Prevent Worm Spreading

Huagang XIE (xie@lids.org, http://www.lids.org)
Apr 6, 2003 , version 0.2

In this paper, we will introduce a way to prevent worm spreading by restricting process network access, including its method and implementation.

Contents

1 Introduction

From the first Morris worm to the newest Microsoft SQL slammer. Network worms spread wider and faster than ever before. Worms get much more damage than any other attacks. For Linux, the most recent worm is apache ssl slapper worm which happen on September 2002 and attacked more than 13,000 hosts in 24 hours[1].

1.1 Worm definition

Worm is a program that makes copies of itself; for example, from one disk drive to another, or by copying itself using email or another transport mechanism. The worm may do damage and compromise the security of the computer. [2]

From the definition of worm, we know that worm is always trying to propagate itself from the infected machine.

1.2 Worm's behaviors

Let's see what is the different behaviors between a normal application and a being attacked application. In normal case, an application will accept connection from a remote client, and then serve the request and then send back the request. But when the application is being attacked, other than the normal operation, the application will also try to connect outside to propagate itself.

So the difference is when under attack, the application will try to connect outside. if we can restrict an application network connection, we can prevent a worm spreading. and make it stop within application.

1.3 3-tier to prevent worm spreading

1.3.1 Firewall

A properly configured firewall can stop worms at the first tier. For example, sql slammer worm attacks on port 1344, if the firewall filtered out this port, the worm can not spread into the internal the network. But when the port has to be open for publish access, such as HTTP, SMTP, this kind of attack can not be easily blocked for the nature of firewall. For example, firewall can not block apache ssl worm attacking an apache httpd server in a DMZ.

1.3.2 Network Based IDS

Network based IDS is the second tier to prevent worm spreading. Most IDS product provide anomaly based and signature based technology to prevent known and unknown attacks. Being applied inline, NIDS can be a very effective way to prevent worm spreading. For example, NIDS can be easily drop CodeRed, Apache SSL slapper. But for NIDS, it is difficult to prevent unknown attacks which is also logistically valid in case of RFC.

1.3.3 Anti-virus and Host based IDS

This final tier is on the host. anti-virus can check the worms which having disk-access, but can not prevent worms like sql slammer which do not have disk access. Host Based IDS can prevent this kind of attacks base on process profile. For example, a SQL process would not connect outside to port 1344, so any network connection activity from the process to outside port 1344 will be denied.

You can not prevent worm effectively without any of these 3 tiers.

LIDS reside on the 3nd tier and prevent worm spreading by restrict the process network access.

2 LIDS prevent worm spreading

2.1 What is LIDS

LIDS is kernel patch to enhance the kernel security released under GPL. LIDS for kernel 2.5 use the LSM(Linux Security Modules)[4] framework and extend it to make it work with NETFILTER to make the network control more granulated. Since LIDS naturally have the capability to inheritance, so any restriction on the process will also restriction it on its children. In next sections, we will only say "process", but that will means "process and its children".

LIDS provides two ways to deal with this situation according to difference application's behaviors.

2.2 Socket Restriction

LIDS provide a way to control network access on a specific application. These controls include socket create, connect, shutdown, socket bind, accept. In this way, we can disable the application's socket create, in that way, a worm will not have the ability to connect outside. This way apply to application which do not need to connect outside, like HTTP, FTP in passive mode. If we apply this restriction to apache, we can avoid the apache ssl worm.

2.2.1 LIDS_SOCKET_CREATE, LIDS_SOCKET_TCP/UDP - Socket Creation

Any network connection thought TCP/IP network must associate with a socket, this is the first step to do any network activity. LIDS provide 3 ways restrict the socket creation.

LIDS_SOCKET_CREATE, when disable in a process, this will disable all the new network operation, like new connection, new bind within the process, but will not prevent the connection from outside to the process. This is very useful when one process do not do bind to a new port or make a new connection in its normal operation stage.

For example, if service A will not do any connection or bind to new port, so you can just simply disable the LIDS_SOCKET_CREATE. That will help you prevent a worm spreading outside thought a new connection or the worm trying to bind a port to install a backdoor.

In the real world,Apache SSL slapper worm attack, the worm will try to install a backdoor listen on a special port and then connect outside to propagate itself. If we disable the capability of the socket connection, the backdoor installation and worm propagation will be stop.

