NYU | SCHOOL OF PROFESSIONAL STUDIES

Cyber Defense
HIGH1-CE9074
Session 3

Introduction to Personal Security

Use a Virtual Private Network (VPN) on public Wi-Fi

Would you fly on a commercial flight if you could afford a private jet? Probably not. A VPN is like your private jet. If you use a VPN, even HTTP traffic will flow encrypted from your machine. Also, your VPN provider's servers will connect to the destination on your behalf, and your machine's identity will be kept secret. If you are using a public network (like at an airport or coffee shop), use VPNs like Tunnelbear from tunnelbear.com, your school-provided VPN, or some other VPN before entering any sensitive data (like a password or an account number) while you are on a public Wi-Fi.

Why is public Wi-Fi so dangerous? A hacker can create a fake Wi-Fi network that looks and feels like a legitimate Wi-Fi. Then, if you use it, the hacker can record your keystrokes and get your usernames, passwords, and other personal information on various sites.

If the hacker also sends fake software updates to your machine and you install it, the hacker will gain complete control over your computer without your knowledge.

To make matters worse, all the tools to do all this are readily available on the internet.

Watch Kevin Mitnick hacking using fake Wi-Fi:

https://www.youtube.com/watch?v=bCaARKQ3-cg

If you have to use public Wi-Fi without a VPN, don't log into any site and don't ever update any software while on it.

Emails - SMTP and POP/IMAP

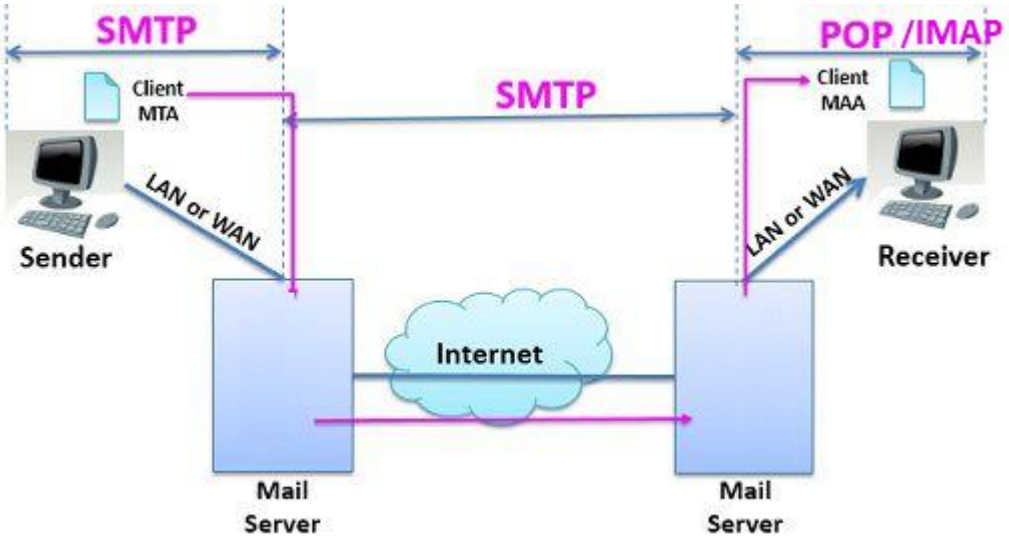
SMTP - Simple Mail Transfer Protocol

POP - Post Office Protocol.

IMAP - Internet Message Access Protocol

When you hit the send button on your email software, your email software (client) connects to your SMTP server using TCP/IP on port 25. Your SMTP server then would become a client and create another TCP/IP connection with the recipient's SMTP server on port 25 to deliver the email. However, SMTP protocol stops there, and it does not deliver the mail to the recipient.

The recipient of your email would have to use an email software (client) to connect to their server using either the POP (port 110) or IMAP (port 143) protocol to retrieve emails.



The message is sent and received in plain text, which is subject to wiretapping.

