<u>**Assignment 2**</u>

1) What is purpose of:
   a) On a Linux system, the /etc/passwd file is used to store user account information such as username, user ID, group ID, the name of the user, the home directory, and the preferred shell. It is called the passwd file because originally the encrypted user password was stored in the file as well. However, security concerns were raised and the password information now resides in the shadow file /etc/shadow.
   b) In modern Linux systems, the /etc/shadow file is used to store all password information for a user account. This information includes the username, the encrypted password, the number of days since the password has been changed, the number of days before the password can be changed, the number of days after which the user is required to change their password, and other password related information. Even though the one-way salted hash would be very difficult to crack, the /etc/shadow file is read/write protected for additional security. Only the root user is capable of reading or writing to the shadow file.
   c) In Linux, setuid (or SUID) is an abbreviated form of Set User ID. Normally, if a program were to be executed, the program would inherit the permissions from the executing user. If your user can only access files in your home directory, the program could only access files in your home directory as well. SUID, however, is a special type of file permissions which allows a program to inherit permissions from the executing user as well as the owner. Therefore, if you execute a program that is owned by root, the program can access the files in your home directory (your permissions) as well as any file that root can access (root's permissions). The SUID attribute is important for files such as the /etc/shadow because it allows non-root users to run tools such as passwd which lets the user change their password.
   d) The chroot command changes the root directory of a process and all its children. If the root directory has been changed, the process is unable to access any files that exist outside of the new root directory and its subdirectories. For example, if you used chroot and changed the root directory for Process X, Process X can *only* access files from your home directory and subdirectories.
2) The ls command with the –l switch displays the long listing of information about a provided file or directory. The information includes native file system attributes such as the file type (link or directory), the owner, group, and user permissions, last modification timestamp, etc.. The lsattr command, however, displays information about a given file's extended file system (e.g. ext2, ext3) attributes. Examples of extended attributes include append only, immutable, secure deletion, and undeletable.
3) In the Android system, applications must request permission from the user to access personal information and device functionality such as camera and GPS. The problem, however, is that permission requests are very broad. For example, an application may only need to read the user's contact list, but the permission request will give the application

permission to read, create, and delete contacts. If the application need only read contacts, the application should only be able to read contacts and nothing more. The challenge, however, is making the permission request process fluid while preserving the least privilege principle. It would be tedious and cumbersome to require the user to respond to three prompts (“Can Application X read contacts?”; “Can Application X create contacts?”; “Can Application X delete contacts?”) when an application wants to access contacts. Granular permissions allow a dedicated user complete control over a system, but a casual user may become frustrated by the tedium.

