disclosure.

   Sources used: Slides and the course text-book

Question 3: Chapter 3 - User Authentication (8 points)

(Based in part on review questions from the textbook)

a. In general terms, what are the three means of authenticating a user's identity?

   The three means of authenticating a user's identity are:

   - Token/password: Something the individual possesses. Example- smartcard
   - Static Biometrics: Something the individual is. E.g. Fingerprint
   - Dynamic Biometrics: Something the individual does. E.g. voice pattern

b. List and briefly describe five possible threats to the secrecy of passwords.

   Five possible threats to the secrecy of passwords are :

   - Offline dictionary attack: The attacker obtains the system password file and compares the password hashes against hashes of commonly used passwords. If a match is found, the attacker can gain access by that ID/password combination.
   - Password guessing against a single user: Attacker attempts to gain knowledge about the account holder and system password policies and uses the knowledge to guess password
   - Specific account attack: The attacker targets a specific user account and repeatedly guesses the password until the password is discovered.
   - Popular password attack: Use popular passwords and use it against a wide range of accounts
   - Exploiting user mistake: If the system assigns a password, it's difficult to remember and the user has to write it down. This gives way to the possibility of an adversary getting access to it.

c. What are three common techniques used to protect stored passwords?

   Three common techniques are:

   1. Storing salted hash of password
   2. Use password based key derivation function such as scrypt/brcypt
   3. Use a user management library to handle the storage

d. Which is more important: password complexity requirements or password expiration requirements? Why?

   Password complexity is more important that password expiration requirements. The original purpose of password expiration requirement was based on an old and outdated threat model. It was estimated it took 90 days for the average computer to crack the average password. However, today the threat model has radically changed. Most of today's "average" or "bad" passwords can be cracked in the cloud in mere seconds. Moreover, the password expiration requirement has huge behavioral cost as it requires systematic update in successive periods.

   What is MFA? What threat model does it deal with?

   Multifactor authentication (MFA) is a security system that requires more than one method of authentication from independent categories of credentials to verify the user's identity for a login or other transaction. The goal of MFA is to create a layered defense and make it more difficult for an unauthorized person to access a target such as a physical location, computing device, network or database. The threat model it deals with is unauthorized disclosure which is the threat consequence and exposure which is the threat action.

e. List and briefly describe the principal characteristics used for biometric identification.

   Principal characteristics used for biometric identification are:

   Facial characteristics: Based on relative location and shape of key facial features

   Fingerprints: Based on pattern of ridges and furrows at the surface of the fingertip

   Retinal Pattern: Based on patterns formed by veins beneath the retinal surface which are unique.

   Hand Geometry: Based on features of the hand including shape and length

   Iris: Based on the detailed structure of the iris.

   Signature: Based on handwriting patterns

Voice: Based on physical and anatomical characteristics of the speaker

f. Describe the general concept of a challenge-response protocol.

A challenge–response authentication is a family of protocols in which one party presents a question and another party must provide a valid answer to be authenticated. In the instance of password authentication, the protocol can use the password as the encryption key to transmit some randomly generated information as the challenge, whereupon the other end must return as its response a similarly encrypted value which is some predetermined function of the originally offered information, thus proving that it was able to decrypt the challenge.

g. In a challenge-response password protocol, why is a random nonce used?

Challenge-response password protocol uses random nonce to prevent a man in the middle attack and subsequent replay attack. A random number issued in an authentication protocol to ensure that old communications cannot be reused in replay attacks.

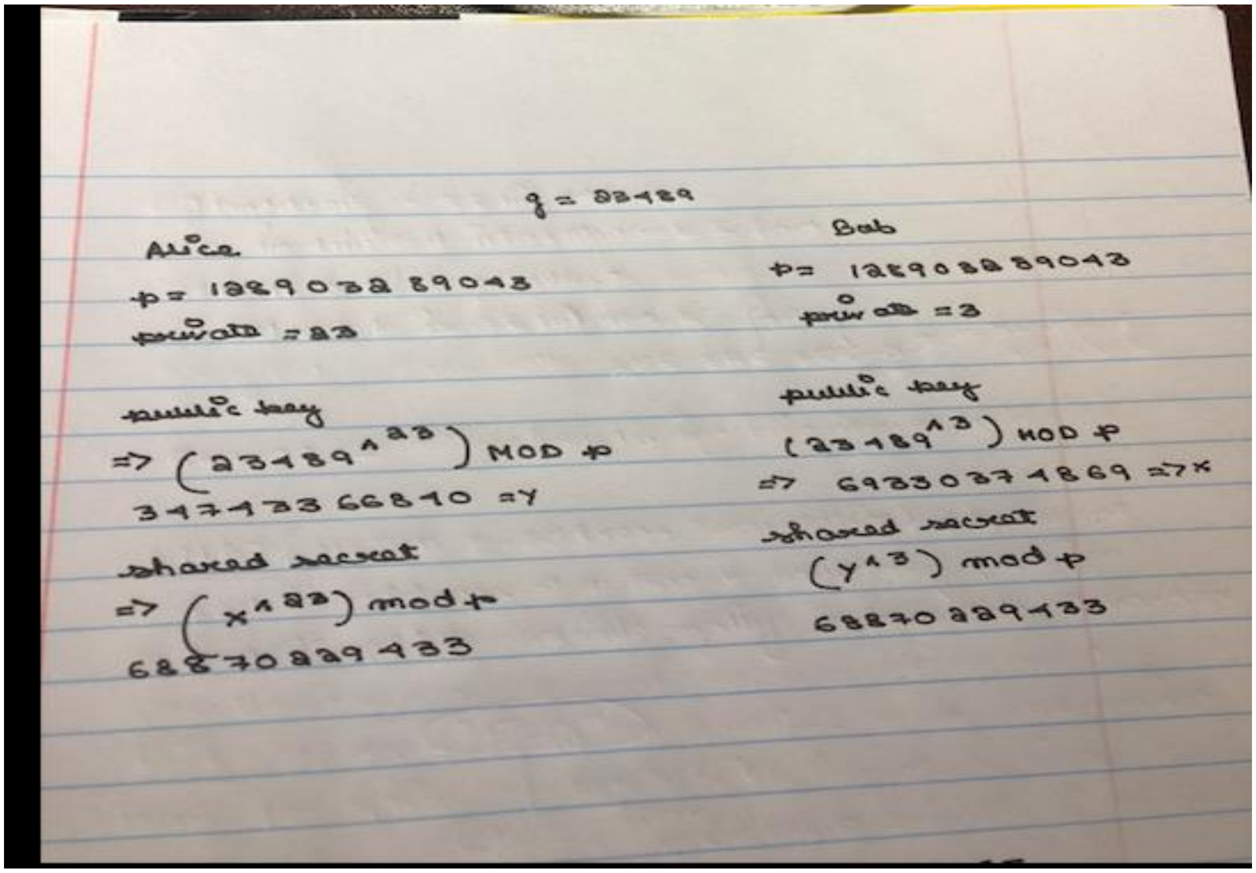
Sources used: Slides and the course text-book

Question 4: Diffie-Hellman computation (4 points)

Alice and Bob have agreed on the prime number p=128903289043 and generator g=23489. Let Alice's random number be 23 and let Bob's be 3.

Compute the shared secret key between Alice and Bob. Show your work.
Hint: Wolfram Alpha will make short work of the large exponentiation/modulo operations; normal integer arithmetic such as that used in C, Python, etc. will not work.

Used Wolfram Alpha for the calculations



$g = 23189$

Alice

$p = 1289032\ 89043$

private $= 23$

public key

$\Rightarrow (23189^{\wedge 23})$ MOD p

$3417133\ 66810 = y$

shared secret

$\Rightarrow (x^{\wedge 23})$ mod p

68870229433

Bob

$p = 1289032\ 89043$

private $= 3$

public key

$(23189^{\wedge 3})$ MOD p

$\Rightarrow 69330371869 \Rightarrow x$

shared secret

$(y^{\wedge 3})$ mod p

68870229433

Question 5: Microsoft NTLM Algorithm (8 points)

NTLM converts your Windows password to UNICODE (UTF-16LE) and then uses the MD4 hashing algorithm *without a salt* (!).

For this exercise, we want you to convert a password on your Windows VM to UNICODE (UTF-16LE) and then hash the UNICODE Hex string with MD4. The output of MD4 should match your Windows NTLM password hash. Then you'll use an online hash database to crack the hash.

Temporarily disable Windows Defender real-time protection

Some of the tools we're going to install (such as "mimikatz") have malicious capabilities and are often deployed by attackers. Therefore, the real-time monitoring provided by Windows Defender will block their download. We need to use these tool for ethical exploration, so we'll need to temporarily disable Defender.

To do so, open Windows Defender Security Center (just hit start and type 'defender security…' and it should appear as an option).

Navigate to "Virus & threat protection" and choose "Virus & threat protection settings".

Set "Real-time protection" to Off.

Additionally, if using Chrome on your Windows VM, you'll need to turn off "Safe browsing" in advanced settings, as it will block download some of the tools below.

Create target account(s) (1 point)

As our goal is to read and them crack some Windows user password hashes, we need some victim "users". (Your NetID password will not be part of this experiment, as it is not a *local* account, but is instead stored remotely using Microsoft's Active Directory system.)

In the start menu, type "users" and choose "Edit local users and groups". Navigate to the "Users" folder, right-click in an empty area of the list, and choose "New user". Call this user "**`test1`**", and provide the weak password "**`simple`**". Mark the account as "disabled" so it can't actually be used to grant access.

Make a few more accounts ("**`test2`**", "**`test3`**", etc.) with increasingly complex passwords (longer, including more character classes) as you see fit. Do not use any actual password you use anywhere else!

Note all passwords used for the test users below.

test1 - simple
test2 – password
test3 – helloworld12

Hash a password manually (3 points)

For the simple password, we must convert it to 16-bit little-endian Unicode. There is a lot to the Unicode standard, but for ASCII text, the conversion to this flavor of Unicode is as simple as appending a 00 byte to each character, converting it from 8-bit to 16-bit little-endian. So for example, the text "abc" is 616263 in ASCII hex, and 610062006300 in 16-bit little-endian Unicode hex.

Convert the test1 password ("simple") to 16-bit little-endian Unicode hex below.

730069006d0070006c006500

The Windows NTLM hash algorithm is just the classic MD4 hash algorithm applied to the above representation. You can use this web tool to do the MD4 hash of the above hex.

Using the tool, compute the MD4 of the Unicode 'simple' password, showing the result below.

c51602d46e08e6fe02b5dc5c6439e538

This operation is common enough that it's just called doing an "NTLM hash", and online tools exist for this as well.