Watch Kevin Mitnick wiretapping an unsecured email connection:

https://www.youtube.com/watch?v=KcJWXpABpVo

Secured Emails

The good news is today's email software uses secured SMTP connections to send and secure POP/IMAP connections to retrieve messages. If your SMTP connections are secure, then the

port changes to 465. If your email software securely retrieves messages using POP3, the port is 995; for IMAP secured connections, the port is 993.

Even with a secured connection, your message may not stay encrypted after it reaches the server. Therefore, the server can still read your messages, and the government may also be able to read your messages.

In addition to secured connections, you can enable PGP (Pretty Good Privacy) encryption to encrypt the message body so that even the email server or anybody else can't read your messages – they deliver. Tools like FlowCrypt will allow you to encrypt your messages using PGP encryption. But, of course, your recipient must have the proper key to decrypt the messages.

What is the WWW?

The web consists of billions of browsers (clients) and web servers (servers) connected through wired and wireless networks using TCP/IP Socket connections.

What is HTML?

Hypertext Markup Language (HTML) is a predefined set of tags to construct our web pages. HTML is a presentation tool. We use HTML tags to define attributes like font, color, style of text, size, locations of images, generate links to other pages, and so on.

A web page is a text file containing data tagged by HTML tags. The browser opens such a file and decorates the content of the file as desired by the HTML tags.

What is HTTP?

HTTP ((Hypertext Transfer Protocol) has nothing to do with HTML other than the fact that these two technology names start with "Hypertext" and are often used together. HTTP is all about transferring text data from point A to point B.

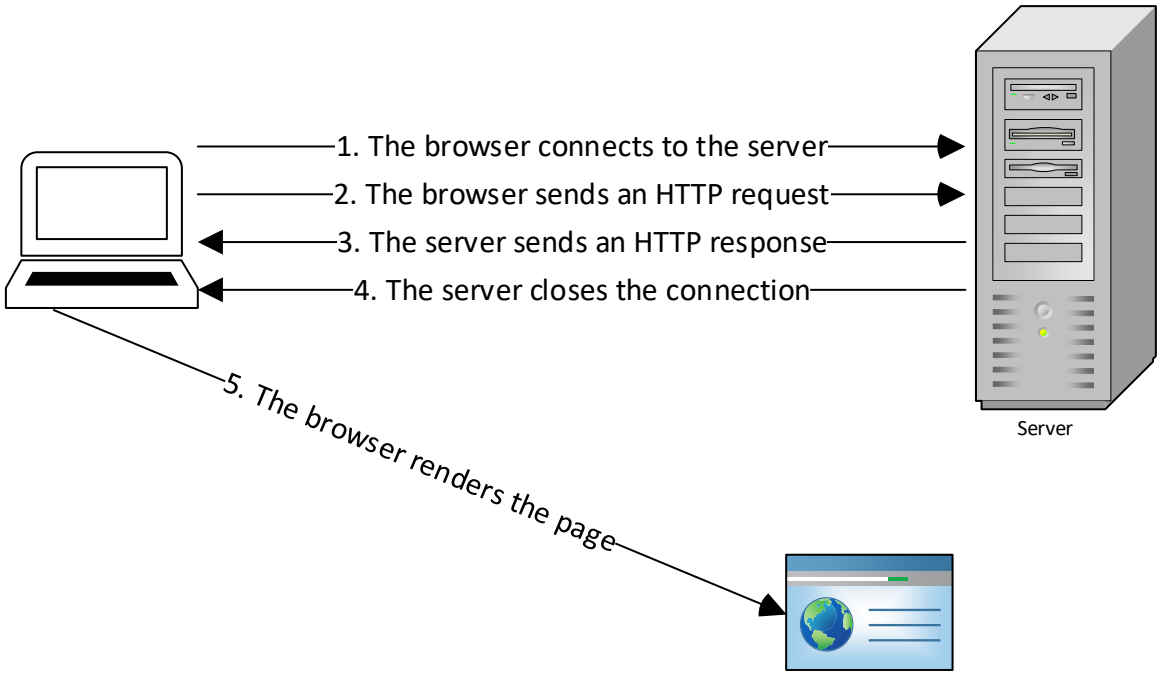
In theory, HTML can be used to mark up some data without ever using HTTP. We do that all the time when we open a web page from our local machine and just view it. On the other hand, we can use HTTP to transfer a word document, excel spreadsheet, pdf, picture, XML files, etc.,

