Assignment 1: Bugs in OpenSSH

Question 1

CVE-2016-0778:

Description: When proxy and forward options are enabled, the roaming_read and roaming_write function in roaming_common.c may assign a high value to connection_in or connection_out (both are file descriptors), but the variable -- setp which used to store the file descriptor is never enlarged. So the following memset() and FD_SET() functions in packet_read_seqnr() and packet_write_wait() function may overflow setp, cause errors when free(setp) is called. A malicious SSH server can request hundreds of forwardings to increase the file descriptors and overflow setp. It will cause data corruption and denial of service.

Affective version: OpenSSH 5.x, 6.x and 7.x before 7.1p2.

Remedial action: adding the undocumented option "UseRoaming no" to the system-wide configuration file (usually /etc/ssh/ssh_config), or per-user configuration file (~/.ssh/config), or command-line (-o "UseRoaming no").

CVE-2016-1907:

Description: The ssh-packet_read_poll2 function in packet.c called sshpkt_disconnect() which sends disconnect message, but does not terminate the execution, program will continue while the data which will be used by rest of this function has been cleared by buffer_clear(), thus cause out-of bounds read and application crash. A remote attack can use a crafted network traffic to trick this error and cause a denial of service.

Affective version: OpenSSH 6.8, 6.8:p1, 6.9, 6.9:p1, 7.0, 7.0:p1, 7:1, 7.1:p1

Remedial action: Update software higher or equal to 7.1:p2.

Question 2

The common weakness in both bugs is that both of the bugs will cause a Denial of Service. So attackers can make use of these bugs to break the Availability of the OpenSSH.

Question 3

Score:

Vulnerabilities are labeled "Low" severity if they have a CVSS base score of 0.0-3.9. Vulnerabilities will be labeled "Medium" severity if they have a base CVSS score of 4.0-6.9.

Vulnerabilities will be labeled "High" severity if they have a CVSS base score of 7.0-8.9.
Vulnerabilities will be labeled "Critical" severity if they have a CVSS base score of 9.0-10.0.

So CVE-2016-0778 is labelled with “High” and CVE-2016-1907 is labelled with “Medium”. These scores are computed rely on the Basic Metirc Vector given to each bugs and using the CVSS v3 Equation.

Vector:

1. Exploitability Metrics: Exploitability metrics reflect the characteristics of the thing that is vulnerable, which we refer to formally as the vulnerable component.
    a. Attack Vector: This metric reflects the context by which vulnerability exploitation is possible. This metric value (and consequently the Base score) will be larger the more remote (logically, and physically) an attacker can be in order to exploit the vulnerable component.
        i. Network(N): A vulnerability exploitable with network access means the vulnerable component is bound to the network stack and the attacker's path is through OSI layer 3 (the network layer). Such a vulnerability is often termed "remotely exploitable" and can be thought of as an attack being exploitable one or more network hops away (e.g. across layer 3 boundaries from routers).
    b. Attack Complexity: This metric describes the conditions beyond the attacker's control that must exist in order to exploit the vulnerability.
        i. Low(L): Specialized access conditions or extenuating circumstances do not exist. An attacker can expect repeatable success against the vulnerable component.
        ii. High(H): A successful attack depends on conditions beyond the attacker's control. That is, a successful attack cannot be accomplished at will, but requires the attacker to invest in some measurable amount of effort in preparation or execution against the vulnerable component before a successful attack can be expected.
    c. Privileges Required(PR): This metric describes the level of privileges an attacker must possess before successfully exploiting the vulnerability.
        i. None(N): The attacker is unauthorized prior to attack, and therefore does not require any access to settings or files to carry out an attack.
    d. User Interaction(UI): This metric captures the requirement for a user, other than the attacker, to participate in the successful compromise of the vulnerable component. This metric determines whether the vulnerability can be exploited solely at the will of the attacker, or whether a separate user (or user-initiated process) must participate in some manner.
        i. None(N): The vulnerable system can be exploited without interaction from any user.
2. Scope (S):ability for a vulnerability in one software component to impact resources beyond its means