Compute the NTLM hash directly using the tool above, pasting the result below. It should match the earlier MD4 hash.

c51602d46e08e6fe02b5dc5c6439e538

Install some tools (2 points)

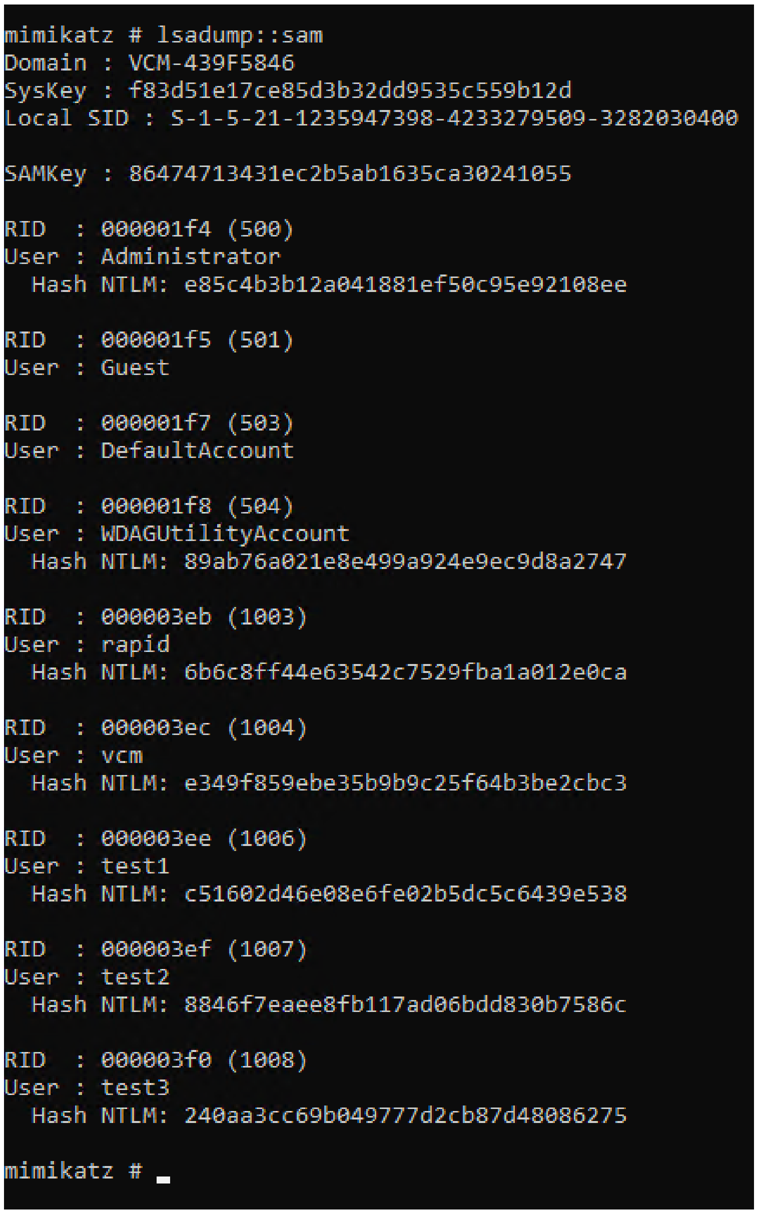
Classically, an attacker with administrator privileges would use tools like "pwdump", "samdump2", and others to print out the stored NTLM hashes of passwords. Because these hashes are unsalted, cracking them is common and fairly easy.

Microsoft has a bit of a problem here, as exchanging NTLM hashes is built into many of their protocols, so they cannot simply discard this way of storing passwords without breaking compatibility. Instead, starting in Windows 10's "Anniversary update" (August 2016), Windows encrypts the stored passwords using AES-128 using a stored random key that is readable only by processes with the rare "SYSTEM" level permission (above simple administrator).

This is similar to the classic "DRM" example from class: the OS is storing key and ciphertext on the same system, and if you have administrator privilege, you can acquire the SYSTEM level privilege to get the key, giving you both together.

One tool that can perform this is mimikatz. Install mimikatz. For dumping hashes, you will need to run it as Administrator (right click on the executable). Consult this part of the documentation on dumping hashes, noting the necessity of `token::elevate`.

Using mimikatz, dump the hashes of the users of the VM; paste a screenshot of the output here.



```
mimikatz # lsadump::sam
Domain : VCM-439F5846
SysKey : f83d51e17ce85d3b32dd9535c559b12d
Local SID : S-1-5-21-1235947398-4233279509-3282030400

SAMKey : 86474713431ec2b5ab1635ca30241055

RID  : 000001f4 (500)
User : Administrator
  Hash NTLM: e85c4b3b12a041881ef50c95e92108ee

RID  : 000001f5 (501)
User : Guest

RID  : 000001f7 (503)
User : DefaultAccount

RID  : 000001f8 (504)
User : WDAGUtilityAccount
  Hash NTLM: 89ab76a021e8e499a924e9ec9d8a2747

RID  : 000003eb (1003)
User : rapid
  Hash NTLM: 6b6c8ff44e63542c7529fba1a012e0ca

RID  : 000003ec (1004)
User : vcm
  Hash NTLM: e349f859ebe35b9b9c25f64b3be2cbc3

RID  : 000003ee (1006)
User : test1
  Hash NTLM: c51602d46e08e6fe02b5dc5c6439e538

RID  : 000003ef (1007)
User : test2
  Hash NTLM: 8846f7eaee8fb117ad06bdd830b7586c

RID  : 000003f0 (1008)
User : test3
  Hash NTLM: 240aa3cc69b049777d2cb87d48086275

mimikatz # _
```

Isolate the hashes (1 point)

The output of mimikatz will have a lot of info besides just the hashes. Copy this text to an environment with bash and use one or more shell commands to produce a file of *just* hashes. For example, if you have:

```
RID  : 000001f4 (500)
User : Administrator
  Hash NTLM: 7ee17fdad80eef8865e6710064b9e686

RID  : 000001f5 (501)
User : Guest

RID  : 000001f8 (504)
User : WDAGUtilityAccount
  Hash NTLM: 9a5c28fd8410c737d9b2c0f7f4ba4926
```

Reduce this to:

```
7ee17fdad80eef8865e6710064b9e686
9a5c28fd8410c737d9b2c0f7f4ba4926
```

Paste the command to isolate the hashes and its output below.

awk '/Hash NTLM:/ {print $3}' hash.txt

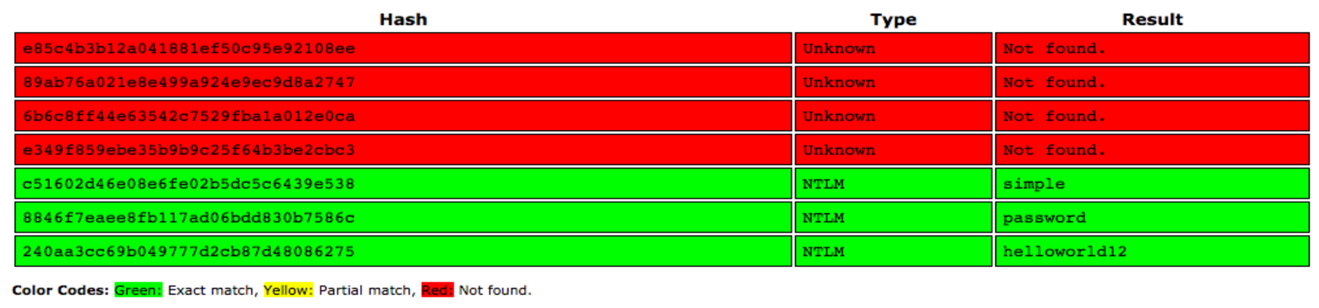
Output:

```
e85c4b3b12a041881ef50c95e92108ee

89ab76a021e8e499a924e9ec9d8a2747

6b6c8ff44e63542c7529fba1a012e0ca

e349f859ebe35b9b9c25f64b3be2cbc3

c51602d46e08e6fe02b5dc5c6439e538

8846f7eaee8fb117ad06bdd830b7586c

240aa3cc69b049777d2cb87d48086275
```

Crack the hashes (1 point)

Take all the hashes and submit them all together to an online rainbow table database such as hashkiller.co.uk or crackstation.net. You should certainly be able to crack the test1 password (“simple”).

Paste a screenshot of this. How many passwords were cracked?



Hash	Type	Result
e85c4b3b12a041881ef50c95e92108ee	Unknown	Not found.
89ab76a021e8e499a924e9ec9d8a2747	Unknown	Not found.
6b6c8ff44e63542c7529fba1a012e0ca	Unknown	Not found.
e349f859ebe35b9b9c25f64b3be2cbc3	Unknown	Not found.
c51602d46e08e6fe02b5dc5c6439e538	NTLM	simple
8846f7eaee8fb117ad06bdd830b7586c	NTLM	password
240aa3cc69b049777d2cb87d48086275	NTLM	helloworld12

Color Codes: Green: Exact match, Yellow: Partial match, Red: Not found.

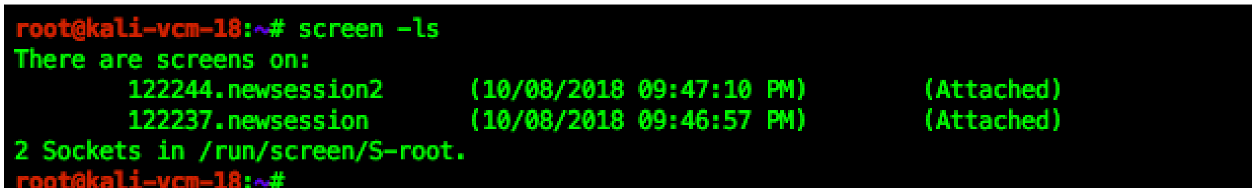
Question 6: GNU Screen (2 points)

Screen is a full-screen window manager that multiplexes a physical terminal (or GUI terminal window) between several processes, typically interactive shells. When screen is called, it creates a single terminal with a shell in it (or the specified command) and then gets out of your way so that you can use the program as you normally would. Then, at any time, you can create new (full-screen) terminals with other programs in them (including more shells), kill the current window, view a list of the active windows, turn output logging on and off, view the scrollback history, switch between terminals, etc. All terminals run their programs completely independent of each other. Especially useful for us is the fact you can *detach* a running screen session from your console and let all the associated terminals keep running in the background, then later *reattach* it to another console. Programs continue to run even when screen is detached.

Review this very brief video intro to screen. Here is a screen quick reference.

This might be useful for the MD5 cracking question below, and it's essential for the problem "John the Ripper" after that. Start a new screen session on your Kali VM. Prove to yourself that you can detach, reattach, and are generally comfortable with screen.

Paste a screenshot below of `screen --list` two separate screen sessions running.



```
root@kali-vcm-18:~# screen -ls
There are screens on:
        122244.newsession2      (10/08/2018 09:47:10 PM)        (Attached)
        122237.newsession       (10/08/2018 09:46:57 PM)        (Attached)
2 Sockets in /run/screen/S-root.
root@kali-vcm-18:~#
```

Question 7: Cracking MD5 hashes with hashcat (6 points)

There is an FTP server called Serv-U. It is actually a commercial program written by CATsoft. Hackers sometimes install pirated copies on computers they compromise. This allows the attacker to access the file system and run commands on the compromised computer. The FTP server usernames and passwords (used by the attackers) are stored in a configuration file. The passwords are hashed with MD5 hashing algorithm. Here is a sample file from a compromised machine:

```
[USER=admin|1]
Password=ly5cf2ff593419bcf3d22c62e65a82128a
HomeDir=c:\
AlwaysAllowLogin=1
TimeOut=600
Maintenance=System
Access1=C:\|RWAMELCDP
```

The MD5 hash of the password is on the line that starts with 'Password='. The two letters after the equal sign are the salt as ASCII text (shown in orange above) followed by the MD5 hash in hex (shown in blue above). This salt is prepended, i.e. the Serv-U password has function is **`md5(salt+pass)`**.

Your Kali VM comes with a hash cracker called **`hashcat`**. Read up on how to use it, and have it crack the salted hash above. Tips:

- Hashcat is optimized for GPUs, but your VM doesn't have one. Override the resulting error message using the `--force` flag.
- You will need to create a "hash file" as input; the format of this file is "**`<hash>:<salt>`**".
- The default brute force mode will be fine, and should take around 30 seconds on the Kali VM.
- Found passwords are echoed in **`<hash>:<salt>:<pass>`** format and also saved to **`~/.hashcat/hashcat.potfile`**.

Paste (1) the command executed, (2) a screenshot of the output, and (3) the password itself below.