without ever using an HTML document. However, typically HTTP protocol is used to bring down an HTML document from a server to the browser.

So, what is HTTP? HTTP is the protocol for text exchange on the web. It uses a TCP/IP protocol and socket connection behind the scenes. So, HTTP is HTTP/TCP/IP. In this paradigm, the client application (like a web browser) sends requests to the server (like a web server of an online toy shop) in plain text and receives information (such as a toy catalog) from the server in response in plain text.

An example of an HTTP conversion would be when a user enters the following on his browser: http://csis.pace.edu/~wolf/HTML/htmlnotepad.htm

The following events happen:



1. The browser opens a connection
2. The browser sends an HTTP Request
3. The server sends an HTTP Response
4. The server closes the connection

Let's assume we type the following on our browser:

http://www.example.com/index.html

The browser opens a connection

The browser opens a TCP/IP Socket connection with the server www.example.com. The browser automatically assumes that the server is listening to the default HTTP port 80. If the server is listening to a different port, you must specify with a colon after the domain name: http://www.example.com:9090/index.html (assuming the server is listening to port number 9090).

The browser sends an HTTP request

The browser (Client) converts the URL into an HTTP Request and sends it out to the server. An HTTP request message is composed of:

1) Request Line: describes HTTP Command

2) Header: browser and content information

3) Message Body: contents of the message

In this example:

1) Request Line would look like this:

HTTP method	Resource Name	Protocol/Version
GET	index.htm	HTTP/1.1

2) The browser will provide the Header.

3) The Message Body will be empty.

Here is the request line:

GET /index.html HTTP/1.1

Here are the request headers:

Host: www.example.com

Connection: keep-alive

Cache-Control: max-age=0

Upgrade-Insecure-Requests: 1

User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 11_2_1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/89.0.4389.82 Safari/537.36

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9

Accept-Encoding: gzip, deflate

Accept-Language: en-US,en;q=0.9

If-None-Match: "3147526947+ident+gzip"

If-Modified-Since: Thu, 17 Oct 2019 07:18:26 GMT

Since this is a GET request, the Request Body is missing. The Request Body is present in other types of Requests like POST. GET, POST, etc., HTTP commands are called HTTP methods.

Here are some of the HTTP methods that appear on the Request Line:

HTTP Methods:

GET: The GET method requests a page/URL from the server. The danger with the GET method is that any browser data sent is visible in the browser's URL. The GET method was initially designed to get content from the server, but nowadays, it is also used to post data.

POST: The POST method also requests a page/URL from the server. It is usually used to save data on the server. The beauty of the POST method is that browser data sent to the server is not reality visible. But the data is not secured either.

HEAD: The HEAD method asks for HTTP headers only; it is not interested in the response's body.

OPTIONS: Typically used to figure out what the server expects request headers, what content type is expected, and so on.

TRACE: Echo's back the user data, typically used for debugging. It can be a security concern.

PUT: The PUT method can also be used to replace the content on the server. This could be a security concern.

PATCH: Like PUT, PATCH partially replaces the content on the server. This is also a security concern.

DELETE: The DELETE method deletes the page/URL. Again, this is a security concern.

CONNECT: The CONNECT method establishes a tunnel to the server. This could be a security problem as well.

GET and POST are the two safe methods that web applications should allow. Among them, POST is preferred whenever possible. HEAD is ok too. Other methods should be disabled on the webserver.

