Design of the Kernel Hardening Function in the Linux Network Module

*Seung-Ju Jang**

Dongeui Univ. Dept. of Computer Engineering, Korea

Summary
A panic state is often caused by careless computer control. It could be also caused by a kernel programmer's mistake. It can make a big problem in computer system when it happens a lot. When a panic occurrs, the process of the panic state has to be checked, then if it can be restored, operating system restores it, but if not, operating system runs the panic function to stop the system in the kernel hardening O.S. To decide recovery of the process, the type of the panic for the present process should be checked. The value type and the address type have to restore the process. If the system process is in a panic state, the system should be designed to shutdown hardening function in the Linux operating system. So it has to decide whether the process should be restored or not before going to the panic state.
Key words:Kernel Hardening, Linux, Network Module, ASSERT() macro

Introduction

Nowaday, the usage of Linux O.S is increasing. The Linux O.S is open source. Therefore, many people and companies use this at web server, file server, and DB server. Somebody can modify and change part of device driver, file system, and network kernel source of the Linux. While the open source is not an advantage in terms of standard and system reliability[1, 2, 3, 4, 5, 6]. The Linux O.S is not commercial product so that it is not stable S/W. When you modify the kernel source or build a dynamic kernel module of Linux O.S, that can make a fatal system error. The error code makes system halt. In fatal error, it will make system data crash[2,13,14,15,16,17].

This paper designs kernel hardening function to recovery for the error in the network module of Linux kernel. The network hardening module in this paper recovers a memory data for recoverable memory using ASSERT() macro. The ASSERT() function is categorized into two types. One is address type. The other is value type. For example, the address type is “ASSERT(new != NULL, return -1;);”. The new variable is char type. The value type is ”ASSERT(self->magic == LAP_MAGIC, return;);“. The self->magic is value type. This paper proposes a new modified ASSERT() macro and it is applied to the network module of Linux kernel. The value type of ASSERT() macro will set correct value when it is incorrect value. The address type of ASSERT() macro will set correct address variable when it is incorrect address and it is possible recovery. Otherwise, the ASSERT() macro makes system panic.

This paper composed of related study in chapter 2, describing a design concept in chapter 3, experimental result in chapter 4, and conclusion in chapter 5..

2. Related Studies

The Linux O.S has usually hierarchical structure. Each level incurs several errors. Therefore, HW, kernel, and application program should be supported fault recovery mechanisms. All fault recovery procedures are related with each others. The fault recovery of O.S is kernel or system administration part. According to strength of fault recovery, the level is categorized five levels: L1-L5.

The commercial fault tolerant systems are Tandem, Fujitsu, Stratus, and DEC. The Tandem developed safety emergency system for OLTP(On-Line Transaction Processing) market. The Tandem system has a dual path of each Hardware element to guarantee fail-safe. A fault detection uses hardware and software mechanism. All processes running two machines which is process pair concept. Stratus system’s fault recovery supports failover concept that is guarantee non-stop system. The Fujitsu ’s fault recovery supports general business model.

The Tandem NonStop system aims non-stop operation during system running. So, this system has dual components for all elements of HW. The file system is not also exceptional. According to UI HAWG(UNIX International Hardware Working Group), this system architecture is perfect fault tolerance system. The Tandem NonStop system has a primary and backup process. “I’m alive” message originates from a primary process periodically. It signals no fault in the system. A backup process runs instead of a primary process during system fault. Fault recovery uses check point function. When a primary process has a fault, a backup process reruns the last check point position.

The Chorus O.S intends to open, distributed, and variable features. To satisfy these features, it adopts micro-kernel technology. The micro-kernel technology is supporting independent server which is existing O.S’s function. To satisfy these features in the Chorus, it provides asynchronous message exchange and

Manuscript received May 25, 2006.
Manuscript revised May 30 , 2006.
This paper was supported by the BB21 project.

RPC(Remote Procedure Call). The file system failover in the Chorus is a fault tolerance using takeover. Takeover function is running two process pairs. Two processes are a primary process and backup process. A backup process runs instead of a primary process during system fault for each server. The file system's server in the Chorus O.S is FM(File Manager). The two FM server do a fault tolerance's role [2].