OPTIONAL: For up to 6 points of extra credit, crack each of the following Serv-U hashes.

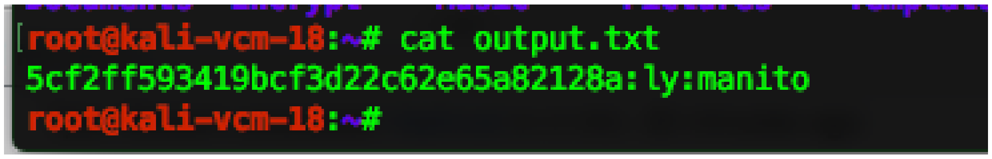
You may need to use more time, wordlists, and possibly even a combination wordlist/brute-force attack.

a. `qje39945197dbc380a5ac611fbbb7b5354`
b. `csa2eb704ec429728c1fe15814c006a4cc`
c. `qj847d6d5952baef48fa101d84a18b8de8`

Command executed:

hashcat -m 20 -a 3 --force hash.txt -o output.txt

Output:



```
root@kali-vcm-18:~# cat output.txt
5cf2ff593419bcf3d22c62e65a82128a:ly:manito
root@kali-vcm-18:~#
```

The password is **manito**

Question 8: John the Ripper (5 points)

To start, do some research, and **describe in detail (purpose and content) the /etc/passwd and /etc/shadow files on a Linux system. (2 points)**

Sources Used:

- https://www.ibm.com/support/knowledgecenter/en/ssw_aix_72/com.ibm.aix.security/passwords_etc_passwd_file.htm
- https://www.cyberciti.biz/faq/understanding-etcshadow-file/

The etc/passwd file is used to keep track of every registered user that has access to a system. The file is a colon-separate file that contains the following:

1. username
2. Encrypted password
3. User ID number
4. User's Group ID number
5. User home directory
6. Login shell

The /etc/passwd file is owned by the root user and must be readable by all the users, but only the root user has writable permissions.

The /etc/shadow file stores actual password in encrypted format for user's account with additional properties related to user password. It stores secure user account information. All fields are separated by a colon (:) symbol. It contains one entry per line for each user listed in /etc/passwd file. The file consists of the following:

1. Username
2. Password (encrypted)
3. Last password change
4. Minimum number of days required between password changes
5. Maximum number of days password

For this exercise, we will use John the Ripper, which is already installed on your Kali VM. I'll be supplying the shadow file for the exercise.

We will be using the shadow file here: http://people.duke.edu/~tkb13/courses/ece590-sec/homework/shadow

Download this shadow file to your Kali VM. Unlike hashcat, john has pretty smart defaults and wordlists, so you can just run it against the shadow file. To improve performance, you can specify **`--fork=<N>`** to run *N* instances in parallel; set *N* to the number of CPU cores on your system (you can check with `top`, `cat /proc/cpuinfo`, etc.).

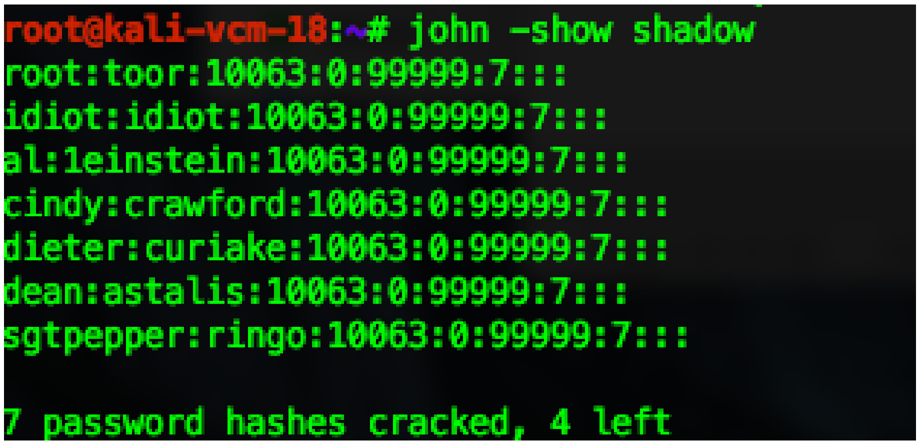
Run john on the provided shadow file in a screen session (so you can detach, disconnect, then reattach later). Let it run for at least 24 hours. It is not expected to be able to crack all the passwords, but you should get at least 6.

Paste the output your results with the following command "`john -show shadow`". (3 points)

It should look something like the example below:

```
$ john -show shadow
abcuser:apple1:15358:0:99999:7:::
defuser:orange:15364:0:99999:7:::
ghiuser:gpgtest:15373:0:99999:7:::

3 password hashes cracked, 25 left
```

Note: Here, "abcuser" is the username and "apple1" is the password.



```
root@kali-vcm-18:~# john -show shadow
root:toor:10063:0:99999:7:::
idiot:idiot:10063:0:99999:7:::
al:1einstein:10063:0:99999:7:::
cindy:crawford:10063:0:99999:7:::
dieter:curiake:10063:0:99999:7:::
dean:astalis:10063:0:99999:7:::
sgtpepper:ringo:10063:0:99999:7:::

7 password hashes cracked, 4 left
```

Question 9: Evaluating MFA approaches (6 points)

Review this article on the recent Reddit data theft attack.

a. What form of Multi-Factor Authentication (MFA) was deployed at Reddit?

   At Reddit, MDA was deployed in the form of mobile text-messages along with online password authentication.

b. What are two ways in which the attacker may have gained access to the second factor?

   Although Reddit didn't report on how the SMS codes were stolen there could be two ways the attacker might have gained access to the codes.

   SIM-swap: In this method the attacker masquerades as the target tricking the target's mobile provider into tying the customer's service to a new SIM card that the attackers control.
   Mobile number port-out scam: The attacker impersonates a customer and requests that the customer's mobile number be transferred to another mobile network provider.

   In both port-out and SIM swap schemes, the victim's phone service gets shut off and any one-time codes delivered by SMS get sent to a device that the attackers control.

c. What are two alternative forms of MFA that could be deployed instead?

   The two alternative forms of MFA that could be deployed instead are:

- The use of a mobile app such as Google Authenticator or Authy which generate the one-time code that needs to be entered in addition to a password. This method is also known as "time-based one-time password," or TOTP.
- Hardware-based security keys. These inexpensive USB-based devices allow users to complete the login process simply by inserting the device and pressing a button. After a

key is enrolled for 2FA at a particular site that supports keys, the user no longer needs to enter their password.

Question 10: Hydra (Online SSH Dictionary Attack) (4 points)

Hydra is an online network-based dictionary attack tool for SSH and many other protocols. For this example, the tool will take 4 input values: username file, a password file, target list, and a service (ssh). With that information, it attempts to perform a network-based authentication to the remote machine(s) on port 22. If it successfully authenticates with the remote machine, it shows the results in the standard output with the username and passwords that worked.

For this exercise, you are going to use Hydra to perform an SSH dictionary attack against a machine specially prepared for this purpose, **target.colab.duke.edu**. Do not use the tool on any other targets! On your Kali VM, do the following.

Acquire a password list

A password list is a text file of likely passwords. There are *many* password lists out there, and Kali Linux ships with many of them pre-installed. For our purposes, we will use the Adobe "top 100" password list, which was derived from a 2013 breach of 150 million Adobe user accounts. You can find this password list in **`/usr/share/wordlists/metasploit/adobe_top100_pass.txt`**

Build user list (1 point)

If an attacker is doing a general scan, they will typically use a common username list, such as one of these. If an attacker is focused on a known server or organization, they will do research using public information to create a list of likely usernames.

In this scenario, let's assume we are targeting an organization's senior leadership, which includes Susan Hypothetical, Scott Samplesberg, and Jacob Exampleface. Based on this, produce a small text file of likely usernames.

Paste your username list below.

ssamplesberg
ssamplesberg2
samplesbergs
samplesbergs2
jexampleface
jexampleface2
examplefacej2
examplefacej
shypothetical
shypothetical2
hypotheticals
hypotheticals2
susan
scott
jacob
susan.hypothetical
jacob.exampleface
scott.samplesberg
hypothetical.susan
exampleface.jacob
samplesberg.scott

67.159.94.115 – IP address of the target server

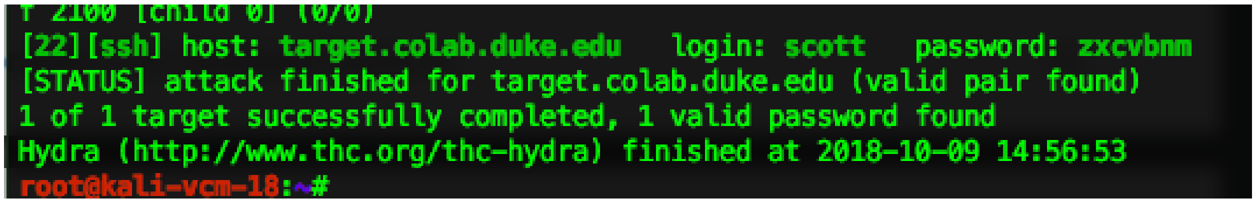
Attack (3 points)

Run hydra against target.colab.duke.edu using the resources gathered.

Give the hydra command you used below.

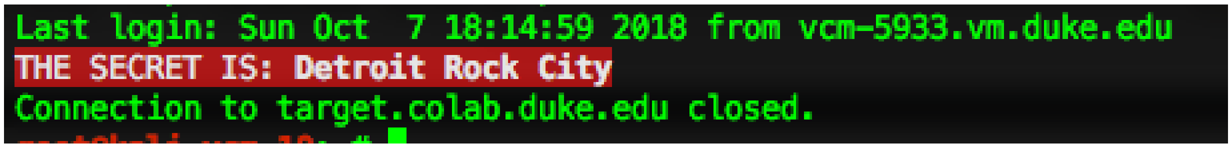
```
hydra -V -f -L userlist -P /usr/share/wordlists/metasploit/adobe_top100_pass.txt  -t 5 ssh://target.colab.duke.edu
```

Display the <u>successful</u> results of your Hydra Attack below. If you don't get any results, try more (and simpler!) usernames.



```
f 2100 [child 0] (0/0)
[22][ssh] host: target.colab.duke.edu   login: scott   password: zxcvbnm
[STATUS] attack finished for target.colab.duke.edu (valid pair found)
1 of 1 target successfully completed, 1 valid password found
Hydra (http://www.thc.org/thc-hydra) finished at 2018-10-09 14:56:53
root@kali-vcm-18:~#
```

When you log into target.colab.duke.edu using the credentials found, a secret message is displayed. What is it?



```
Last login: Sun Oct  7 18:14:59 2018 from vcm-5933.vm.duke.edu
THE SECRET IS: Detroit Rock City
Connection to target.colab.duke.edu closed.
```

OPTIONAL: For up to 5 points of extra credit, get a shell on target.colab.duke.edu.

The vulnerable account on the server is configured to print a secret bit of text and exit without providing a prompt. **Show how you can get an interactive shell using these credentials.**

Question 11: Simple Encryption Program (20 points)

Write a command-line program in the Linux environment that performs symmetric encryption using a secret key. You may use any language or library you wish. Do not implement an encryption algorithm yourself! Make use of a library to perform an existing, respected algorithm, such as AES. If called without arguments, the program should print a usage message, e.g.:

```
Syntax:
  To encrypt:  ./duke-crypter -e <input_file> <output_file>
  To decrypt:  ./duke-crypter -d <input_file> <output_file>
```

Upon being called with one of the syntaxes above, the program will prompt for the secret key on the console. Most programs hide the keys as they are typed, but that involves some weird terminal calls, so for this assignment, your program may echo the typed keys as normal.