Regardless, HTTPS should only be allowed to secure content.

The server sends an HTTP response

The server reads and interprets the request and sends an HTTP Response message back to the browser (Client). An HTTP Response message is composed of:

1) Response Line: Server protocol and status line

2) Header: Response Metadata

3) Message Body: Content of Message

Here is what the response line will look like:

HTTP/1.1 200 OK

Here are the response headers:

Content-Encoding: gzip

Accept-Ranges: bytes

Age: 196537

Cache-Control: max-age=604800

Content-Type: text/HTML; charset=UTF-8

Date: Fri, 12 Mar 2021 16:09:33 GMT

Etag: "3147526947"

Expires: Fri, 19 Mar 2021 16:09:33 GMT

Last-Modified: Thu, 17 Oct 2019 07:18:26 GMT

Server: ECS (nyb/1D07)

Vary: Accept-Encoding

X-Cache: HIT

Content-Length: 648

This particular site sends the response body in zipped format. However, if you unzip it, here is the response body:

```
<!doctype html>
<html>
<head>
  <title>Example Domain</title>
  <meta charset="utf-8" />
  <meta http-equiv="Content-type" content="text/html; charset=utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1" />
  <style type="text/css">
  body {
    background-color: #f0f0f2;
    margin: 0;
    padding: 0;
    font-family: -apple-system, system-ui, BlinkMacSystemFont, "Segoe UI", "Open Sans", "Helvetica Neue", Helvetica, Arial, sans-serif;

  }
  div {
    width: 600px;
    margin: 5em auto;
    padding: 2em;
    background-color: #fdfdff;
    border-radius: 0.5em;
    box-shadow: 2px 3px 7px 2px rgba(0,0,0,0.02);
```

```
    }
    a:link, a:visited {
        color: #38488f;
        text-decoration: none;
    }
    @media (max-width: 700px) {
        div {
            margin: 0 auto;
            width: auto;
        }
    }
    </style>
</head>
<body>
<div>
    <h1>Example Domain</h1>
    <p>This domain is for use in illustrative examples in documents. You may use this
    domain in literature without prior coordination or asking for permission.</p>
    <p><a href="https://www.iana.org/domains/example">More information...</a></p>
</div>
</body>
</html>
```

Here are the status codes used by the server on Response Line:

Status Codes:

Code Range	Category
100-199	Information
200-299	Successful (200 OK)
300-399	Redirection
400-499	Client Error (404 File Not found, a command status that we see all the time)
500-599	Server Error

Here are the Multipurpose Internet Mail Extension (MIME) content types used by the server for Header:

MIME Content Types:

text/html

text/plain

text/xml

application/octet-stream

…….

Server Closes the connection

As soon as the response is sent, the server closes the connection with the browser. Thus, for the following URL/page, the browser has to start the connection process from the beginning.

The HTML page that the server returns may have links to other HTML pages. When the user clicks on those links, the browser sends another brand new request to the server, the server sends another brand new response, and the browser displays the new page, and this request-response game goes on.

Fiddler Tool

There are quite a few free tools that allow you to monitor the data you are sending and the data you are receiving over your TCP/IP socket. In other words, these tools will enable you to see the raw request and the raw response. Fiddler is one of them. Here is how you get it to work.

Download the latest version Fiddler

Install Fiddler

Start Fiddler from your startup menu. You will have a Fiddler console available to you at this point.

Open your browser and type:

http://www.example.com

Fiddler automatically plugs into your browser like a proxy server and will record your requests and responses in the Fiddler console. If it does not, please read Fiddler documentation to manually put the proxy setting in your browser.

The right top section of Fidler is dedicated for the "HTTP request" and the right bottom for the "HTTP response".

Go to your Fidler and open the "Inspectors" tab of the Top-Right section. Next, open the "Raw" tab under the "Inspectors" tab, and you will see the raw request.