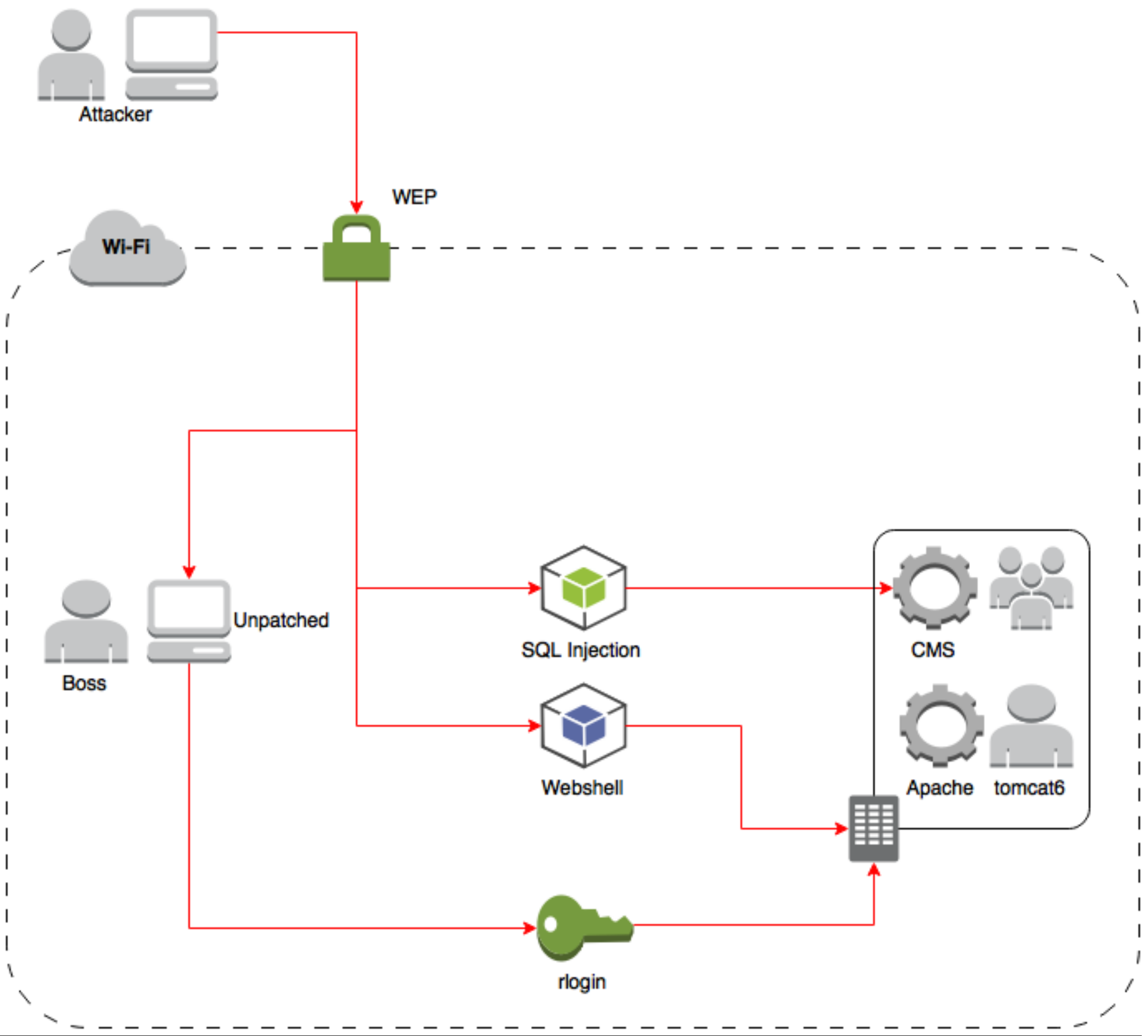
4) The standard UNIX permission model allocates read, write, and execute permissions to a file for the file owner, the owning group, and everyone else. This model works on a very basic level, but the model does not scale very well into, for example, an enterprise network of UNIX systems. The scaling issues can be resolved with Access Control Lists. An access control list allows permissions to be allocated to sets of users and sets of user groups. This is contrary to the standard model which only allows permissions to be set for a single group. Furthermore, suppose there is a directory that needs to be accessed by Quality Assurance agent and a Development team but no one else. The standard UNIX model does not allow for this type of permission allocation. An access control list, however, does by granting access to both the Development group and the Quality Assurance group.
5) When executing a program, the Real User ID (RUID) is set to the user ID of the user executing the program. In the case of a SUID, however, the Effective User ID (EUID) is the user ID of the SUID user ID. For example, in the case of the passwd tool, the RUID is your user ID, but the EUID is root because passwd SUID attribute is root.
6) Similar to any other illegal trespassing, an attacker who has compromised a system needs to ensure they cannot be detected nor traced. The type of attack wildly changes the types of cleaning an attacker needs to perform. One common method is to sanitize the system logs. For example, suppose that an attacker compromised the remaining user account of a terminated employee and used the credentials to log into the system. The login will be recorded in the /var/log/auth.log. An analyst that notices a terminated employee account in the user log would be a huge red flag that an unauthorized person gained access to the system. Another important log is the bash_history log file stored in the user home directory. The bash_history file stores a record of every single bash command executed by the user. An analyst could easily see what an attacker had done by looking at the log. In order for the attacker to edit the logs, the attacker will require root permissions. Secondly, an attacker must be sure to hide or remove all files that were created. Temporary executables should be removed securely to further reduce chances of recovery and detection. If the attacker needs for a file or executable to stick around, the attacker needs to hide and protect that file. Simple ways to hide a file include placing it in an obscure directory and prepending the name with a dot which signals the operating system to hide the file from normal listing commands. In the event that the file was found, it could be desirable to prevent a file from being removed. By setting the extended attribute immutable, the file is given an extra layer

of removal protection. The permissions for hiding and deleting files range from basic user permissions to root permissions depending on system directories.