Your program may be called something other than "`duke-crypter`", but otherwise must function as specified above. Your program must not use stdin for anything other than the secret key, though it may write to stderr or stdout.

On a successful operation, it should exit with status 0. The program should exit with a non-zero status if, during decryption, the provided secret key is not correct OR the cipher text is determined to have been tampered with. Your program should also exit with a non-zero status code if any other error occurs (file not found, etc.).

NOTE: The above implies that your program should be able to detect failures in decryption. This means that the ciphertext file should also include some form of signature of the plaintext, likely in the form of a cryptographic hash such as MD5, SHA-1, SHA-2, or SHA-3. Alternately, you may use an encryption algorithm that includes built-in integrity verification.

This assignment will be **self-grading**. Once you have a working version, download a tool called **`cryptotest.pyc`** which has been developed for this course. You can retrieve it by running:

```
  $ wget http://people.duke.edu/~tkb13/courses/ece590-sec/homework/cryptotest.pyc
```

You can run it with the syntax:

```
  $ python cryptotest.pyc <your_encryption_program>
```

This tool will perform the following steps:

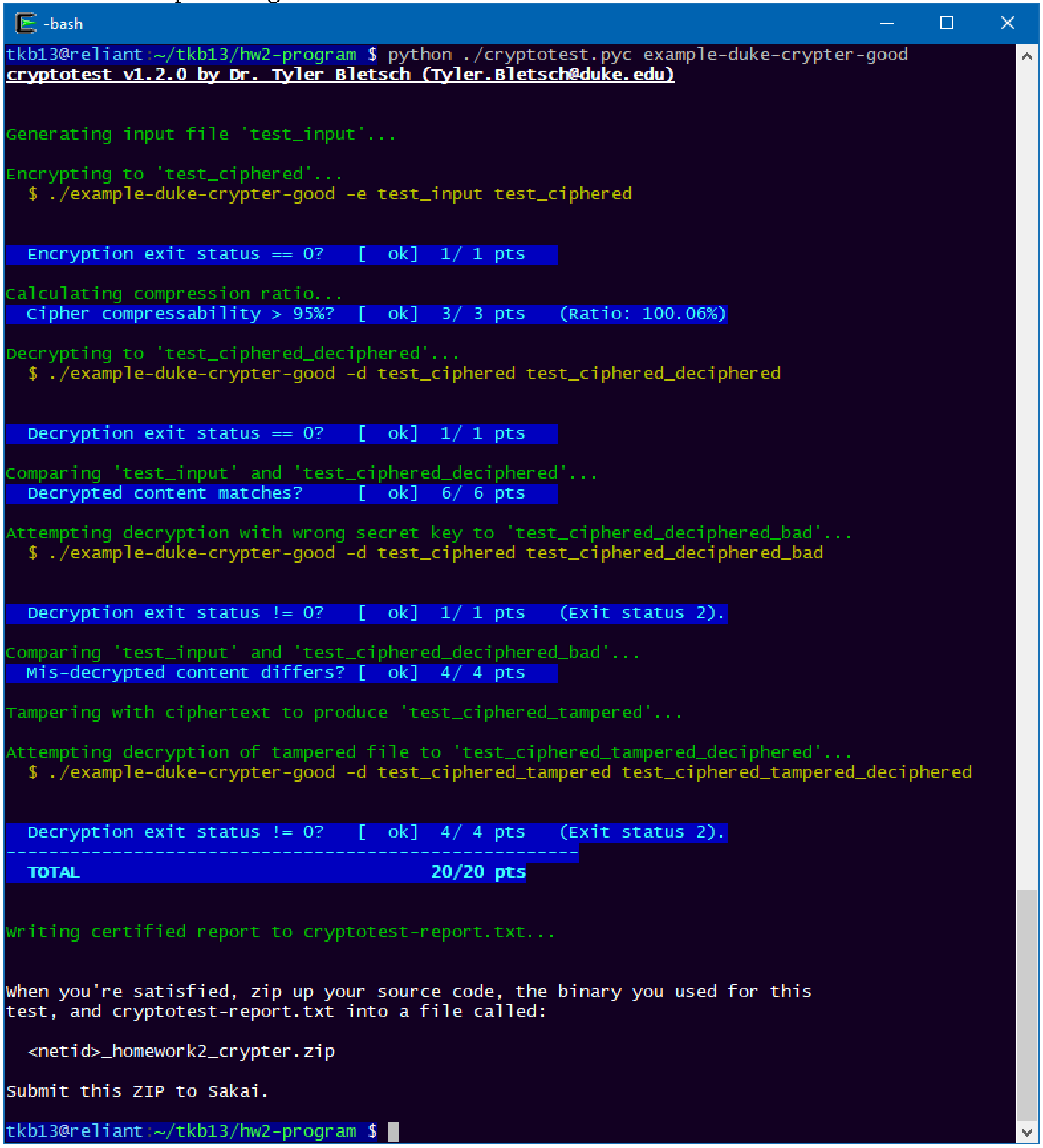
- Generate a plaintext input file
- Encrypt this plaintext file
- Examine the ciphertext and verify that it is not compressible
- Decrypt the ciphertext file

- Verify that the decrypted content matches the original plaintext

- Attempt to decrypt the ciphertext with the wrong secret key
- Verify that the exit status of this attempt is non-zero (indicating failure, i.e. that the incorrectness of the key was detected)
- Verify that the output written, if any, does not match the original plaintext

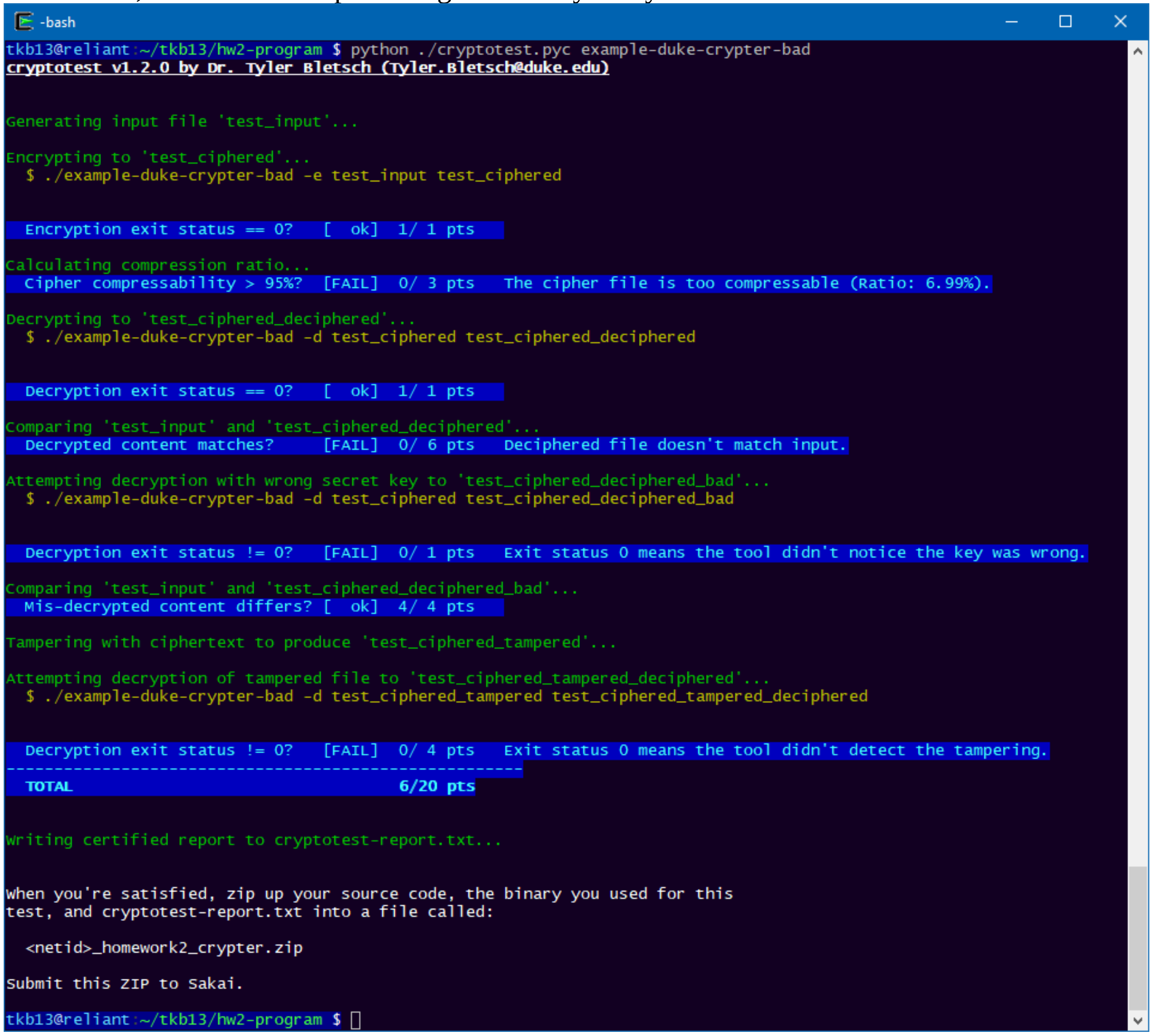
- Tamper with the ciphertext by modifying a byte
- Attempt decryption of this tampered file using the correct secret key
- Verify that the exit status of this attempt is non-zero (indicating failure, i.e. that the tampering was detected).

Here is an example run against the instructor solution:


-bash

```
tkb13@reliant:~/tkb13/hw2-program $ python ./cryptotest.pyc example-duke-crypter-good
cryptotest v1.2.0 by Dr. Tyler Bletsch (Tyler.Bletsch@duke.edu)


Generating input file 'test_input'...

Encrypting to 'test_ciphered'...
  $ ./example-duke-crypter-good -e test_input test_ciphered


  Encryption exit status == 0?   [  ok]  1/ 1 pts

Calculating compression ratio...
  Cipher compressability > 95%?  [  ok]  3/ 3 pts   (Ratio: 100.06%)

Decrypting to 'test_ciphered_deciphered'...
  $ ./example-duke-crypter-good -d test_ciphered test_ciphered_deciphered


  Decryption exit status == 0?   [  ok]  1/ 1 pts

Comparing 'test_input' and 'test_ciphered_deciphered'...
  Decrypted content matches?     [  ok]  6/ 6 pts

Attempting decryption with wrong secret key to 'test_ciphered_deciphered_bad'...
  $ ./example-duke-crypter-good -d test_ciphered test_ciphered_deciphered_bad


  Decryption exit status != 0?   [  ok]  1/ 1 pts   (Exit status 2).

Comparing 'test_input' and 'test_ciphered_deciphered_bad'...
  Mis-decrypted content differs? [  ok]  4/ 4 pts

Tampering with ciphertext to produce 'test_ciphered_tampered'...

Attempting decryption of tampered file to 'test_ciphered_tampered_deciphered'...
  $ ./example-duke-crypter-good -d test_ciphered_tampered test_ciphered_tampered_deciphered


  Decryption exit status != 0?   [  ok]  4/ 4 pts   (Exit status 2).
  -----------------------------------------------------
  TOTAL                                  20/20 pts


Writing certified report to cryptotest-report.txt...


When you're satisfied, zip up your source code, the binary you used for this
test, and cryptotest-report.txt into a file called:

  <netid>_homework2_crypter.zip

Submit this ZIP to Sakai.

tkb13@reliant:~/tkb13/hw2-program $
```