The kernel hardening study didn't achieve much until now. The representative product of kernel hardening is the Monta Vista O.S. A Monta Vista company sells a CGE(Carrier Grade Edition) version Linux O.S which includes kernel hardening [3,6,7,8,9]. Monta Vista CGE classifies three categories in kernel hardening [3]. The kernel hardening is generally classified into code review, panic removal, and fault injection testing [3]. Code reviews prevents kernel code error originally by continuously checking a kernel code's error. Panic removal decides a process kill or panic by checking O.S code. Fault injection testing checks a Linux kernel can recover some S/W error or not.

The Monta Vista O.S rechecks kernel code by a code review concept. When a specific process enters a panic routine, the Monta Vista O.S kills that process. If the current killed process is system-related process, the kernel will panic the system. But if the error kernel code is made by a programmer, the kernel will just kill the process and will not disturb system running. The Monta Vista kernel hardening checks all kinds of kernel panic conditions. The kernel hardening guarantees high availability system [1, 3, 4, 10, 11, 12].

3. Design of the kernel hardening in the network module

3.1 Design of the kernel hardening

The suggesting basic concept of kernel hardening in this paper is that if a kernel error can be recovered the kernel hardening module recovers an error kernel code before kernel panic() execution. We can make a stable system by supporting kernel hardening. But the kernel hardening is not an almighty. Panic() execution can not make fatal error in some cases of kernel error. In those cases the kernel hardening module leads panic() execution. In order to implement the kernel hardening in this paper, the ASSERT() macro is available. The kernel hardening procedure in the ASSERT() macro is [Fig. 2].

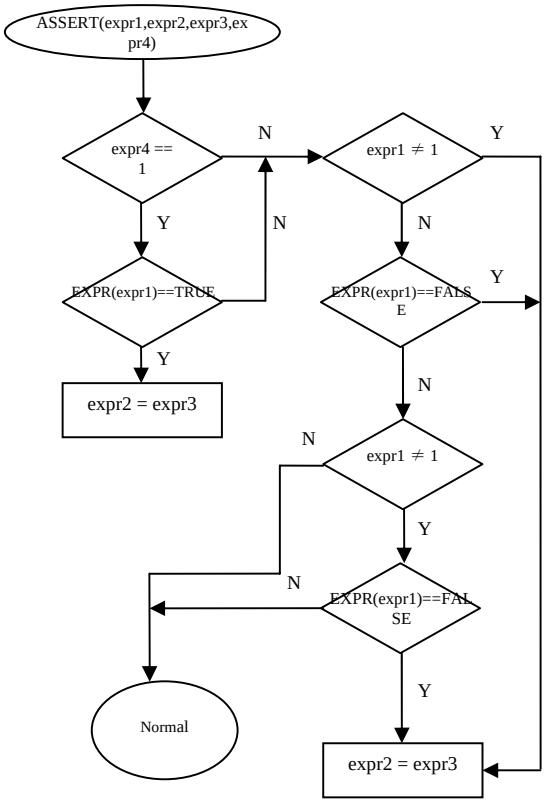


Fig. 1 ASSERT() macros procedure for the kernel hardening

The proposing kernel hardening in this paper is implemented ASSERT() macros function. The ASSERT() macros decides panic() function execution. When the macros can recover kernel code error, it will recover error kernel code. If the macros can recover error kernel code, the system acts normal operation. The [Table 1] shows the case whether recoverable kernel error or not.

[Table 1] The Recoverable Type of Kernel Hardening

The Recoverable Type of Kernel Hardening	If expression type is value in the ASSERT() macros, the wrong value is recovered.
	If expression type is address in the ASSERT() macros and that address is accessable, that address is recovered.

The recovery procedure is as following in the kernel hardening. The expression types are two kinds in ASSERT() macros.: value and address. In value type if the wrong value of the ASSERT() macros can be changed, the Linux kernel does not run panic(). The error kernel code can be recovered.