7) <u>IPMI Vulnerabilities:</u>
    a) The author analyzes the security of Intelligent Platform Management Interface (IPMI) frequently used by IT professionals. The author finds that IPMI is very unsecure and enumerates a set of exploitable vulnerabilities including: insecure input validation, shell injection, and buffer overflows. The author found that input is not sanitized properly in many areas which allows an attacker to achieve shell injection. Specific shell injection attacks resulted in the attacker gaining unauthorized root access. Finally, the simple buffer overflow in the standard login page allowed a crafty attacker to remotely execute instructions.
    b) The author provides relatively basic advice to both IPMI users and IPMI developers. To the users, the author states that IPMI should only be turned on if absolutely necessary. If IPMI is necessary, the author suggests that the machine should be isolated to a virtual management network and not exposed to the Internet. Finally, the author urges users to keep the IPMI firmware up-to-date. To the IPMI developers, the author calls for security-driven development. Companies providing tens of thousands of machines to enterprise customers should not allow simple buffer overflow bugs pass through unnoticed. The author calls these vulnerabilities "textbook." Furthermore, the author notes that some IPMI machines could be more secured simply by having secure defaults such as requiring a user to manually enabled IPMI.
    c) IPMI Vulnerabilities:
        i) <u>Insecure Input Validation</u>
            (1) Improper size checking of fields such as username and password resulting in buffer overflow
            (2) Lack of input sanitization allowing shell code injection
            (3) User permission management on the client-side allowing a user to escalate privileges
        ii) <u>Shell Injection</u>
            (1) Insecure input validation allows for execution of arbitrary shell code
            (2) Other pre-installed shell commands allow for code injection. The author uses an openssl utility and wget to execute code from another server.
        iii) <u>Buffer Overflow</u>
            (1) Use of functions such as strcpy without length validation
            (2) Lack of buffer overflow defenses
            (3) Able to execute system commands with escalated privileges
            (4) Stack defenses has a search space of 4096 possibilities and can be easily brute forced

8) <u>Web Server Breach</u>
    a) To begin, the attacker was able to breach the public wireless network by leveraging the insecurities of the WEP security protocol. WEP is widely known to be very insecure.

Once the attacker had access to the network, he was able to use SQL Injection to access the administration console for the content management system. Through the administration console, the attacker successfully uploaded a PHP program (bkdoor.php) which implemented a web-shell for remote code execution on the web server. The interesting piece of the attack is the level of permission that the commands executed by bkdoor.php held. The level of permissions offered by the user tomcat6 – responsible for the webserver process – only passwd and hosts.equiv would have been readable. The remaining files would require root access to read. The problem, though, was the hosts.equiv contained an IP address meaning that the machine assigned to that IP address could log into server machine without supplying credentials. The attacker was likely able to isolate the machine assigned the given IP address and found that it was very insecure due to lack of security patches. The attacker exploited some vulnerabilities and was able to gain root access on the unpatched machine. After executing the rlogin command, the server machine gave the unpatched machine – and the attacker – access without supplying any credentials, thanks to IP address in the hosts.equiv file. Because the unpatched machine actually belonged to the boss, the boss likely had root access on the server machine. Therefore, the attacker had root access to the server machine. One interesting piece of the puzzle that has not been completely accounted for is the fact that in the passwd file, the tomcat6 user, who should be responsible for executing commands from the web-shell, has the shell set to /bin/false. Setting a user's shell to /bin/false is a standard method of denying the user access to an interactive shell. Without an interactive shell, the user should not have been able to execute commands.



b)

c) For each point in the attack diagram, there are simple pieces of advice that grant better security. To begin, the company wireless network needs to use a better security protocol than WEP. WEP is widely known to be extremely insecure and is all but defunct. The company should upgrade to, at least, WPA2 (or WPA2-Enterprise) which is the de-facto standard for modern wireless networks. Next, the content management system needs to employ input sanitation to prevent SQL Injection. Input sanitation should be implemented across the entire system to avoid SQL Injection in other parts of the system. Without SQL Injection, the attacker would not have been able to upload the web-shell. Next, the boss' computer needs to be patched immediately. On top of the patches, the system administrator should implement security protocols that would be implemented on any other company machine. Lastly, the system administrator should rethink the use of the hosts file to implicitly trust the boss' IP Address. Instead, the boss should be required to SSH into the server with credentials.