To contrast, here is an example run against a very lousy solution:



```
tkb13@reliant:~/tkb13/hw2-program $ python ./cryptotest.pyc example-duke-crypter-bad
cryptotest v1.2.0 by Dr. Tyler Bletsch (Tyler.Bletsch@duke.edu)


Generating input file 'test_input'...

Encrypting to 'test_ciphered'...
  $ ./example-duke-crypter-bad -e test_input test_ciphered


  Encryption exit status == 0?   [  ok]  1/ 1 pts

Calculating compression ratio...
  Cipher compressability > 95%?  [FAIL]  0/ 3 pts   The cipher file is too compressable (Ratio: 6.99%).

Decrypting to 'test_ciphered_deciphered'...
  $ ./example-duke-crypter-bad -d test_ciphered test_ciphered_deciphered


  Decryption exit status == 0?   [  ok]  1/ 1 pts

Comparing 'test_input' and 'test_ciphered_deciphered'...
  Decrypted content matches?     [FAIL]  0/ 6 pts   Deciphered file doesn't match input.

Attempting decryption with wrong secret key to 'test_ciphered_deciphered_bad'...
  $ ./example-duke-crypter-bad -d test_ciphered test_ciphered_deciphered_bad


  Decryption exit status != 0?   [FAIL]  0/ 1 pts   Exit status 0 means the tool didn't notice the key was wrong.

Comparing 'test_input' and 'test_ciphered_deciphered_bad'...
  Mis-decrypted content differs? [  ok]  4/ 4 pts

Tampering with ciphertext to produce 'test_ciphered_tampered'...

Attempting decryption of tampered file to 'test_ciphered_tampered_deciphered'...
  $ ./example-duke-crypter-bad -d test_ciphered_tampered test_ciphered_tampered_deciphered


  Decryption exit status != 0?   [FAIL]  0/ 4 pts   Exit status 0 means the tool didn't detect the tampering.
-----------------------------------------------------
  TOTAL                                  6/20 pts


Writing certified report to cryptotest-report.txt...


When you're satisfied, zip up your source code, the binary you used for this
test, and cryptotest-report.txt into a file called:

  <netid>_homework2_crypter.zip

Submit this ZIP to Sakai.

tkb13@reliant:~/tkb13/hw2-program $ 
```

The tool produces a report a report called "**`cryptotest-report.txt`**" which contains content similar to the following:

```
cryptotest v1.2.0 by Dr. Tyler Bletsch (Tyler.Bletsch@duke.edu)
= Certified results report =

Binary under test: ./example-duke-crypter-good
Test points: (1, 3, 1, 6, 1, 4, 4)
Total points: 20 / 20
Current username: tkb13
Current hostname: reliant.colab.duke.edu
Timestamp: 2018-09-26 21:29:06.601592

Signatures:
```

```
a930a627634fc9a2bb3a592cd6792bd2
e259debe710fbf5de9fe0425ff19da53
82b66934d0bb6eeb4e3aa36eaf3446df
```

To prevent tampering with this report, the signatures at the bottom are salted hashes of your binary, the cryptotest tool, and the report itself. Like the tool says in its output, when you're satisfied, zip up your source code, the binary you used for this test, and `cryptotest-report.txt` into a file called `<netid>_homework2_crypter.zip` and submit it to Sakai.

Some further tips:

- If using Java, you should write a small shell script front-end so that your program can be called without explicitly running the java virtual machine. For example, if you write MyDukeCrypter.java, the following will make an appropriate front-end script for it:
```
$ echo '#!/bin/sh'             > duke-crypter
$ echo 'java MyDukeCrypter $@' >> duke-crypter
$ chmod +x duke-crypter
$ ./duke-crypter (…whatever…)
```
- This Wikipedia page lists programming languages and associated crypto libraries capable of at least the AES algorithm. You're in no way restricted to these languages or libraries; this is just meant to serve as a starting point.

OPTIONAL: Figure out how to cheat.

If you are already familiar with reverse engineering techniques or want a challenge, it is conceivable that you could defeat the self-grading tool to have it certify arbitrary output. If you do so, please demo to the instructors for up to 10 points extra credit.

*Note: Do not *actually* cheat.*

Question 12: Shell practice (9 points)

For each of the following questions, give the command(s) and output. If the output is more than a few lines, you may truncate it.

Web file hash (2 points)

Develop a shell command to find the SHA 512 hash of the JPEG picture of the instructor's dog, found on this page. To help check your work, the last byte is 0x6d. Show the command and its output. A properly fancy solution will avoid the need to write any files to disk.

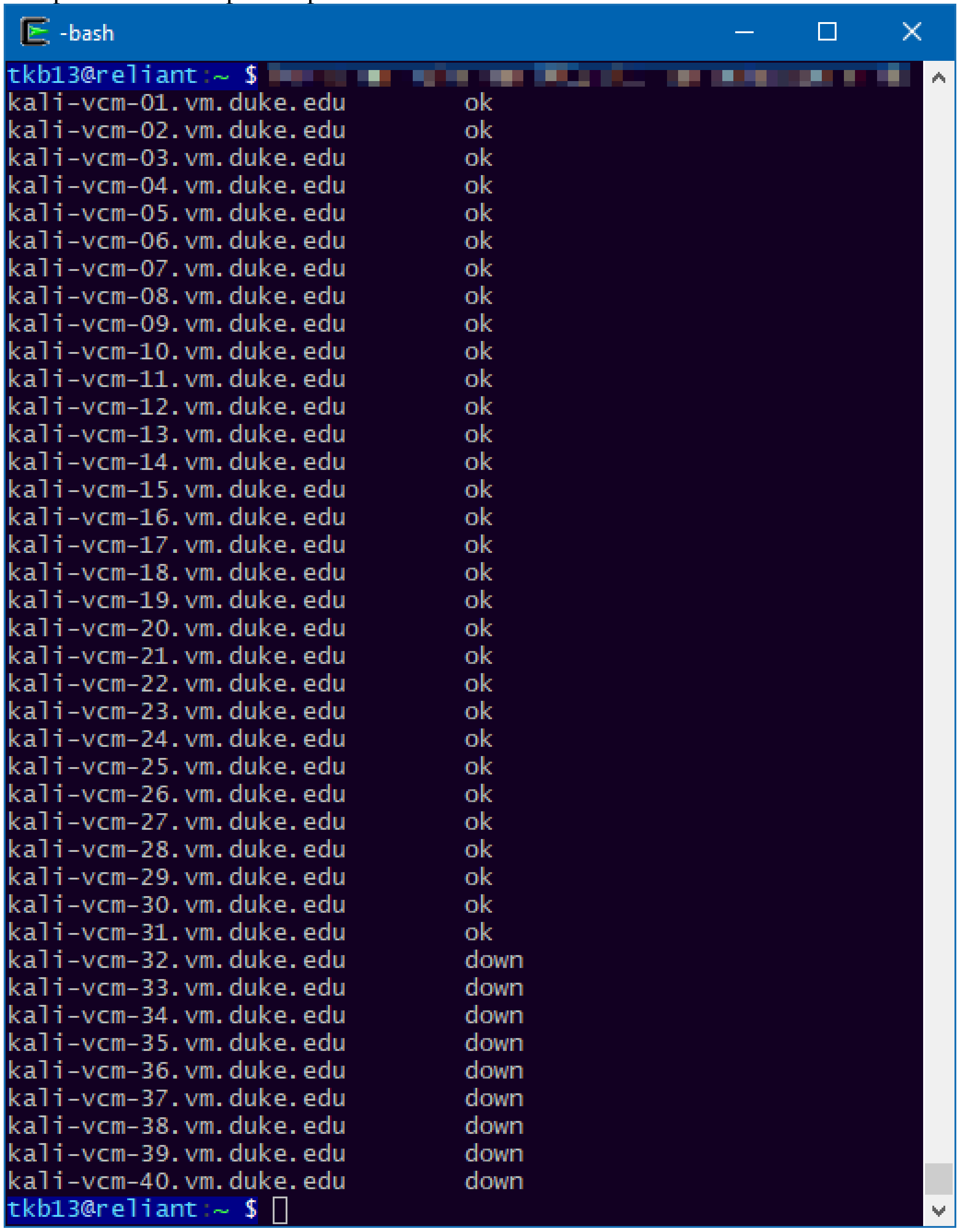
```
curl http://people.duke.edu/~tkb13/images/reg-chair.jpg |sha512sum
```

The hash is:

```
9848f6b4ae43805c111f76eead20e32aecf2e3b5a00540eb09aba74acdedd9c18acb3064
9c2c14a4e232539fddb29d1399ca3d6f006d2ddf603d89b26a991c6d
```

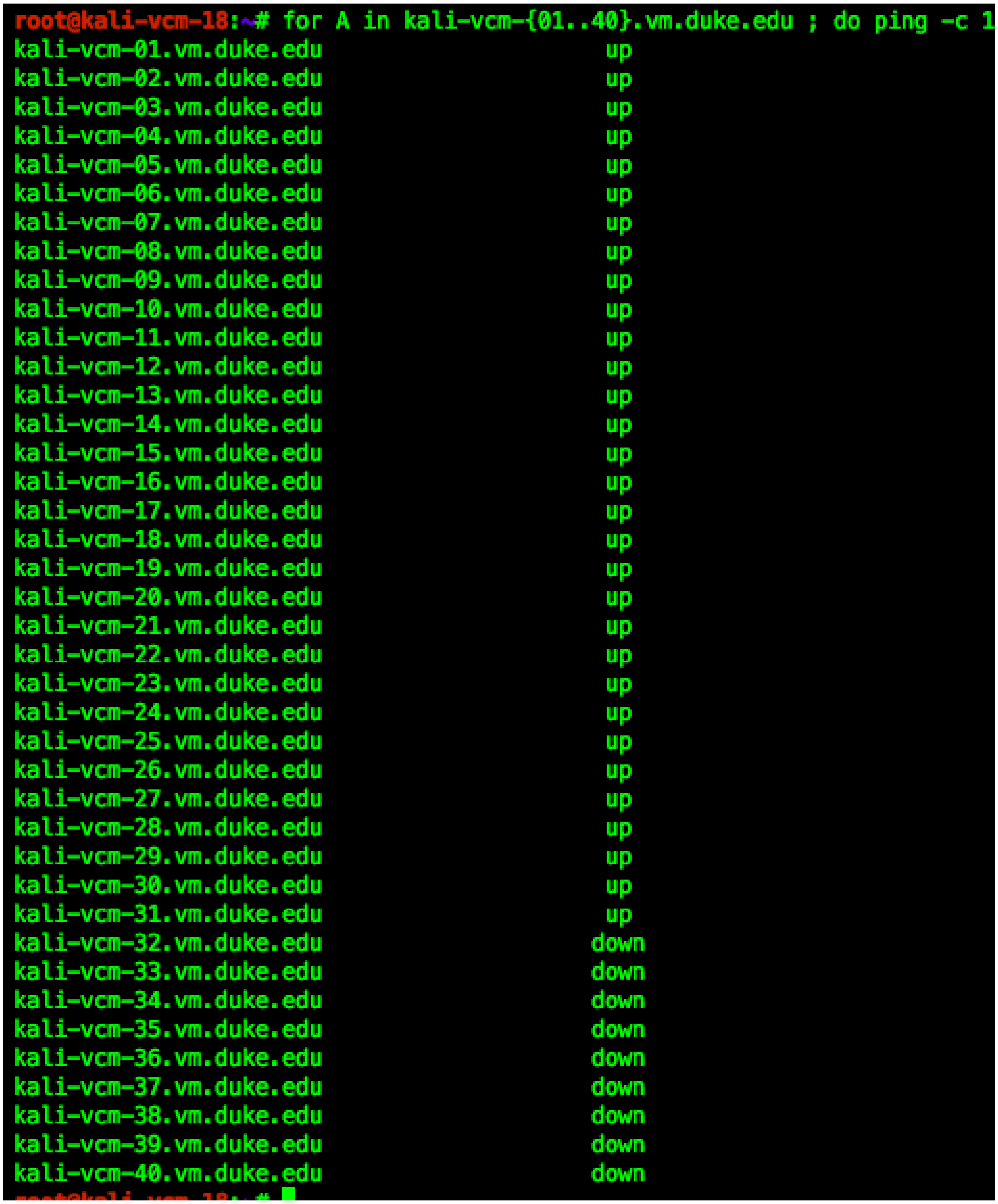
Ping check (2 points)