    a. Unchanged(U): An exploited vulnerability can only affect resources managed by the same authority. In this case the vulnerable component and the impacted component are the same.
3. Impact Metrics: The Impact metrics refer to the properties of the impacted component. Whether a successfully exploited vulnerability affects one or more components, the impact metrics are scored according to the component that suffers the worst outcome that is most directly and predictably associated with a successful attack.
    a. Confidentiality Impact(C): This metric measures the impact to the confidentiality of the information resources managed by a software component due to a successfully exploited vulnerability.
        i. High(H): There is total loss of confidentiality, resulting in all resources within the impacted component being divulged to the attacker. Alternatively, access to only some restricted information is obtained, but the disclosed information presents a direct, serious impact.
        ii. None (N): There is no loss of confidentiality within the impacted component.
    b. Integrity Impact: This metric measures the impact to integrity of a successfully exploited vulnerability.
        i. High(H): There is a total loss of integrity, or a complete loss of protection.
        ii. None(N): There is no loss of integrity within the impacted component.
    c. Availability Impact(A): This metric measures the impact to the availability of the impacted component resulting from a successfully exploited vulnerability.
        i. High(H): There is total loss of availability, resulting in the attacker being able to fully deny access to resources in the impacted component; this loss is either sustained (while the attacker continues to deliver the attack) or persistent (the condition persists even after the attack has completed). Alternatively, the attacker has the ability to deny some availability, but the loss of availability presents a direct, serious consequence to the impacted component (e.g., the attacker cannot disrupt existing connections, but can prevent new connections; the attacker can repeatedly exploit a vulnerability that, in each instance of a successful attack, leaks a only small amount of memory, but after repeated exploitation causes a service to become completely unavailable).
        ii. None(N): There is no impact to availability within the impacted component.

How servere is each CVE?

CVE-2016-0778: This bug is given a very high score (8.1). So this bug is very severe, it can be spreaded over the internet. Most important, this bug could lead to a total loss of Confidentiality, Integrity and Availability. With a stack overflow attack, computer may be fully controlled by attackers by using malicious shellcode.

CVE-2016-1907: This bug is given 5.3, which means this bug is labelled “Medium”. This bug will not lead to the any loss of Confidentiality and Integrity. A little of Availability may be corrupted, but user may reconnect OpenSSH after restart the software.

Question 4

The CVSS v3 score and vector changes from 9.8 with AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H to 8.1 with AV:N/AC:H/PR:N/UI:N/S:U/C:H/I:H/A:H. As we can see there is only one change in Attack Complexity vector (from Low to High), so basically means that a successful attack depends on conditions beyond the attacker's control. In this case, two non-default options: a ProxyCommand, and either ForwardAgent (-A) or ForwardX11 (-X) is needed to perform an attack, this is out of attackers` control. What’s more, performing a successful attack the attacker have to do a file descriptor leak first to improve exploit reliability.

Question 5

Use a top N bugs list activity in Code Review practice may help them discovered these bugs. The SSG maintains a list of the most important kinds of bugs that it wants to eliminate from the organisation’s code and uses it to drive change. This list helps them to turn their focus on the bugs which are important. Beside of that, developers can quickly found the possible vulnerable code location. OWASP top 10 list is a good example, developers can look up the list and check whether there is a vulnerability in their program. It is faster than simply review all the code.

Build attack patterns and abuse cases tied to potential attackers activity in Attack Model practice may help them discovered these bugs. The SSG prepares for security testing and architecture analysis by building attack patterns and abuse cases tied to potential attackers. They can apply these security testings and architectures to test their program to find the vulnerabilities easier and faster. Beside of that, these attack patterns and abuse cases could reuse in new applications or new version.

Use external penetration testers to find problems activity in Penetration Testing practice may help them discovered these bugs. External penetration testers bring a new set of eyes to the problem. Developers somethings do not understand security in-the-wild, so a professional penetration tester may help them to handle incidents and find real problems because he considers a program in final environment.

Question 6

I think we could not say which software is better just comparing the number of CVEs these two SSH server had. These 80 CVEs reports also indicate that OpenSSH is widely used, well tested and most vulnerabilities have been fixed since 1999. Even if I search for another popular SSH Client, I still get more than 50 CVE reports. SSH server which has only 10 CVEs maybe so good with so fewer errors (nearly impossible) or could be so bad that they did not fix there 10 problems and did not update software. Most importantly, we should focus

on the how critical of these CVEs, not just numbers of reports. Also, we can track the patch log of both SSH server, OpenSSH has been published since 1999 and kept updating. But this is hard to say, less popular SSH server may not have so many attacks but they are very serious attacks. For me, I may still choose OpenSSH since it is widely used, well tested and bugs are fixed faster.
In term of pros and cons of changing SSH server. I think the advantages could be that you can use some new features such as UI or more configuration settings. But we have to face the fact that the more popular this SSH server is, the more we may be attacked.

Assignment 2: Memory Corruption

Question 1 :

Vulnerability: We can input 29 characters input the program, but the variable password is defined to be 16 characters. Above the password variable, is the hash code we want to compare with. So meaning, we can overflow at most 13 characters into the hash code, and what's more, during to the fact that scanf will add a 0x00 into as the end of input, so the 14th place of hash code will be replaced by 0x00. So, we can find a fake password, whose hash code is ending with 0x00 0xda 0xac. Another point is, if we input 0x00, scanf will end input, and the MD5 function will also only encrypt the fake password, not the whole input, because we use strlen() function, which will only return the length of fake password. So we can build our input in two part, the first part is a fake password whose hash code ends will 0x00 0xda 0xac, then we add enough 0x00 after the fake password to fill it to 16 characters, so that we can add the hash to the right location. Second is the hash code of the fake password to overwrite the old hash code. So we can get true in the memcmp function and get “Correct password!”.