On the Bottom-Right section of Fidler, open the "Raw" tab to see the raw response.

Secured Socket Layer(SSL) and Transport Layer Security (TLS)

SSL is a security protocol for establishing encrypted links between a web server and a browser in online communication. Unfortunately, SSL is old technology, and companies have moved away from SSL and use TLS instead. There are several versions of TLS out there, but TLS 1.2 is recommended, and TLS 1.3 is also out. SSL or TLS does not matter, they both require certificates, which we will discuss shortly, but we need to know a bit of encryption and decryption techniques before that.

Symmetric vs. Asymmetric Cryptography

Let's first get our hand around cryptography – a way to encrypt your data to preserve privacy. Symmetric cryptography is something that we are naturally familiar with. You use a secret key and an algorithm to encrypt your data. The same key will be needed to decrypt the data. As wonderful as it may be, symmetric cryptography falls apart when a hacker has access to the secret key.

Can we do better? It turns out we can. We can use asymmetric cryptography. In asymmetric cryptography, the key you use to encrypt the message differs from the key you use to decrypt the message. How is that possible? Well, the two keys are mathematically related but not identical. For example, to encrypt a message, you need a number, let's say 2,035,153 (known as a public key), but to decrypt, you need two numbers, 1,009 and 2,017 (known as private keys) that are multiplied to get 2,035,153 as the public key. Ok, but can't I figure out the two private numbers if I have access to the public number? It is challenging and time-consuming to decode a long public key into two private keys since they are prime numbers.

In real life, public keys are typically thousands of digits (for example, 2048 bits), making it impossible to break down to two private keys in any reasonable amount of time.

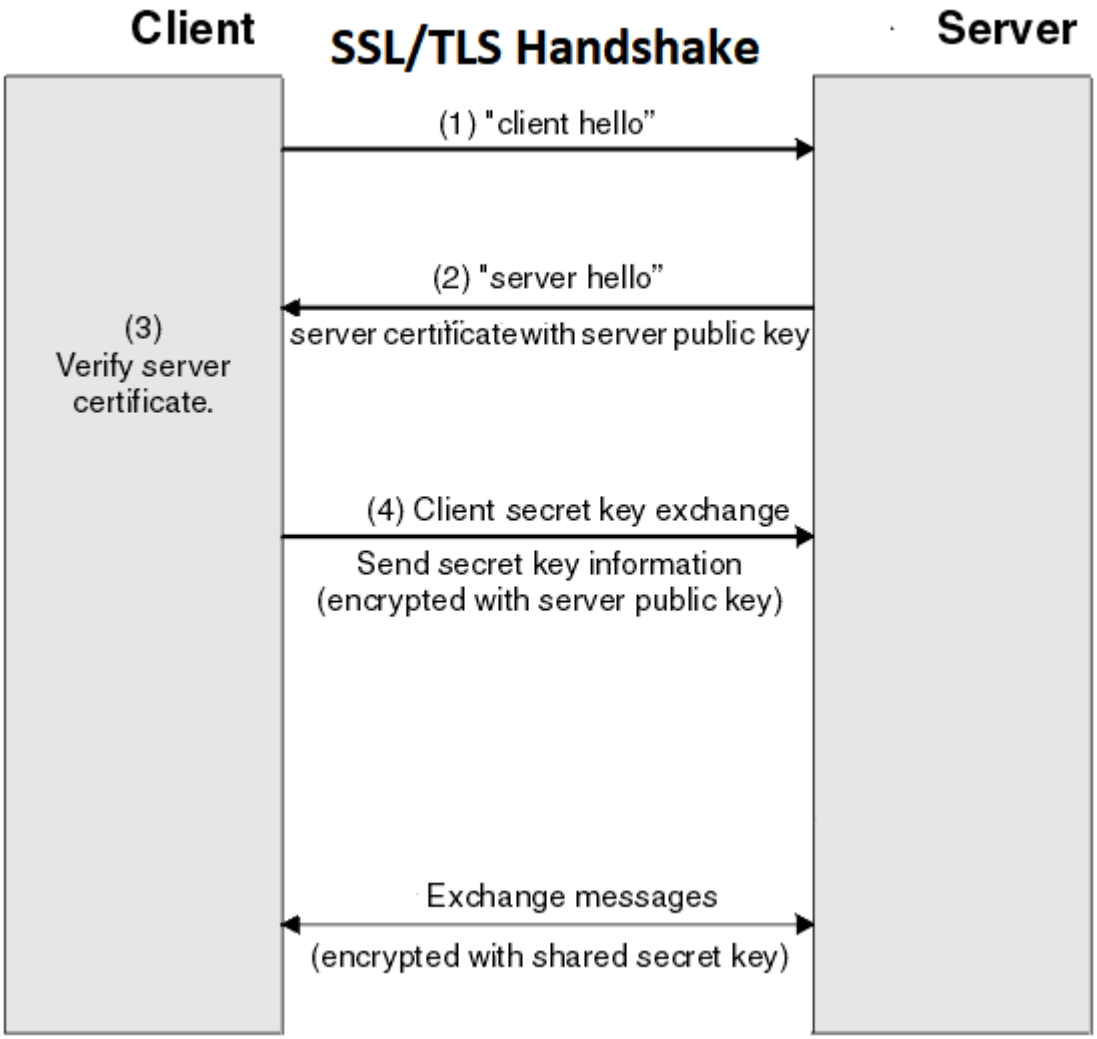
What is Digital Certification?