Develop a shell command or shell script (max 5 lines) to **`ping`** *each* of the Kali VMs with a single packet with a timeout of 1 second with an output of simply “ok” or “down” for each host, one per line. Example output:



```
tkb13@reliant:~ $ [redacted]
kali-vcm-01.vm.duke.edu         ok
kali-vcm-02.vm.duke.edu         ok
kali-vcm-03.vm.duke.edu         ok
kali-vcm-04.vm.duke.edu         ok
kali-vcm-05.vm.duke.edu         ok
kali-vcm-06.vm.duke.edu         ok
kali-vcm-07.vm.duke.edu         ok
kali-vcm-08.vm.duke.edu         ok
kali-vcm-09.vm.duke.edu         ok
kali-vcm-10.vm.duke.edu         ok
kali-vcm-11.vm.duke.edu         ok
kali-vcm-12.vm.duke.edu         ok
kali-vcm-13.vm.duke.edu         ok
kali-vcm-14.vm.duke.edu         ok
kali-vcm-15.vm.duke.edu         ok
kali-vcm-16.vm.duke.edu         ok
kali-vcm-17.vm.duke.edu         ok
kali-vcm-18.vm.duke.edu         ok
kali-vcm-19.vm.duke.edu         ok
kali-vcm-20.vm.duke.edu         ok
kali-vcm-21.vm.duke.edu         ok
kali-vcm-22.vm.duke.edu         ok
kali-vcm-23.vm.duke.edu         ok
kali-vcm-24.vm.duke.edu         ok
kali-vcm-25.vm.duke.edu         ok
kali-vcm-26.vm.duke.edu         ok
kali-vcm-27.vm.duke.edu         ok
kali-vcm-28.vm.duke.edu         ok
kali-vcm-29.vm.duke.edu         ok
kali-vcm-30.vm.duke.edu         ok
kali-vcm-31.vm.duke.edu         ok
kali-vcm-32.vm.duke.edu         down
kali-vcm-33.vm.duke.edu         down
kali-vcm-34.vm.duke.edu         down
kali-vcm-35.vm.duke.edu         down
kali-vcm-36.vm.duke.edu         down
kali-vcm-37.vm.duke.edu         down
kali-vcm-38.vm.duke.edu         down
kali-vcm-39.vm.duke.edu         down
kali-vcm-40.vm.duke.edu         down
tkb13@reliant:~ $
```

```
for A in kali-vcm-{01..40}.vm.duke.edu ; do ping -c 1 $A &> /dev/null && echo "$A
up" || echo "$A                    down" ; done
```



```
root@kali-vcm-18:~# for A in kali-vcm-{01..40}.vm.duke.edu ; do ping -c 1
kali-vcm-01.vm.duke.edu                         up
kali-vcm-02.vm.duke.edu                         up
kali-vcm-03.vm.duke.edu                         up
kali-vcm-04.vm.duke.edu                         up
kali-vcm-05.vm.duke.edu                         up
kali-vcm-06.vm.duke.edu                         up
kali-vcm-07.vm.duke.edu                         up
kali-vcm-08.vm.duke.edu                         up
kali-vcm-09.vm.duke.edu                         up
kali-vcm-10.vm.duke.edu                         up
kali-vcm-11.vm.duke.edu                         up
kali-vcm-12.vm.duke.edu                         up
kali-vcm-13.vm.duke.edu                         up
kali-vcm-14.vm.duke.edu                         up
kali-vcm-15.vm.duke.edu                         up
kali-vcm-16.vm.duke.edu                         up
kali-vcm-17.vm.duke.edu                         up
kali-vcm-18.vm.duke.edu                         up
kali-vcm-19.vm.duke.edu                         up
kali-vcm-20.vm.duke.edu                         up
kali-vcm-21.vm.duke.edu                         up
kali-vcm-22.vm.duke.edu                         up
kali-vcm-23.vm.duke.edu                         up
kali-vcm-24.vm.duke.edu                         up
kali-vcm-25.vm.duke.edu                         up
kali-vcm-26.vm.duke.edu                         up
kali-vcm-27.vm.duke.edu                         up
kali-vcm-28.vm.duke.edu                         up
kali-vcm-29.vm.duke.edu                         up
kali-vcm-30.vm.duke.edu                         up
kali-vcm-31.vm.duke.edu                         up
kali-vcm-32.vm.duke.edu                        down
kali-vcm-33.vm.duke.edu                        down
kali-vcm-34.vm.duke.edu                        down
kali-vcm-35.vm.duke.edu                        down
kali-vcm-36.vm.duke.edu                        down
kali-vcm-37.vm.duke.edu                        down
kali-vcm-38.vm.duke.edu                        down
kali-vcm-39.vm.duke.edu                        down
kali-vcm-40.vm.duke.edu                        down
root@kali-vcm-18:~#
```

Binary file analysis (3 points)

Using **`file`**, **`strings`**, and **`hd`**, let's start analyzing **`cryptotest.pyc`** from the earlier "Simple Encryption Program" question for possible reverse engineering. Answer each of the following questions and show the command(s)/output that led you to your answer.

- What format is the file?

  The file is python 2.7 byte compiled. I used the command – **file cryptotest.pyc**
  Also got to know that it's a regular file of size 8318 using the **stat** command.

- What message will likely be printed if the tool is able to compress the cipher text too much?

  The message displayed would be - The cipher file is too compressable (Ratio: %.2f%%).id.
  I got this by running the command –
  strings cryptotest.pyc | grep compressable

- What is the "magic number" of the file, as described here?

  '03f30d0a' is the magic number

  Using python

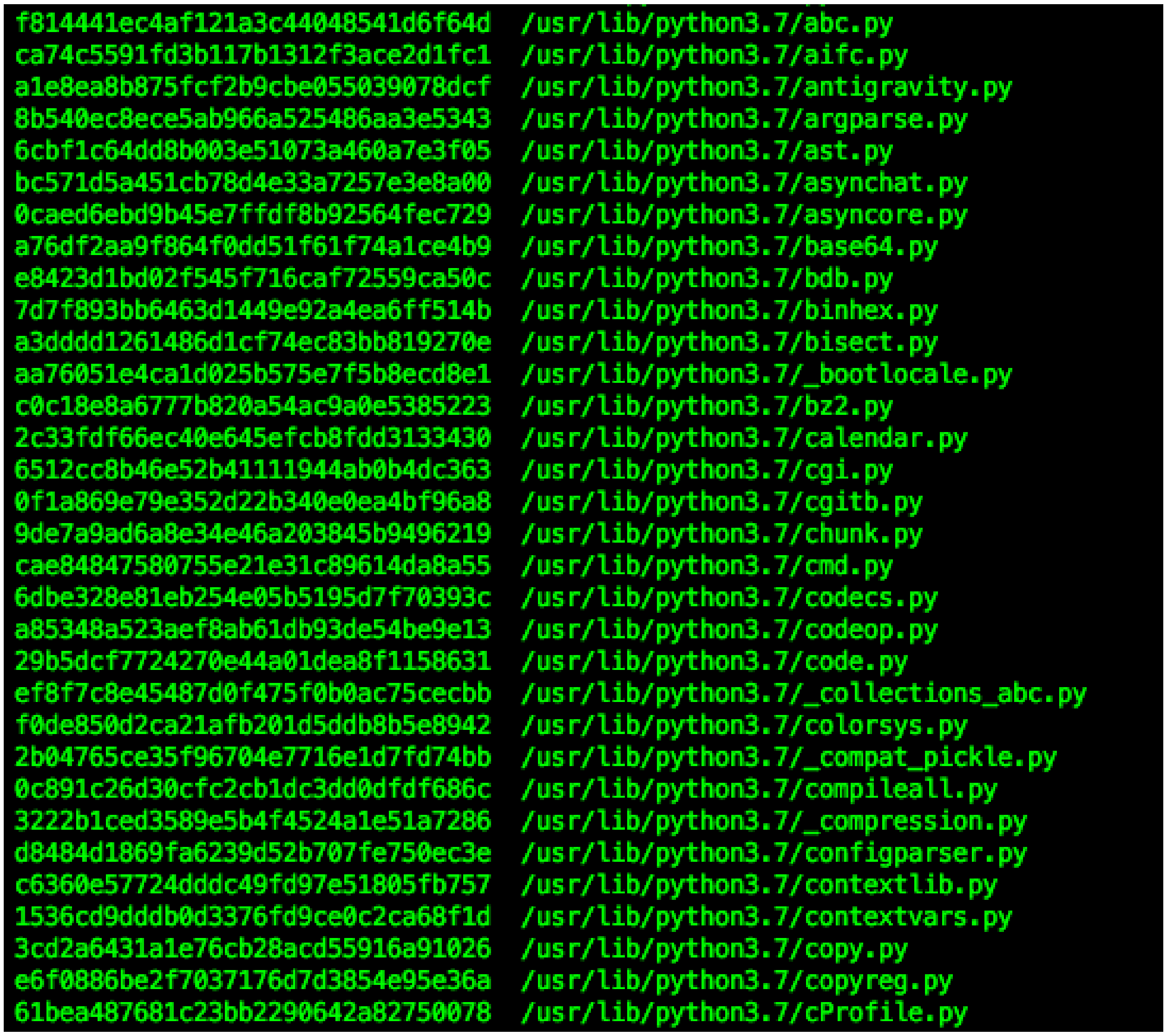
```
import imp
>>> f = open("cryptotest.pyc")
>>> magicnum = f.read(4)
>>> magicnum.encode('hex')
'03f30d0a'
```

Bulk hash (2 points)

On your Kali VM, develop a shell command that will print the MD5 hash of every `.py` file under `/usr/lib/python3.7`.