The proposing algorithm of the ASSERT() macros is as follow.: when expr1 that is argument in ASSERT() macros

is FALSE, the hardening algorithm decides value or address type. The expr1 value is determined result of "expr2==expr3". When expr1 is FALSE, the wrong expr2 value is changed with normal expr3 value in value type. In address type, access_ok() checks that the (expr2, expr3) arguments are an available memory area or not. And then the kernel hardening module decides recoverable or not.

When one of (expr2,expr3) is NULL, the kernel hardening module handles like this. The kernel hardening module run access_ok() function for this address. When the two comparing arguments are all NULL, the force_sig() kills a process so that it makes there is no exceptional events. The force_sig() removes the root of generating panic process. It only kills the process so that other processes run normally. The force_sig() only kills a user process. But, force_sig() does not kill the system process(daemon process). Because the system process does not kill forever during system is running. When the system process have an error condition, the kernel hardening module do panic() procedure. [Algorithm 1] shows kernel hardening procedure in the address type.

Algorithm RAT_KH_NM(Recovery-Address-Type for Kernel Hardening in Network Module)
Input : The set of the Expr = {expr1, expr2, expr3, expr4} and process id(pid)
Output : recovery variable, kill process or execute panic()

RAT1 : if address type then **RAT2**
else **RAT7**
end
RAT2 : expr1 = FALSE(in the wrong address value)
RAT3 : check the expr2 and expr3 are normal main memory address
if expr2 and expr3 are normal address
then **RAT4**
else
goto **RAT6**
end
RAT4 : if pid = user process then **RAT5**
else
goto **RAT6**
end
RAT5 : force_sig_kill(pid)
RAT6 : Panic()
RAT7 : Stop

[Algorithm 1] Procedure of Kernel Hardening in the Address Type

[Algorithm 1] shows kernel hardening procedure in the address type of Linux network module. Left side(RAT) of [Algorithm 1] shows the statement number. When expr4 is 2 in ASSERT() macros and address type, the kernel hardening module checks expr1 is TRUE or FALSE. The kernel hardening module proceeds normal procedure when expr1 is TRUE. The kernel hardening module checks whether expr2 and expr3 is normal memory address or not when when expr1 is FALSE. The access_ok() that needs three arguments checks legal memory address. The first argument is readable or writable memory. The second argument is memory address. The third argument is memory size.

The return value of access_ok() is 0 that the memory address is normal. The return value of access_ok() is 1 that the memory address is abnormal. One return value of access_ok() among expr2 and expr3 is 1 that is abnormal memory address. All return values are 0 that is normal memory address.

3.2 System Architecture

The designed system architecture is Intel CPU 450MHz Processor, 128 MByte RAM and RedHat 9.0 Linux O.S. The Linux kernel version is 2.4.20. The GNU tool is used to develop program.

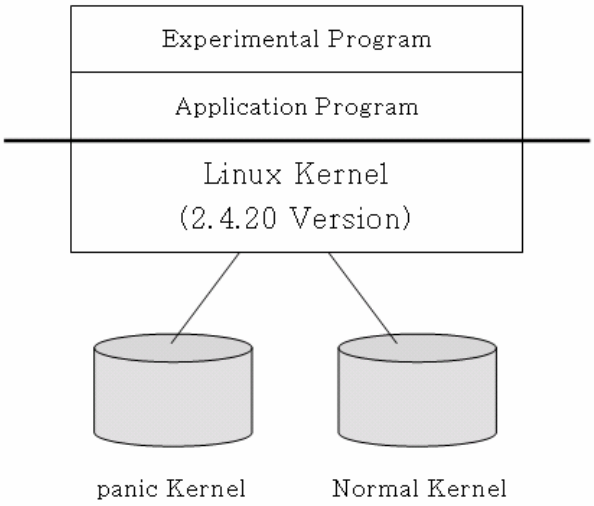


Fig. 2. System Architecture

3.3 Design of Kernel Hardening in a Network Module

The ASSERT() macros types are four in Linux network module. This four types are [Table 2].