LIDS_SOCKET_CREATE_TCP, when disable, will disable the socket operation for TCP only.

LIDS_SOCKET_CREATE_UDP, when disable, will disable the socket operation for UDP only.

2.2.2 LIDS_SOCKET_CONNECT - Socket Connection

When a new connection is establish, it will do the socket connection first, so if we disable this capability, we will disable any new connection from the process.

2.2.3 LIDS_SOCKET_BIND Socket bind

When an application try to listen on a port to service the remote client, it will first do a socket bind first, so if we disable this capability, we will disable any operation trying to bind to a special port. This is very helpful to stop backdoor installation.

2.3 Packet label and Netfilter control

Some application need to connect outside, for example DNS, SMTP, TELNET, SSH or even HTTP sometimes need try to resolve hostname from remote DNS server. In this case, we can not simply just disable the socket or disable all the connections which may stop some valid connections.

NETFILTER[3] provide a very good method to control packet via the mangle table. In mangle table, a hook in the NETFILTER can alter the packet and also take actions based on the packet's special value. It has the capability to label a packet with a value and also match a packet with the special value and then take an action on it. Netfilter also provide a PID, UID and GID or session based match. Using the netfilter, one can also implement a very similar function as what LIDS do, but it also need some dynamic rules to support it.

LIDS provide a way to label all the packets coming out from a process and its children with a value and then you can use NETFILTER to control it. For example, you can label all the packets generated by DNS to 6, and then use NETFILTER to restrict the marked packet as 6, in this way, you have the total control over all the packets generated by DNS and its children. Say if just want it to connect to an DNS server outside, you can simply use iptables to write a rules for it. For Telnet, SSH, you can use this to restrict the users of ssh/telnet can only access some side outside.

Since all the restrict is on an application and its children, so if the worm get a shell and run command, the shell and the commands will also under the restriction.

3 Implementation

LIDS extend its protection from FILESYSTEM, CAPABILITY to SOCKET by adding some data structure and also extent it by providing a patch to NETFILTER to make LIDS work with NETFILTER seamlessly.

3.1 Data Structure for Socket

First of all, LIDS has a central ACL database. It contains all the ACL predefined. When a process executed, kernel will attach an ACL to it based on the search on the database. The TASK ACL defined as,

```
/* security/lids/include/linux/lidsif.h */
struct lids_sys_acl {
        unsigned long int ino;          /* the subject node number */
        unsigned long flags;            /* capability flags */
        unsigned long lids_cap;         /* Move from tsk*/
        unsigned long socket;           /* socket */
        struct lids_cap cap[32];        /* inheritable array*/
        int forked;                     /* fork tags */
        int mark;                     /* fork tags */
        int port[LIDS_PORT_ITEM][2];
        struct lids_acl *lids_acl;      /* object acl */
        struct lids_acl *lids_domain;
        kdev_t  dev;                    /* the subject dev number */
        struct task_struct *tsk;        /* back to the pointer */
};
/*  end of lids_sys_acl */
```

You will see there is member "socket" in the structure. "socket" is a 32bits integer, each bit will represent a socket operation which defined as,

```
/* security/lids/include/linux/lidsif.h */
#define LIDS_SOCKET_CREATE      0
#define LIDS_SOCKET_CONNECT     1
#define LIDS_SOCKET_BIND        2
#define LIDS_SOCKET_LISTEN      3
#define LIDS_SOCKET_ACCEPT      4
#define LIDS_SOCKET_SENDMSG     5
#define LIDS_SOCKET_RECVMSG     6
#define LIDS_SOCKET_GETSOCKNAME 7
#define LIDS_SOCKET_GETPEERNAME 8
#define LIDS_SOCKET_GETSOCKOPT  9
#define LIDS_SOCKET_SETSOCKOPT  10
#define LIDS_SOCKET_SHUTDOWN    11
#define LIDS_SOCKET_CREATE_TCP  12
#define LIDS_SOCKET_CREATE_UDP  13
#define LIDS_SOCKET_NF_MARK     14
```

When one bit set, it means "disable" that socket operation. But for NF_MARK is a special value here. when set, it means the process have the ability to mark the packet as a special value from "mark" in the same data structure.

3.2 socket operation restriction.

