

Sri Lanka Institute of Information Technology

GHOST VULNERABILITY AND HOW TO EXPLOIT IT

CVE -2015-0235

## SYSTEMS AND NETWORK PROGRAMMING

Individual Assignment

Submitted by:

## P M S T Rathnayake

IT19163314

CONTENT

1.Abstract

2.Introduction to THE GHOST Vulnerability

3.what is GHOST Vulnerability

4.How this GHOST Vulnerability Works

5.Impact from GHOST vulnerability

6.Examples of how to se the codes to check GHOST vulnerability

7.Process I have done to exploit GHOST vulnerability

8.Refferences used to gather information

Abstract

In this report submission, it describes the GHOST vulnerability. It is called as GHOST vulnerability because it can be triggered by calling the gethostbyname() function.

The report consists the details about how GHOST vulnerability is significant today. It is rated as critical vulnerability by linux distributors.

You can study the function gethostbyname() under glibc.

It is gotten to know that any glibc version 2.02 through 2.17 are vulnerable. And, version 2.18 to 2.20 are not vulnerable.

Some of the targeted or possibility to be affected OS platforms are below.

Ubuntu ubuntu Linux x 12.04 LTS i386

Ubuntu ubuntu Linux x 10.04 ARM

Red hat enterprises Linux Desktop 5 client

GNU glibc 2.2

Debian Linux 6.0 sparc.

And more…..

This GHOST vulnerability was exploited by sending a specially crafted email to get a remote shell to the Linux machine.

In this report submission, you can find the details of how to test the GHOST vulnerability. I have depicted some of codes which we can use. Since we are still in 2nd year., so, we used scripts which was published on web.

In my case, I have chosen centos Linux OS platform to test and Exploit GHOST vulnerability.

I have run **sudo**\* commands which will update the server with any available patches.

Introduction to THE GHOST VULNERABILITY

A cloud security firm Qualys’ did a research that, stumbled across a bug which could leave millions of LINUX-based workstation vulnerability to attacks. This Qualys’ research discovered that, this bug is working closely with LINUX operating system, the glibc library is an implementation of the standard C library and the core part.

So, the glibc library is affected by GHOST vulnerability is one of the common pieces of software found on LINUX workstation. GNU library consists with responsibilities like , implementing variety of software task by different applications or platforms. The GHOST vulnerability is used to resolve hostnames to IP addresses from the computers which are connected to the internet. The main risk is the GHOST vulnerability causes difficulties for the organizations to stop at the firewalls. Therefore, people say that this is much similar to Heartbleed vulnerability.

Another dangerous thing about GHOST vulnerability is, their flaw leaves internet-connected computers vulnerable to a potentially allowing hackers to run malicious code on system.

What is GHOST Vulnerability?

To mitigate the buffer overflow vulnerability like GHOST is to identify vulnerable systems, deploy patches or prioritizing the remediation process. In glibc there is a buffer overflow which was discovered in the function \_nss\_hostname\_digits\_dots(). Both locally and remotely the bugs can be reachable via function called gethostbyname() .Before a bad actor is answering, it is good to keep the current inventory of devices, operating system and application in network. This kind of situations will remind you to keep vendors who are active in security community.

As mentioned in earlier, \_nss\_hostname\_digits\_dots() was found in the heap\_based\_buffer overflow. This is used by the glibc function call gethostbyname() and gethostbyname2(). Any remote attacker can execute an arbitrary code running an affected application with the permission of the users.

In this vulnerability, byte executions are limited. For 32 bit computer an attacker can execute limited up to 4 bytes. And, for 64 bit computer an attacker can execute limitation as 8 bytes.

How this GOST Vulnerability Works?

The function called “gethostbyname()”, is accessible by all programs. We must declare variables, to write a function by extension a program.

Let us take an example for this,

For the gethostbyname() function, one declares FQDN variable and states that is must does not exceed a determined number of characters.

The program looks matching information in the corresponding RAM part in the computer with respected to the condition. So, in here the IP address linked to the domain name.

The program falls, and requested will seek to use some resources in the machine if there was nothing planned to the block the requests. So, one exceeds the variable capacity and touch memory space that is not dedicated to the variable.

Many gethostby…() functions , there is having a risk of characters’ size misuse, coding mistakes or lack of security in the coding.

Impact from GHOST Vulnerability

gethostbyname() and gethostbyname2() function calls in the glibc library can affected by GHOST buffer overflow bug. These are used to perform DNS resolution, which is a common and most frequently used task. So this vulnerability will cause the attacker to control the system. To exploit this vulnerability, attacker should trigger a buffer overflow by supplying an invalid hostname argument to an application that performs DNS resolution.

GHOST vulnerability was not only remotely exploitable across many different platforms, but it was remotely exploitable with virtually the same payload. Each of this service are vulnerable to GHOST has very likely require unique payload.

Will take a sample example and see how to use the code to check the vulnerability

#include <netdb.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <errno.h>

#define CANARY "in\_the\_coal\_mine"

struct {

char buffer[1024];

char canary[sizeof(CANARY)];

} temp = { "buffer", CANARY };

int main(void) {

struct hostent resbuf;

struct hostent \*result;

int herrno;

int retval;

/\*\*\* strlen (name) = size\_needed - sizeof (\*host\_addr) - sizeof (\*h\_addr\_ptrs) - 1; \*\*\*/

size\_t len = sizeof(temp.buffer) - 16\*sizeof(unsigned char) - 2\*sizeof(char \*) - 1;

char name[sizeof(temp.buffer)];

memset(name, '0', len);

name[len] = '\0';

retval = gethostbyname\_r(name, &resbuf, temp.buffer, sizeof(temp.buffer), &result, &herrno);

if (strcmp(temp.canary, CANARY) != 0) {

puts("vulnerable");

exit(EXIT\_SUCCESS);

}

if (retval == ERANGE) {

puts("not vulnerable");

exit(EXIT\_SUCCESS);

}

puts("should not happen");

exit(EXIT\_FAILURE);

}

Will use another code example to check whether the program is safe.