[Table 2] ASSERT() macros Types

	route.c ip_rt_init( )	tcp_ipv4.c tcp_v4_init()	tcp.c tcp_init( )	tcp_diag.c tcpdiag_init( )	icmp.c icmp_init( )
ASSERT_TCP_CACHE	O		O		
ASSERT_TCP_PAGE	O		O		
ASSERT_NETLINK				O	
ASSERT_CREATE		O			O

[Table 2] shows implemented network module of kernel hardening in a types of ASSERT() function. ASSERT_TCP_CACHE()and ASSERT_TCP_PAGE() macros shows a kernel hardening in TCP related part. This macros is in implemented into route.c tcp.c source code. kmem_cache_create() function uses ASSERT_TCP_CACHE() macros in network module. __get_free_pages()function uses ASSERT_TCP_PAGE() macros in network module. ASSERT_NETLINK() macros is applied to network diagnostics. ASSERT_CREATE() macros is available on TCP protocol and ICMP protocol for kernel hardening. Some source files of directory Linux/net/ipv4 makes a panic(). Tcp_init() function of tcp.c source file makes a panic(). When a specific variable of tcp_init() function is NULL, system makes a panic. This variable's value is determined return value of __get_free_pages() and kmem_cache_create (). The route.c source file has also this kind panic. These types of variable is [Fig. 3].

```
tcp_openreq_cachep =
kmem_cache_create("tcp_open_request",sizeof(struct
open_request),0,
                SLAB_HWCACHE_ALIGN,NULL, NULL);
if(!tcp_openreq_cachep)
    panic("tcp_init: Cannot alloc open_request cache.");
```

```
tcp_bucket_cachep =
kmem_cache_create("tcp_bind_bucket",sizeof(struct
tcp_bind_bucket),0,
                SLAB_HWCACHE_ALIGN, NULL, NULL);
if(!tcp_bucket_cachep)
    panic("tcp_init: Cannot alloc tcp_bind_bucket cache.");
```

```
tcp_timewait_cachep =
kmem_cache_create("tcp_tw_bucket",sizeof(struct
tcp_tw_bucket),0,
                SLAB_HWCACHE_ALIGN,NULL, NULL);
if(!tcp_timewait_cachep)

    panic("tcp_init: Cannot alloc tcp_tw_bucket cache.");
```

```
tcp_ehash = (struct tcp_ehash_bucket
*)__get_free_pages(GFP_ATOMIC, order);
if(!tcp_ehash)
    panic("Failed to allocate TCP established hash table\n");
```

```
tcp_bhash = (struct tcp_bind_hashbucket
*)__get_free_pages(GFP_ATOMIC, order);
if(!tcp_bhash)
    panic("Failed to allocate TCP bind hash table\n");
```

[Fig. 3] Panic Decision Variable Type in Network Module

The condition decides panic() execution or not. The designed kernel hardening is running that the panic() function is executed in case of a panic condition. That is, The new designed ASSERT() macros is applied to the kernel. The argument values are using in the ASSERT() macros. The new designed ASSERT() macros is defined [Fig.4], [Fig. 5].

```
#define
ASSERT_TCP_CACHE(expr1,expr2,expr3,expr4,
                 expr5,expr6,expr7,expr8) \
  if(!expr1){\
      expr1 = kmem_cache_create(expr2,expr3,
                 expr4,expr5,expr6,expr7);\
      if(expr1 == NULL){\
        expr8=1;\
      }else{\
          expr8=0;\
      }
  }
```

[Fig. 4] The new designed ASSERT macros in the kmem_cache_create() function

```
#define ASSERT_TCP_PAGE(expr1,expr2,expr3,expr4) \
  if(!expr1){\
      expr1 = __get_free_pages(expr2, expr3);\
      if(expr1 == NULL){\
          expr4=1;\
      }else{\
          expr4=0;\
      }\
  }
```

[Fig. 5] The new designed ASSERT macros in the __get_free_pages() function

Let us look about decision condition in ASSERT() macros, you can think the ASSERT(a != NULL, a, NULL, 2) case. When we cannot decision for the recoverable address value in ASSERT() macros, it is efficient that exits the "panic" incurring process using "force_sig(SIGKILL, p_pid)" rather than "panic" proceeding. The user process is applied to process exit of force_sig(). The system process(like daemon) should not exit using force_sig() because it

is abnormal system down. Accordingly, the system process should do “panic” procedure rather than “exit” process procedure. A way of classification user and system process is PID number(p_pid) in proc. table. The usage format of ASSERT() based on [Fig. 5] is [Table 2].