All the implementation use LSM hooks, for example, LSM provide a hooks to socket_create() as

```
/* net/socket.c */
int sock_create(int family, int type, int protocol, struct socket **res)
{
......
        err = security_socket_create(family, type, protocol);
        if (err)
                return err;
.....
}
```

and in LIDS, we register the hook first,

```
struct security_operations lids_security_ops = {
......
        .socket_create =                lids_socket_create,
......
}
```

and then, we implement the checking in lids_socket_create() as following,

```
/* security/lids/lids_lsm.c */
static int lids_socket_create (int family, int type, int protocol)
{
        if( lids_load && lids_local_load && family==AF_INET) {
                if ( lids_socket_perm(current,LIDS_SOCKET_CREATE) < 0 ) {
                        lids_security_alert("Attempt to create socket");
                        return -EPERM;
                }
                if( type == SOCK_DGRAM ) {
                        if ( lids_socket_perm(current,LIDS_SOCKET_CREATE_UDP) < 0 ) {
                                lids_security_alert("Attempt to create udp socket");
                                return -EPERM;
                        }
                } else if( type == SOCK_STREAM) {
                        if ( lids_socket_perm(current,LIDS_SOCKET_CREATE_TCP) < 0 ) {
                                lids_security_alert("Attempt to create tcp socket");
                                return -EPERM;
                        }
                }
        }
        return 0;
}
/* ---------------- end of socket_create ------------------------- */
```

the lids_socket_perm will check the current task's socket value, to see if correspond bit has been set or not, when set, it will return "-1" and the system_call will return an error "Permission Denied".

3.3 Packet Label and Netfilter

LIDS use inode->i_security to store the marker information. In the socket_create(), after the socket create a "sock" data, LSM has a hook named as socket_post_create(), we hook it as follow,

```
static void lids_socket_post_create (struct socket *sock, int family, int type,
                                    int protocol)
{
#ifdef CONFIG_LIDS_NF_MARK
        struct lids_sys_acl *tsk_sys_acl;
        struct ipt_mark_target_info  *markinfo;

        if( lids_load && lids_local_load && family==AF_INET) {
                if ( lids_socket_perm(current,LIDS_SOCKET_NF_MARK) < 0 ) {
                        struct inode *inode = SOCK_INODE(sock);

                        if(inode) {
                                tsk_sys_acl=  current->security;
                                markinfo = kmalloc(sizeof(struct ipt_mark_target_info),GFP_KERNEL);
get mark info from the acl ---> markinfo->mark = tsk_sys_acl->mark;
mark it in i_security   ------> inode->i_security = markinfo;
                                LIDS_DBG("DEV: [%d %d] Mark socket as %d \n",
                                        current->pid, current->parent->pid,
                                       markinfo->mark);
                        }
                }
        }
#endif
        return;
}

/* ------------------------ */
```

But this is not end yet, because the packet do not get the mark info yet. LIDS provide a patch to netfilter's MARK module, ipt_MARK.c,

```
/* net/ipv4/netfilter/ipt_MARK.c */

static unsigned int target(struct sk_buff **pskb,
.......
#ifdef CONFIG_LIDS_NF_MARK
        if(hooknum == NF_IP_LOCAL_OUT) {
                struct iphdr *iph = (*pskb)->nh.iph;
                struct inode *inode;

                if (!(*pskb)->sk || !(*pskb)->sk->socket) {
                        return IPT_CONTINUE;
                }
                inode = SOCK_INODE((*pskb)->sk->socket);
                if(!inode || !inode->i_security)
                        return IPT_CONTINUE;

/* following code mark the packet based on the markinfo get from i_security */
                markinfo = (struct ipt_mark_target_info *)(inode->i_security);
                if((*pskb)->nfmark != markinfo->mark) {
                        (*pskb)->nfmark = markinfo->mark;
```

```
                        (*pskb)->nfcache |= NFC_ALTERED;
                }
        }
#endif
        return IPT_CONTINUE;
}
/* --------------- end of ipt_MARK patch -----------*/
```

The patch mark the packet based on the i_security value to the "nfmark". After that, the packet marking finished. You need a rule to active this marking, a rule like this will work,

```
        # iptables -A OUTPUT -t mangle -j MARK --set-mark 0
```

After that, all the packets generated will marked by a special value and we can use netfilter iptables rules to restrict it. We will show some examples based on it in the next section.