Step:

1. we need to find a fake password, whose hash code ends with 0x00 0xda 0xac.
2. get the hash code of the fake password.
3. build exploit string with 3 step.
    a. put fake password at first
    b. add padding with 0x00 to fill the string to 16 characters
    c. followed with the hash code of fake password
4. output the exploit string to a file, eg. exploit.out
5. use pipe operation to exploit the program

Question 4

Vulnerability 1:
strncpy() function will not add an NUL character into the destination in some case. For example, if we input more than 64 characters in the console, strncpy() function copies the first 64 character, but without adding an NUL character at the end. This will cause a memory leak when program call strcat() or printf(), these function will read until seeing NUL

character, beyonding the boundary of original content. When the *username* is copying to *display*, strcpy are expecting an NUL character to end the copying, but since there is no NUL character in the username, strcat will continue to read until encountering an NUL character, which in the case, the password is copied to the first part of the *display*. Fix is simple, we can add an NUL character to the *username* and *password* manually.

Vulnerability 2:
strcat() function does not check the boundary when combining strings. As we talked about in the first vulnerability, due to the vulnerability of strncpy(), strcat() will copy a string beyond username till encountering the NUL character. So if we input a long username and password, the first part string in the display will be 128 characters (because it contains both username and password). Then since strcat() does not check the boundary, it will continue to combine password into the display, which is defined as 132 bytes long, rest of password characters will overflow the memory, resulting in a buffer overflow. A buffer overflow may cause heap corruption, stack corruption, program exit. An attacker may also be able to control computer with crafted code. We can fix this by using strncat() to make sure strncat() do not connect string into destination beyonding the boundary.

Assignment 3: Web Security

Question 1:

1. First, attacker can login into his account and remember other user`s name. For example, david log into the main page and remember a username -- “joseph”.
2. Second, attacker can input “joseph’ -- ” as the username and a random password.
3. Right now, attacker has logged into system with username “joseph”, he can vote for his own picture(david).

Question 2:

From the view of attacker:

1. First, attacker log into system using his account. For example, attacker`s username is “lyc”.
2. Second, attacker submits/updates his image link to “http://localhost:8080/vote.php?vote=lyc”, this link will send a request to the server to cast a vote for “lyc”.

From the view of victim:

1. victim log into system using his account.
2. The main page uses the URLs to read the locations of images and display them. But since the image link of “lyc” is a crafted link which sends a request to server(Using victim`s cookie), so it will automatically cast a vote for “lyc” in the user of victim transparently.

Question 3:

Vulnerability 1:

Using non-constant string to build prepared statement query in login() function in functions.php. A SQL injection happens when an attacker uses crafted input so that system will regard the crafted input as part of the SQL instruction and execute it. An attacker can use '--' to avoid password check, use ';' to inject a new SQL break the integrity and confidentiality. That means an attacker can log into others accounts, change database data and even drop tables.

We can use a constant string to build prepared statement, then bind parameter to query. So that the username and password will not be regarded as parts of the SQL instruction. So that SQL injection can be avoided.

Vulnerability 2:

Using GET to send a vote request to update vote count and database. An attacker can perform a CSRF attack by injecting a crafted link or scripts into a vulnerable page. When victim user logs into the system and triggers the malicious link or script, the browser will send a request from the malicious link or script, along with victim`s cookie. So an attacker can perform a request in the name of the victim.

The fix can be made by using POST method to send the vote request.

Vulnerability 3:

add_user() and signup() function in functions.php directly output username to the web page directly. This is vulnerable when attacker input a crafted string as username. For example, attacker can input </p><script>alert('hello')</script> as the username, it will trigger JavaScripts in next page. So an attacker can use this vulnerability to execute JavaScripts to get information such as cookies.

The fix can be made by using htmlspecialchars() function to wrap the $username. So that the the dangerous characters such as '<' '>' will be transform into < and >. Although whey will be seen the same in the web page, but this transformation will defend the crafted input to change the DOM elements in this HTML.

Vulnerability 4:

System store user`s password in plaintext, source can be find in the add_user() function in functions.php. So if the information of database leaks, an attacker can get the password of every user in this system.

The fix can be made by storing a hash of password using md5 into the database, but plaintext. When a user wants to sign up, the password they inputted will be transformed into

a MD5 hash code and stores into database. When they logged in, the password they inputted will be hashed into MD5 first, then compare to database. So even when the database is leaked, the attacker still cannot be get user’s password directly.