Table 2. An Argument Value for ASSERT() Macros

Arguments	function
expr1	Obtain result of expr2 == expr3
expr2, expr3	Variable or address for a specific value
expr4	Decides address or value

[Fig. 5] shows the kernel hardening source code that is really applied to network module in Linux O.S. The applied network function is “tcp_init()” that is executed “ping(1)” command.

```
. . . . .
if (!tcp_ehash){
      ASSERT_TCP_PAGE(tcp_ehash,GFP_
            ATOMIC,order,p_flag);
    if(p_flag==1)
      panic("Failed to allocate TCP established
                  hash table\n");
}
for (i = 0; i < (tcp_ehash_size<<1); i++) {
      tcp_ehash[i].lock = RW_LOCK_UNLOCKED;
      tcp_ehash[i].chain = NULL;
}
do {
      tcp_bhash_size = (1UL << order) *
      PAGE_SIZE /sizeof(struct tcp_bind_
                  hashbucket);
    if ((tcp_bhash_size > (64 * 1024)) &&
                  order > 0)
      continue;
    tcp_bhash = (struct tcp_bind_hashbucket *)
        __get_free_pages(GFP_ATOMIC, order);
} while (tcp_bhash == NULL && --order >= 0);
. . . . .
```

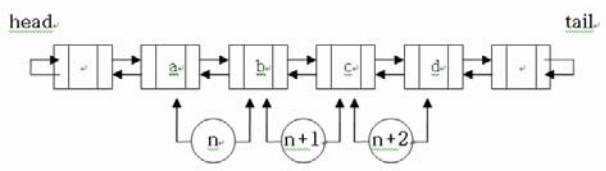
[Fig. 6] Example Code of Kernel Hardening in Network Module

ASSERT_TCP_PAGE() function is kernel hardening code in [Fig.4] and [Fig. 5]. tcp_돔노 variable is a return value of _get_free_pages(GFP_ATOMIC, order) in this function. p_flag is assigned to 0 in case of allocating tcp_ehash and 1 in case of non-allocating tcp_ehash.
ASSERT_TCP_PAGE() function is added into primitive network-related function for kernel hardening in [Fig. 6]. The tcp_ehash, order variables are primitive variables in ASSERT_TCP_PAGE() function. GFP_ATOMIC is defined value to assign memory allocation. p_flag has 1 value which is added when unrecoverable kernel. The panic() function is executed when this variable has 1. Otherwise kernel recovery is doing in the ASSERT_TCP_PAGE() macros.