In below example getaddrinfo() calls gethostbyname2\_r() on if first call to inet\_aton() requirement, the internal calls be safe.

at->family = AF\_UNSPEC;

...

if (\_\_inet\_aton (name, (struct in\_addr \*) at->addr) != 0)

{

if (req->ai\_family == AF\_UNSPEC || req->ai\_family == AF\_INET)

at->family = AF\_INET;

else if (req->ai\_family == AF\_INET6 && (req->ai\_flags & AI\_V4MAPPED))

{

...

at->family = AF\_INET6;

}

else

return -EAI\_ADDRFAMILY;

...

}

...

if (at->family == AF\_UNSPEC && (req->ai\_flags & AI\_NUMERICHOST) == 0)

{

...

size\_t tmpbuflen = 512;

char \*tmpbuf = alloca (tmpbuflen);

...

rc = \_\_gethostbyname2\_r (name, family, &th, tmpbuf,

tmpbuflen, &h, &herrno);

...

}

Take another SUID-root binary as example.

In here also it calls getaddressinfo() instead of gethostbyname\*() function. Therefore, it is also not vulnerable.

#ifdef ENABLE\_IPV6

/\* gethostbyname2() is deprecated so we'll use getaddrinfo() instead. \*/

...

error = getaddrinfo( Hostname, NULL, &hints, &res );

if ( error ) {

if (error == EAI\_SYSTEM)

perror ("Failed to resolve host");

else

fprintf (stderr, "Failed to resolve host: %s\n", gai\_strerror(error));

exit( EXIT\_FAILURE );

}

...

#else

host = gethostbyname(Hostname);

if (host == NULL) {

herror("mtr gethostbyname");

exit(1);

}…

#endif

Process I Have Done Here to Exploit GHOST Vulnerability

I have chosen CENTOS linux os to check and exploit the GHOST vulnerability

first of all, we have to type ldd --version in the terminal.

In here you will get a small description saying which centos version you have. In my case version is 2.17, but keep in mind that any centos version which is before 2.18 is likely to be vulnerable. And from version 2,18 onwards they are patched.

So, in below it shows how execution happens when we use ldd –version

A screenshot of a social media post

Description automatically generated

Then we are going to run “ **yum list glibc** “ to pull up all the packages we have within the GNU library( glibc )

glibc is the most critical part in the operating system. So, we have to make sure which packages have to be updated.

Let us see the outcome if we type “ yum list glibc “

A screenshot of a social media post

Description automatically generated

From above figure it shows that what are the available packaged we have and what are the installed packages within glibc

The next step is to run “**sudo yum update glibc**”. This will display updates you have and asking for the permission to update other packages.

A screenshot of a cell phone

Description automatically generated

This figure shows the updates we have

* And this will show what dependencies we have. In the below figure you can see that it is displaying how to resolve dependencies.

Then it will check the transactions which are running and processing dependencies within glibc.

A screenshot of a cell phone

Description automatically generated

A screenshot of a social media post

Description automatically generated

This will take few minutes to update all the packages

After updating all the dependencies, the terminal will ask for a user input.

In that case we need to enter ‘y’ as yes.

A screenshot of a social media post

Description automatically generated

So ,in this case it still requires some down loadings.

And we have another user input again. In that also we must press “y”

A screenshot of a cell phone

Description automatically generated

So, after updating all the packages, then it is patched or exploited if there is an existing GHOST vulnerability.

A screenshot of text

Description automatically generated

The above figure shows how we have completed all the dependencies we have. So now the os is patched from GHOST vulnerability.

Another point to mention is, if we type ldd –version again in the terminal it will displays you the current version before the updates. So,don’t be upset in that case.

So simply you are required reboot your system again.

To do that you have to type “**reboot**” in your terminal.

We just know that we have exploit the vulnerability by updating glibc. But we should make sure that our system is not vulnerable anymore

In this case we have reboot the system and open the terminal again

If you don’t have enough utilities in the gcc you can install the relevant updates needed for gcc

As earlier type:

**sudo yum install gcc**

this will install the GNU compiler.

In next step we need to use a code or any other published script to check the vulnerability.

If you are going to use script you can download it from following links:

<https://access.redhat.com/labs/ghost?/>

and we can use wget commands like:

wget <https://webshare.uchicago.edu/orgs/ITservices/itsec/Downloads/GHOST.c>

wget -o GHOST.c <https://gist.githubsercontent.com/koelling/efgb2b9d0be6dbabe>

and you can use codes which have been taken from web also:

in my case I have used a code to check or test the vulnerability:

A screenshot of a social media post

Description automatically generated

I have exploited the vulnerability so my output will display as “ NOT VULNERABLE”;

A screenshot of a social media post

Description automatically generated

References used to take information

<https://www.cyberciti.biz/faq/cve-2015-0235-patch-ghost-on-debian-ubuntu-fedora-centos-rhel-linux/>

<https://www.acunetix.com/vulnerabilities/web/the-ghost-vulnerability/>

<https://www.cyberciti.biz/faq/cve-2015-0235-ghost-glibc-buffer-overflow-linux-test-program/>

<https://www.thegeekstuff.com/2012/08/lsof-command-examples/>

<https://ghost.org/docs/concepts/vulnerabilities/>