To rent a car, you need to present a valid driver's license. How does the rental company know that your driver's license is genuine, not a fake one? Well, they don't. So, they validate your driver's license with DMV.

Similarly, let's assume that you visited Google's website for a secured connection and receive a public key. How do you know that no "man in the middle" gives you its public key instead and pretends to be Google? How do you know that you received a public key from Google? Well, that's where digital certificates come into play. A digital certificate contains a public key, but your browser can verify a) The public key belongs to the party you expect it from b) A trustworthy certificate issuing authority issues the certificate.

How does SSL/TLS work?

In SSL/TLS, both asymmetric and symmetric encryption is used one after another. Sounds crazy? Why? Because communication with long asymmetric keys (let's say 2048 bits) is slow. So, it should be avoided as much as possible. On the other hand, a short symmetric key (256 bits) is speedy. That's why the initial handshake is done using long asymmetric keys, and then quickly, the communication turns to a symmetric encryption/decryption style. How? Well, let's see.



Your browser sends a hello message to the server. The server returns a hello message with a certificate that contains the long public key. The browser validates that the certificate is genuine and accepts it. It then generates a short random key (called a session key) used for symmetric cryptography going forward for the rest of the session. It then encrypts the key using the server's public key. The browser sends this encrypted session key back to the server. The server decrypts the message using its private key and retrieves the client's random session key from it. Now they both use symmetric cryptography for exchanging messages back and forth using the session key. So, SSL/TLS starts with asymmetric cryptography but ends up in symmetric cryptography. Of course, the symmetric key changes in the next session.

Demo: Go to https://www.nyu.edu

Click on the lock icon, click on certificate and go to the details page.

Who issued the certificate?

When does the certificate expire?

How long is the public key?

What DNS names are supported by the certificate? (subject and subject alternative names)

Now, go to https://www.mscdirect.com

Answer the same questions.

Introduction to Password Protection

Password Weaknesses

If you are not careful, your password can be cracked. Cybercriminals have stolen more than a billion online usernames and passwords, and most are sold on the darknet. But it goes beyond hacked passwords. In 2009, a survey showed that 18% of people use the same password for multiple sites. 44% uses short passwords (less than eight characters). 44% of users write their passwords down on a piece of paper or in a digital document. Here are some examples of commonly used passwords taken from the real world: 123456, password, 111111, welcome, ninja, freedom, baseball, superman, America, starwars, batman, spiderman, startrek.