The shell command I used was:

```
for file in /usr/lib/python3.7/*.py ; do md5sum $file; done
```

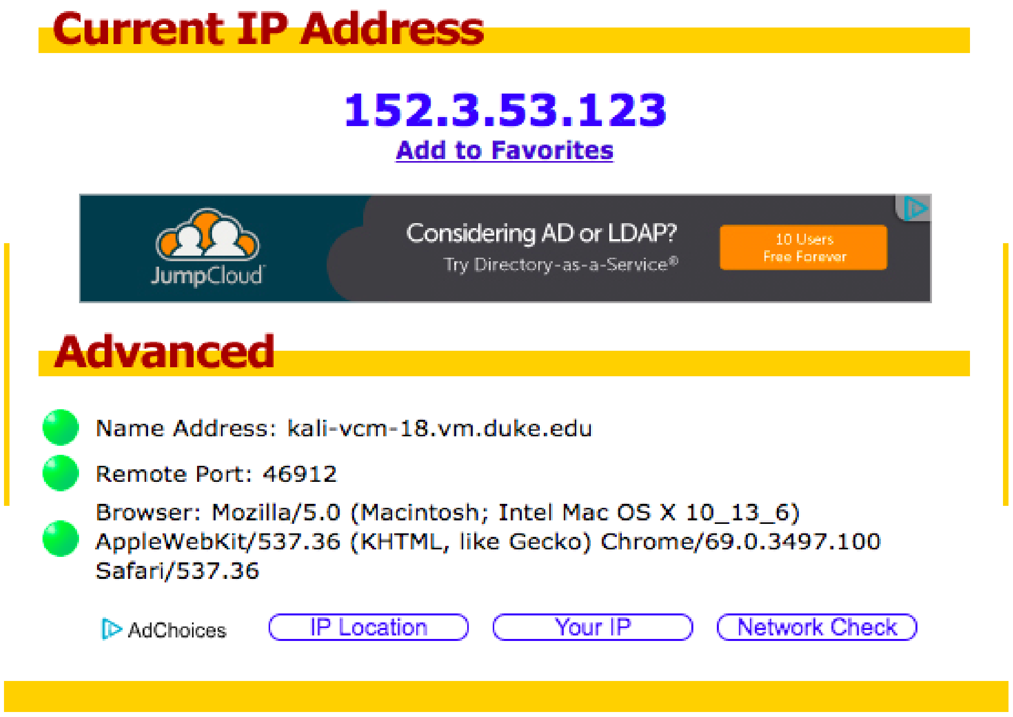


```
f814441ec4af121a3c44048541d6f64d  /usr/lib/python3.7/abc.py
ca74c5591fd3b117b1312f3ace2d1fc1  /usr/lib/python3.7/aifc.py
a1e8ea8b875fcf2b9cbe055039078dcf  /usr/lib/python3.7/antigravity.py
8b540ec8ece5ab966a525486aa3e5343  /usr/lib/python3.7/argparse.py
6cbf1c64dd8b003e51073a460a7e3f05  /usr/lib/python3.7/ast.py
bc571d5a451cb78d4e33a7257e3e8a00  /usr/lib/python3.7/asynchat.py
0caed6ebd9b45e7ffdf8b92564fec729  /usr/lib/python3.7/asyncore.py
a76df2aa9f864f0dd51f61f74a1ce4b9  /usr/lib/python3.7/base64.py
e8423d1bd02f545f716caf72559ca50c  /usr/lib/python3.7/bdb.py
7d7f893bb6463d1449e92a4ea6ff514b  /usr/lib/python3.7/binhex.py
a3dddd1261486d1cf74ec83bb819270e  /usr/lib/python3.7/bisect.py
aa76051e4ca1d025b575e7f5b8ecd8e1  /usr/lib/python3.7/_bootlocale.py
c0c18e8a6777b820a54ac9a0e5385223  /usr/lib/python3.7/bz2.py
2c33fdf66ec40e645efcb8fdd3133430  /usr/lib/python3.7/calendar.py
6512cc8b46e52b41111944ab0b4dc363  /usr/lib/python3.7/cgi.py
0f1a869e79e352d22b340e0ea4bf96a8  /usr/lib/python3.7/cgitb.py
9de7a9ad6a8e34e46a203845b9496219  /usr/lib/python3.7/chunk.py
cae84847580755e21e31c89614da8a55  /usr/lib/python3.7/cmd.py
6dbe328e81eb254e05b5195d7f70393c  /usr/lib/python3.7/codecs.py
a85348a523aef8ab61db93de54be9e13  /usr/lib/python3.7/codeop.py
29b5dcf7724270e44a01dea8f1158631  /usr/lib/python3.7/code.py
ef8f7c8e45487d0f475f0b0ac75cecbb  /usr/lib/python3.7/_collections_abc.py
f0de850d2ca21afb201d5ddb8b5e8942  /usr/lib/python3.7/colorsys.py
2b04765ce35f96704e7716e1d7fd74bb  /usr/lib/python3.7/_compat_pickle.py
0c891c26d30cfc2cb1dc3dd0dfdf686c  /usr/lib/python3.7/compileall.py
3222b1ced3589e5b4f4524a1e51a7286  /usr/lib/python3.7/_compression.py
d8484d1869fa6239d52b707fe750ec3e  /usr/lib/python3.7/configparser.py
c6360e57724dddc49fd97e51805fb757  /usr/lib/python3.7/contextlib.py
1536cd9dddb0d3376fd9ce0c2ca68f1d  /usr/lib/python3.7/contextvars.py
3cd2a6431a1e76cb28acd55916a91026  /usr/lib/python3.7/copy.py
e6f0886be2f7037176d7d3854e95e36a  /usr/lib/python3.7/copyreg.py
61bea487681c23bb2290642a82750078  /usr/lib/python3.7/cProfile.py
```

Question 13: SSH forwarding (3 points)

Using SSH, prepare a SOCKS proxy tunnel on your personal computer to your Kali VM, and configure your local web browser to use this proxy. Paste a screenshot your local browser visiting https://ipchicken.com/ showing the IP address and hostname of your Kali VM.

Command used: `ssh -i deployment_key.txt -D 2001 root@kali-vcm-18.vm.duke.edu`


Current IP Address

152.3.53.123

Add to Favorites

Considering AD or LDAP? Try Directory-as-a-Service® 10 Users Free Forever

Advanced

Name Address: kali-vcm-18.vm.duke.edu

Remote Port: 46912

Browser: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/69.0.3497.100 Safari/537.36

Question 14: Tor Project: Anonymity Online (5 points)

Tor is free software and an open network that helps you defend against traffic analysis, a form of network surveillance that threatens personal freedom and privacy, confidential business activities and relationships, and state security.

Source used: https://en.wikipedia.org/wiki/Tor_(anonymity_network)

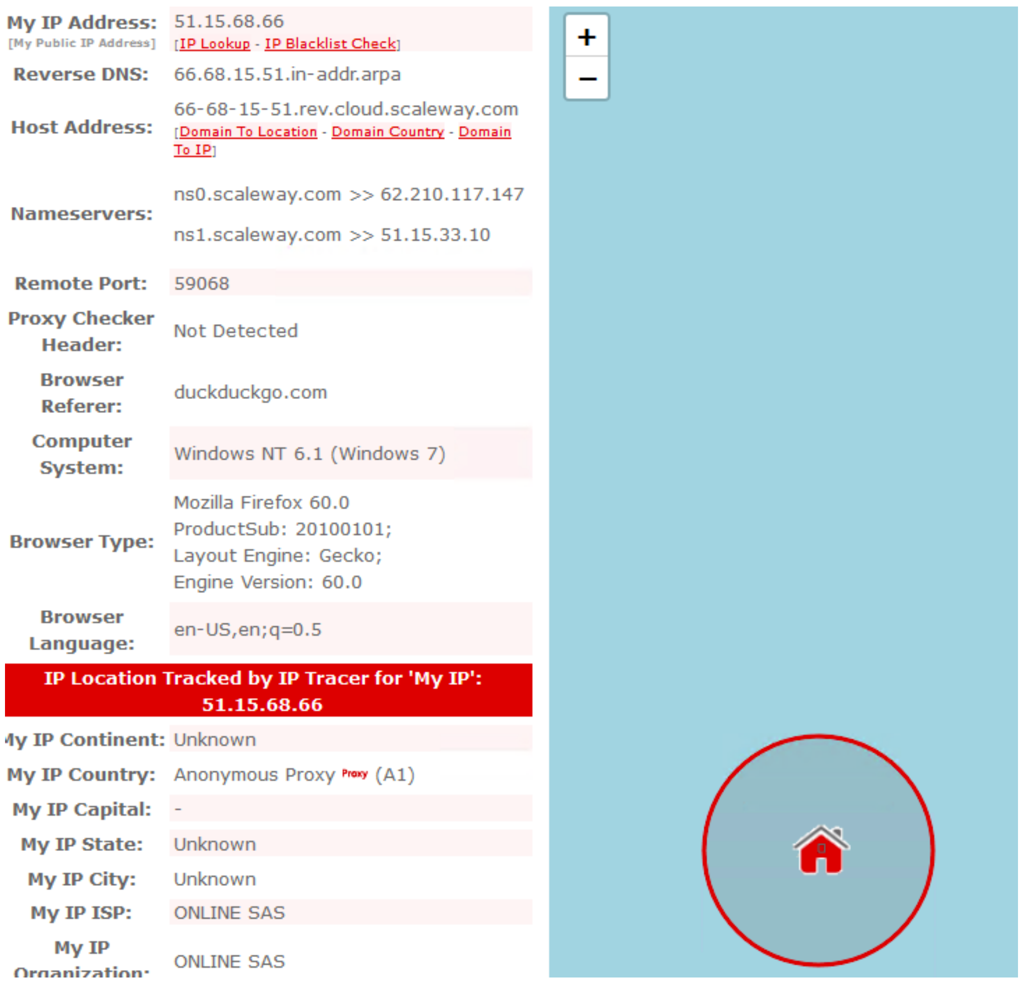
Explain what Tor is and how it can be used for both good and nefarious purposes. (2 points)

Tor is a software that enables anonymous communication by directing internet traffic through a free, worldwide, volunteer overlay network consisting of more than 7000 relays to conceal a user's location and usage from anyone conducting network surveillance.

Tor enables its users to surf the Internet, chat and send instant messages anonymously, and is used by a wide variety of people for both licit and illicit purposes. Tor is not meant to completely solve the issue of anonymity on the web. Tor is used by journalists, activists and privacy advocates as a means of evading prying eyes. However, Tor has been used for anonymous defamation, unauthorized news leaks of sensitive information, copyright infringement, distribution of illegal sexual content, selling controlled substances, weapons, and stolen credit card numbers, money laundering, bank fraud, credit card fraud, identity theft and the exchange of counterfeit currency.

Download and install the Tor Browser to your Windows VM. Start the Tor browser and **provide the IP address and hostname of the Tor exit relay through your initial circuit by visiting the site that tells you what it thinks your IP address is. (1 point)**

Use IP Tracker to get location information from a Tor and non-Tor Browser. Show screenshots of both here. Geographically, where is your Tor exit node? (2 points)



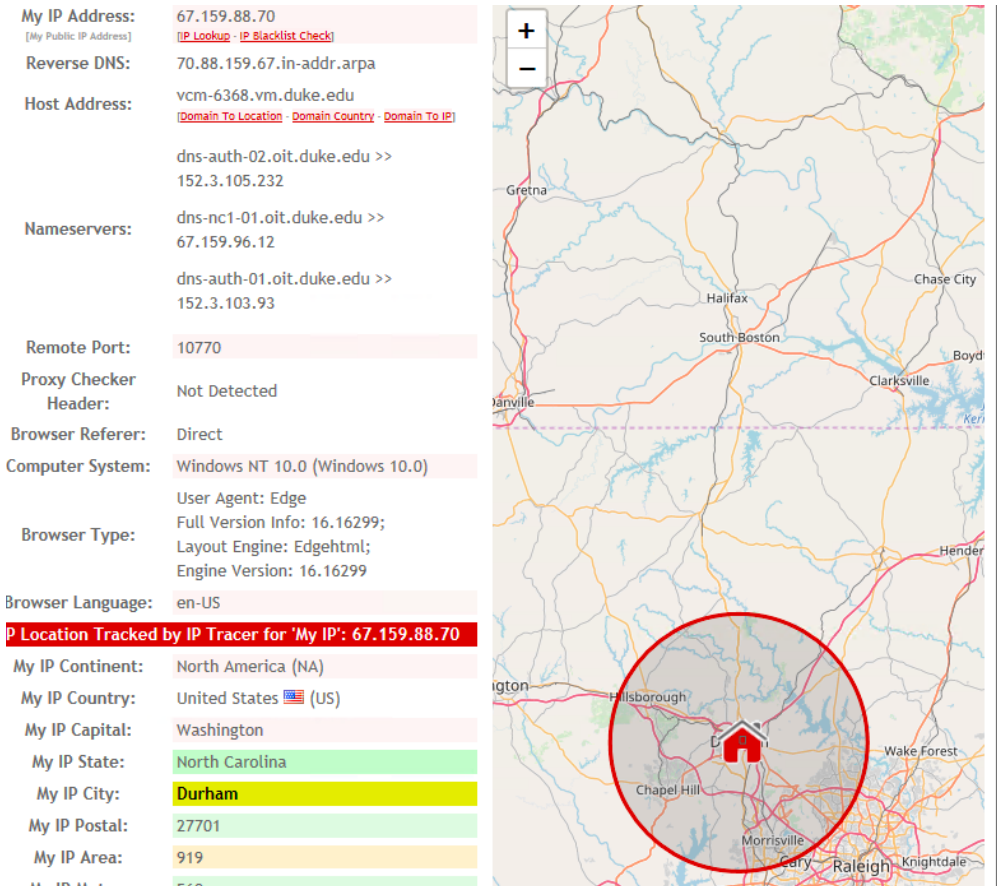
My IP Address: [My Public IP Address]	51.15.68.66 [IP Lookup - IP Blacklist Check]
Reverse DNS:	66.68.15.51.in-addr.arpa
Host Address:	66-68-15-51.rev.cloud.scaleway.com [Domain To Location - Domain Country - Domain To IP]
Nameservers:	ns0.scaleway.com >> 62.210.117.147 ns1.scaleway.com >> 51.15.33.10
Remote Port:	59068
Proxy Checker Header:	Not Detected
Browser Referer:	duckduckgo.com
Computer System:	Windows NT 6.1 (Windows 7)