4 Examples

This section will show some examples on how to use LIDS and Netfilter to restrict the process network access. The examples may not fit your environment.

4.1 Prevent http worm.

In this section, we will provide an example to prevent worm spreading on a Apache httpd daemon. We assume you have install LIDS 2.0.3pre2 for 2.5.62 and later, and Netfilter modules.

First of all, you need to insert the necessary modules. then you can do

```
        # lidsconf -A -o /usr/sbin -j READ
        # lidsconf  -A -s /usr/sbin/httpd -o LIDS_SOCKET_CREATE -j DISABLE
```

After that, httpd itself can not connect outside and bind to any port. That will certainly stop the Apache's SSL slapper worm which propagate itself and trying to install a backdoor on the machine.

But if the http process need to connect outside, for example, it will try to connect to an database server or ad DNS server. We can use another Netfilter to control it,

```
        # lidsconf -A -s /usr/sbin/httpd -o LIDS_SOCKET_NF_MARK 3 -j DISABLE
                ^------------ this rule get the process and its children capability to mark all
                              the packet from new socket(new connection and new bind) the packets as 3.
        # iptables -A OUTPUT -t mangle -j MARK --set-mark 0
                ^------------ this rule make the LIDS mark the packet at this stage.
        # iptables -A OUTPUT -d 10.0.0.1 --protocol udp --destination-port 53 -m --mark 3 -j ACCEPT
                ^------------ this rule enable the access to 10.0.0.1 port 53 which is a DNS server.
        # iptables -A OUTPUT -d 10.0.0.2 --protocol tcp --destination-port 3306 -m --mark 3 -j ACCEPT
                ^------------ this rule enable the access to 10.0.0.2 port 3306 which is a MySQL server.
        # iptables -A OUTPUT  -m mark --mark 3 -j DROP
                ^------------ this rule drop all the packets marked as 3
```

Let's see what happen when a HTTP WORM coming in, the worm will try to install a backdoor on port 10999, it success since we do not prevent it, but when he try to connect to backdoor port 10999 from outside, it will fail for all the packets that backdoor generated have been marked as 3 and dropped silently by the rule when the packet go outside.

When the HTTP server try to resolve a hostname or try to send a database request to a remote database server, it can do it successfully. But when the worm trying to connection outside to port 80 and propagate, he will find out that he can not since the last netfilter rule will also drop the packet silently.

4.2 Prevent ftp worm

for passive mode only FTP,

```
# lidsconf -A -o /usr/sbin -j READ
# lidsconf -A -s /usr/sbin/in.ftpd -o LIDS_SOCKET_CREATE -j DISABLE
```

4.3 Prevent pop3 worm

```
# lidsconf -A -o /usr/sbin -j READ
# lidsconf -A -s /usr/sbin/in.pop3d -o LIDS_SOCKET_CREATE -j DISABLE
```

4.4 Restrict ssh and telnet user connection.

following rules will allow SSH and telnet user can only connect to 10.0.0.1

```
# lidsconf -A -s /usr/sbin/sshd -o LIDS_SOCKET_NF_MARK 5 -j DISABLE
# lidsconf -A -s /usr/sbin/in.telnetd -o LIDS_SOCKET_NF_MARK 5 -j DISABLE
# iptables -A OUTPUT -t mangle -j MARK --set-mark 0
# iptables -A OUTPUT -d 10.0.0.1 22 -m --mark 5 -j ACCEPT
# iptables -A OUTPUT -m mark --mark 5 -j DROP
```

4.5 Prevent backdoor install on sendmail

```
# lidsconf -A -o /usr/sbin -j READ
# lidsconf -A -s /usr/sbin/sendmail -o LIDS_SOCKET_BIND -j DISABLE
```

5 Summary

LIDS can restrict an application and its children's network access without affecting its normal operation. Under the restriction, LIDS can prevent worm spreading and backdoor installing.

6 Acknowledgment

Thanks Nils Toedtmann for the idea to integrate LIDS with Netfilter and all the LIDS users who discuss it on the mailing list. Thanks Rafal Wojtczuk for the review and good suggestion for this paper.

7 Reference

1. F-Secure, http://www.f-secure.com/slapper/
2. Symantec, http://securityresponse.symantec.com/avcenter/refa.html#worm
3. Netfilter, http://www.netfilter.org
4. LSM, http://lsm.immnuix.org