Password Encryption (Hashing)

Even passwords that are not as common as shown above but weak can be cracked. So, how do they steal passwords, you may ask. Are they saved in plain text? Probably not. Are they encrypted? They can be, but as it turns out, encrypting a password is not the most protective measure since all you need to decrypt all the passwords is the key, and hackers are good at finding the key. That's why passwords are generally hashed and not encrypted. What's the difference? The difference between hashing and encryption is that encrypted data can be decrypted, but hashed values can never be reversed. Once hashed, you can never retrieve the original value from the hash. That sounds strange since we need to validate a password. Don't worry, that is no problem.

Let's assume you entered your password in an online site for the first time. The password is hashed and saved to the database. The next day you went back to the site and entered your password again. Now, it is time to validate your password. The site will 1) hash the password you just entered and 2) compare it with the saved hashed value. If they are identical, your password is correct, and you are in.

Let's do password hashing. Go to:

http://www.fileformat.info/tool/hash.htm

Type in "freedom" for String hash. After all, "freedom" is one of the most popular passwords. Let's take a strong hash like SHA-256 value:

13b1f7ec5beaefc781e43a3b344371cd49923a8a05edd71844b92f56f6a08d38

Looks cryptic, right? Let's now see how hackers crack this password.

Rainbow Table

"If the mountain won't come to Muhammad, then Muhammad must go to the mountain."

That's precisely what happens with hashed passwords.

If a hashed password can not be reversed, how does password hacking work? Sure, a hacker can hack your hashed password, but what good is a hashed password to the hacker? They still can't figure out the password. Wrong. Hackers use a Rainbow Table (some call it a Magic Table) to get around it. A Rainbow Table is a database of hashed passwords. If the hacker can look up your password's hashed value in this table, the hacker has the password in the next column.

Let's now go to the following site to find a rainbow table:

http://reverse-hash-lookup.online-domain-tools.com/

Copy/paste the hashed value and change the hash function to SHA-256 on the dropdown below it.

Click on Reverse, and you will see it will reverse the hash showing the following result:

Hash #1: freedom

Brute Force Cracking

Passwords can also be cracked using the brute force mechanism and without a rainbow table. I gave you a password-protected zip file that you need to break the password for. Install "John the Ripper on your machine." Copy "secret.zip" from the "demo3" folder to John's Run folder.

Run the following command:

zip2john secret.zip >hash.txt

you can open hash.txt now to review the hashed value of my password that I used to encrypt my file.

Now, run,

John hash.txt

In a minute, you should see the password in cleartext.

Use the password to open the zip file and read my secret text file inside the zip.

However, I have another zip file with a long password. John the ripper, tried for a month to crack it and eventually hung. No cracking was possible.

What do you learn from here? If you are going to protect your zip file or spreadsheet, use a long password so a brute force cracking program will take forever to crack it, or the rainbow tables will be needs to be infinitely long.

Better security options

So, how does a company protect its users from rainbow table attacks or brute force attacks? Well, few things can be done:

Secure the hashed file: The hashed file containing all passwords for all users need to be protected with utmost security.

Force a long password: The longer the password gets, the harder it gets to create a rainbow table since the permutation and combination of different letters and symbols grow exponentially as the password's length grows, so the brute force method takes longer and longer.

Add salt to password: We will discuss salt next, but it essentially a) adds length to the password b) provides randomness to the generated hash. Together it makes Rainbow table generation an impossible task.

Ask for two-factor authentication: Like the idea of salt, two-factor authentication can be used to enhance authentication security further. In two-factor authentication, you need a password and a code. After you enter the correct password, you are challenged with the code. The code is usually texted to your phone or emailed to you. The idea is you have the correct password and access to another device of yours to log in.