Browser Type:	Mozilla Firefox 60.0
	ProductSub: 20100101;
	Layout Engine: Gecko;
	Engine Version: 60.0

Browser Language:	en-US,en;q=0.5

IP Location Tracked by IP Tracer for 'My IP': 51.15.68.66

My IP Continent:	Unknown
My IP Country:	Anonymous Proxy Proxy (A1)
My IP Capital:	-
My IP State:	Unknown
My IP City:	Unknown
My IP ISP:	ONLINE SAS
My IP Organization:	ONLINE SAS



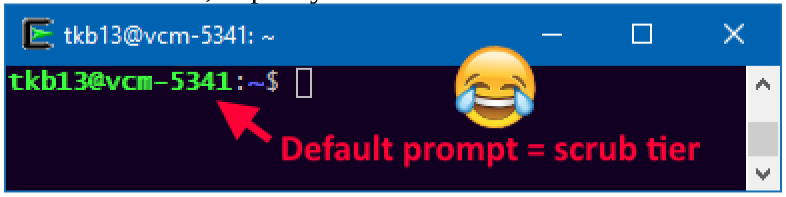
My IP Address: [My Public IP Address]	67.159.88.70 [IP Lookup - IP Blacklist Check]
Reverse DNS:	70.88.159.67.in-addr.arpa
Host Address:	vcm-6368.vm.duke.edu [Domain To Location - Domain Country - Domain To IP]
Nameservers:	dns-auth-02.oit.duke.edu >> 152.3.105.232 dns-nc1-01.oit.duke.edu >> 67.159.96.12 dns-auth-01.oit.duke.edu >> 152.3.103.93
Remote Port:	10770
Proxy Checker Header:	Not Detected
Browser Referer:	Direct
Computer System:	Windows NT 10.0 (Windows 10.0)

Browser Type:	User Agent: Edge
	Full Version Info: 16.16299;
	Layout Engine: Edgehtml;
	Engine Version: 16.16299

Browser Language:	en-US
P Location Tracked by IP Tracer for 'My IP': 67.159.88.70	
My IP Continent:	North America (NA)
My IP Country:	United States 🇺🇸 (US)
My IP Capital:	Washington
My IP State:	North Carolina
My IP City:	Durham
My IP Postal:	27701
My IP Area:	919

Question 15: Craft your prompt (3 points)

A rite of passage of UNIX users is the customization of the bash prompt. The default, even on modern Ubuntu, is pretty lame.

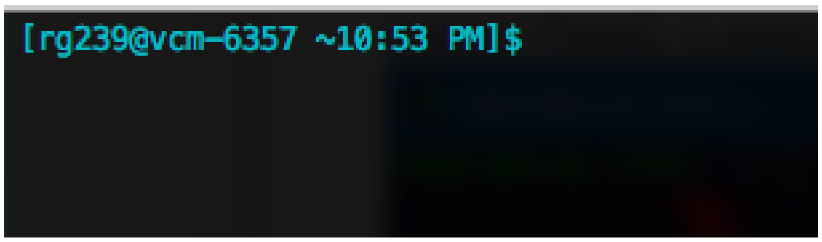


```
tkb13@vcm-5341:~$
```

Using your understanding of shell color codes, develop a custom prompt by modifying the PS1 environment variable. You'll need to mark the escape codes for the shell using `\[` and `\]` indicators (this the shell needs to know what characters are printed vs. interpreted for cursor control purposes); details are here.

Paste your PS1 environment variable command & a screenshot of the resulting prompt.

export PS1="\e[0;36m[\u@\h \W\\@]\$ \e[m "
Changed the color and added the current time.



```
[rg239@vcm-6357 ~10:53 PM]$
```

Question 16: Regular expressions (6 points)

NetID validator (2 points)

Develop a regular expression that matches if and only if the input is a valid NetID (1-3 letters, 1 or more digits).

The regex is **^\w{1,4}\d{1,}$**

URL parser (2 points)

Develop a regular expression that, for a given URL, matches if and only if it is a Wikipedia page, and it captures just the article part of the URL. Examples:

URL	Matches?	Match group 1
https://en.wikipedia.org/wiki/Computer_security	yes	Computer_security
https://duke.edu/	no	

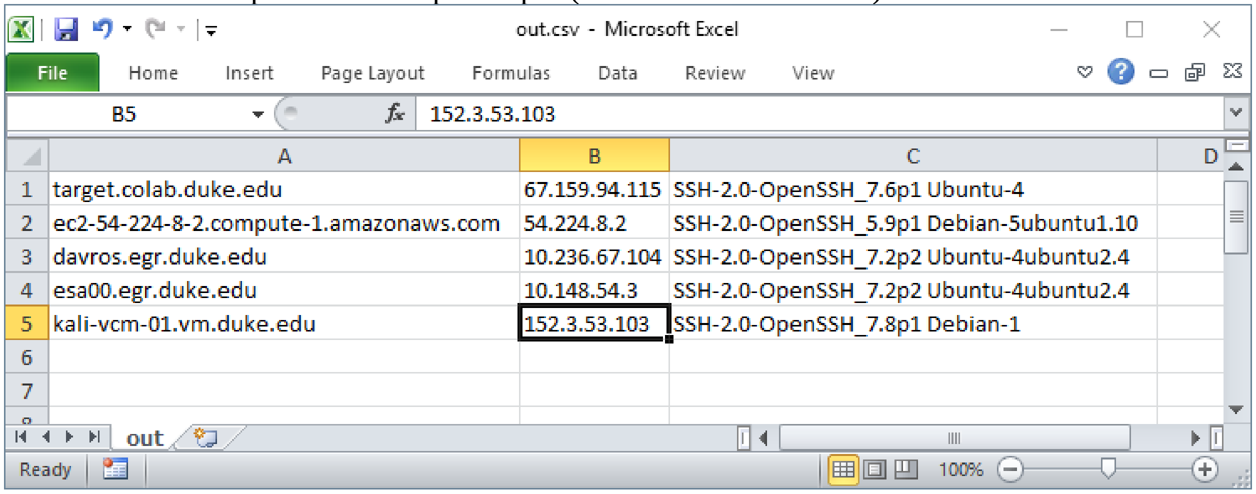
The regex is **^https?:\/\/(en\.wikipedia\.org\/wiki\/([\w%]+))**

Pi digits (2 points)

Develop a shell command to print the MD5 hash of pi (π) up to the first appearance of decimal digits "12345". To help check your work, the last byte of the answer is 0x02. You can download the value of pi, no need to compute it. For actually employing the regex, you can use grep with -E and other options, perl, Python, or other tools.

Question 17: Attack recon (6 points)

One of the first things an attacker will do when focusing on a particular target organization is conduct reconnaissance. Using your choice of tools we've learned, develop a command or script that will produce a CSV file containing the hostname, IP address, and SSH version of all the hosts listed in an input file. Example output (once loaded into Excel):



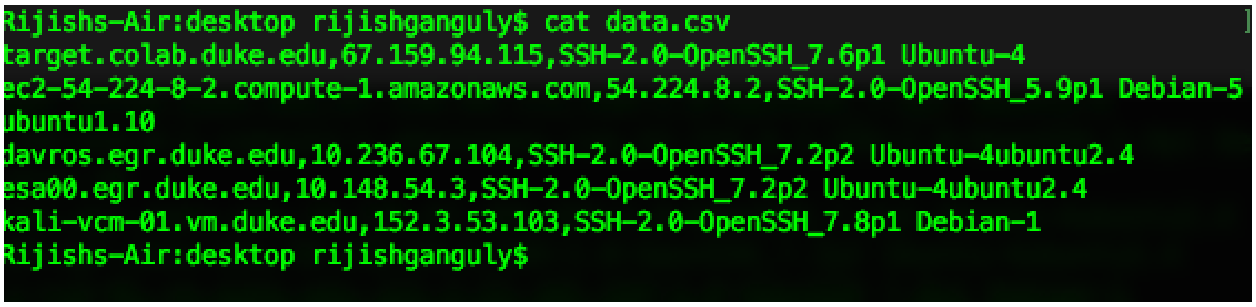
	A	B	C
1	target.colab.duke.edu	67.159.94.115	SSH-2.0-OpenSSH_7.6p1 Ubuntu-4
2	ec2-54-224-8-2.compute-1.amazonaws.com	54.224.8.2	SSH-2.0-OpenSSH_5.9p1 Debian-5ubuntu1.10
3	davros.egr.duke.edu	10.236.67.104	SSH-2.0-OpenSSH_7.2p2 Ubuntu-4ubuntu2.4
4	esa00.egr.duke.edu	10.148.54.3	SSH-2.0-OpenSSH_7.2p2 Ubuntu-4ubuntu2.4
5	kali-vcm-01.vm.duke.edu	152.3.53.103	SSH-2.0-OpenSSH_7.8p1 Debian-1

Apply this to the following hosts:

```
target.colab.duke.edu
ec2-54-224-8-2.compute-1.amazonaws.com
davros.egr.duke.edu
esa00.egr.duke.edu
kali-vcm-01.vm.duke.edu
```

(Do not scan machines other than the above. The Amazon machine listed is my EC2 instance.)

```
for H in `cat servers`; do printf "%s" "$H," >> data.csv ; printf "%s," `host $H | grep -oE "\b([0-9]{1,3}\.){3}[0-9]{1,3}\b"`>> data.csv;  echo hi | nc $H 22| head -n1 >> data.csv; done
```



```
Rijishs-Air:desktop rijishganguly$ cat data.csv
target.colab.duke.edu,67.159.94.115,SSH-2.0-OpenSSH_7.6p1 Ubuntu-4
ec2-54-224-8-2.compute-1.amazonaws.com,54.224.8.2,SSH-2.0-OpenSSH_5.9p1 Debian-5
ubuntu1.10
davros.egr.duke.edu,10.236.67.104,SSH-2.0-OpenSSH_7.2p2 Ubuntu-4ubuntu2.4
esa00.egr.duke.edu,10.148.54.3,SSH-2.0-OpenSSH_7.2p2 Ubuntu-4ubuntu2.4
kali-vcm-01.vm.duke.edu,152.3.53.103,SSH-2.0-OpenSSH_7.8p1 Debian-1
Rijishs-Air:desktop rijishganguly$
```