Salting Password

Rainbow tables only work because the same password is hashed to the same value. If two users have the same password, they'll have identical hashes. What if we could stop this? What if we add a random (and long) string to each password before hashing and save this string called "salt" with the user id and hashed value?

Let’s take our example of an easy password “freedom”. Let’s add a randomly generated salt to the password “Ut45hjs1dds”. Together we get “freedomUt45hjs1dds”. SHA-256 hash value of this password is: ab3ea23056f8f3c71c8e43b68d88d4cea83aab75df38a0ff79280799a804ae05

If you try to look up this in any rainbow table, you will not find the reverse value because it is a hash of a very long string. More importantly, let's assume that another user uses the same password, "freedom." This time, the randomly generated salt would be different. Let's say it is "H78yrt2allau". Together we get "freedomH78yrt2allau". SHA-256 hash value of this password is: db4658237125ec28b78071e1384b9e3d7c982e431ef44e4666814c143262df1e. This hash value is significantly different from the previous hash, although the password is the same. Anybody who tries to create a rainbow table now must create a table with all possible passwords and all possible salts. Together, the task becomes impossible since the possibilities are infinite.

So, saving the salt and hashed password increases the complexity of the password.

Long Password

Almost all sites force you to use special characters and at least eight characters for a password, but a longer password is much more complex to crack than a short cryptic password. For example, "!2Hjrdwu" may take minutes to crack, but "IloveadozencrispycreamDonuts" will take months. Is that harder to remember than the cryptic one? No, yet much more robust. If salt is added to this password, the probability of cracking this password reaches zero.

Use Different Passwords for Different Sites

The first problem is that many sites do not use a salted password. The second problem is that sites are often hacked and hacked user id and hashed password is sold on the black market like

the darknet. If you use the same password for multiple sites and one of the sites is hacked, your password is now exposed to many other sites. Therefore, it is a sage idea that your password is different for different sites.

Use Password Manager

Using different passwords for different sites and long passwords makes sense, but managing all these passwords could quickly become a nightmare. The best option is to use an online vault, also known as a "password manager" like Lastpass.com or onepassword.com, and save all your passwords there.

Now, is saving all your passwords in one site safe? Isn't that equivalent to putting all your eggs in one basket? Yes, you are right. However, the alternative is even more dangerous. So, what are you going to do? Write down all passwords in a file? On a piece of paper?

There is some good news about password managers. Password managers like LastPass and Sticky Password use zero-knowledge security protocols that encrypt users' master passwords with an encryption key stored only on users' devices (so that the companies have 'zero knowledge' of users' passwords). This encryption includes thousands of rounds of authentication hashing. An algorithm converts a string of text into a longer string, making it more difficult for hackers to crack the hashed text.

Demo: Lastpass

Use Two Factor Authentication for Password Manager

Fortunately, there is a way to secure your password manager even more. Use two-factor authentication for the password manager. This way, even if someone cracks your password for the password manager (which is hard to do), they still need access to your phone or email to get into your account.

Demo: Lastpass demo

Watch Kevin Mitnick cracking password:

https://www.youtube.com/watch?v=K-96JmC2AkE

Homework 3

1. Like HTTP, there is a protocol to transfer files from one computer to another. It is called FTP. FTP is just as insecure as HTTP and should be avoided at all costs. What should you use instead? (hint: there are at least three options, FTPS, SFTP, and AS2. Look them up and try to describe them).
2. What is the difference between hashing and encryption?
3. Who is DigiNotar? What was the issue with DigiNotar? How did the browsers deal with the problem?
4. A company decided to salt passwords the following way:

   Take the user input and hash it.

   Concatenate the hashed value with the password as salt.

   And then hash it again and save the ultimate hashed value.

Although they are salting each password with a different salt, and they never have to save the salt value, it is still not the best way to do it.

What is the problem with salting this way?

(Hint: How many rainbow tables will be needed to crack all the passwords of all users if the salting mechanism is known to the user and they gain access to the hashed passwords).