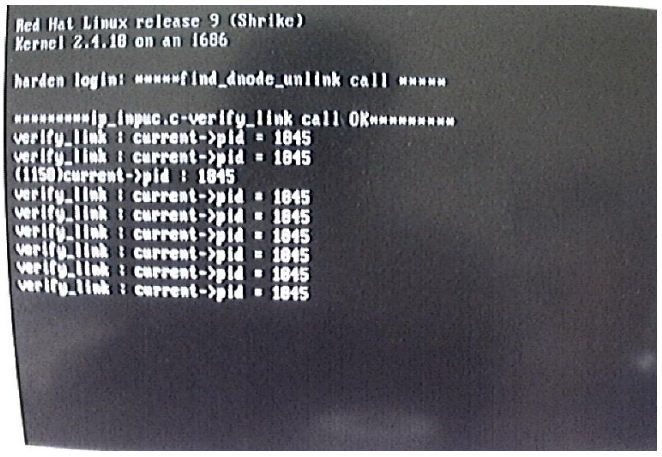
4. Experiments

4.1 Experiments Methods

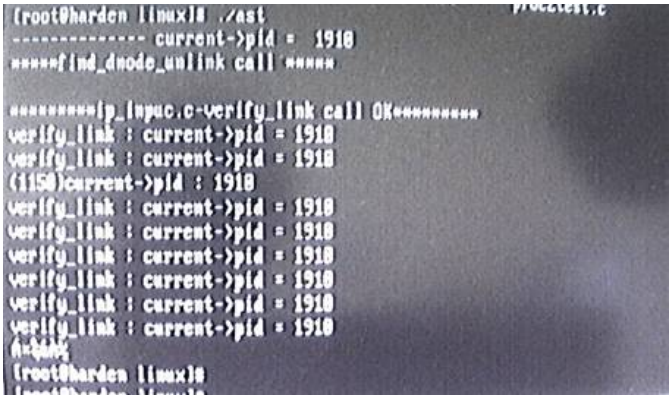
The suggesting kernel hardening algorithm has implemented and experimented in the Linux O.S. The implemented source code is compiled. We can test the implemented module. That booting kernel initializes double linked list in the ip_rcv(). When intermediate link pointer unlinks, the kernel cannot find next node. At this time, in the ASSERT() macros calls do_page_fault() of fault.c like [Fig. 6]. The do_page_fault() executes die() function so that the system makes panic state. The experiment case of double linked list has a wrong forward link pointer, but previous link pointer has a correct link pointer. At this time the wrong link pointer can be recovered by set the previous link pointer. Without system panic, the system is recoverable.



[Fig. 7] Double Linked List



[Fig. 8] Execute ip_rcv() function



[Fig. 9] Execute asl program

5. Conclusion

The wrong operation and program makes an error in the Operating System and halts the system. If a kernel has a problem, it makes serious problem in the computer system. In this case, if we can recover a kernel problem so that makes no error status system, the system runs well. This paper designs kernel hardening module which recovers some recoverable errors in the Linux O.S.

The recoverable errors are killing error process and memory error in the Linux O.S. The suggested kernel hardening module is implemented in the ASSERT macro. The function of killing error process is implemented into ASSERT macro. The ASSERT macro checks whether condition is FALSE or TRUE. If FALSE condition, the kernel hardening module checks that is recoverable or not. If recoverable process, the kernel hardening module kills that process. Otherwise, the kernel hardening module runs panic() function and normal procedure as original Linux O.S method. I experimented the suggested kernel hardening module. As the experiment, the suggested kernel hardening module is working well.

References

[1] Beck, Linux Kernel Programming, pp2 ~ 5, ADDISON WESLEY,2002.

[2] http://hpc.postech.ac.kr/~dolphin/research/ds/mighty/design/designfault.html,2000.

[3] Jeffery Oldham & Alex Samuel, Advanced Linux Programming, pp45-55, Mark Mitchell, 2001.

[4] John Mehaffey, Montavista Linux Carrier GradeEdition[WHITE PA PER],Montavista Software Inc., April 8, 2002.

[5] Tim Udalll, "kernel Hardening Guidelines", SEQUOIA, 1994.

[6] SILBERSCHATZ&GALVIN&GAGNE, Operating System Concepts(6th), JOHNWILEY&SONGS INC. 2002.

[7] Software Fault Tolerant, http://user.chollian.net/ ~hsn3/korea/study_k2.html, 2000.

[8] http://www.mvista.com/cge/index.html, 2002.

[9] Michael Beck,Mirko Dziadzka,Ulrich Kunitz and Harald Bohme, Linux Kernel Internals, Addison-Wesley,1997.

[10] The Linux Online, http://www.linux.org

[11] Gary Nutt, Kernel Projects for Linux,Addison Wesley L-ongman, 2001.

[12] A.Rubini&J.Corbet, Linux Device Driver(2nd), O'Relly, 2001.

[13] BOVET & CESATI, OREILLY, Understanding the Linux Kernel, p216-p222, 2001.

[14] http://nodevice.com/sections/ManIndex/man055.html, 2002.

[15] G.B.Adams III, and H.J.Siegel,"The Extra Stage Cube: A Fault-Tolerant Interconnection Network for Supersystems, pp.443-454. IEEE Trans. on Computer. Vol. C-31, No.5 May 1982.

Seung-Ju, Jang received a B.Sc. degree in Computer Science and Statistics, and M.Sc. degree, and his Ph.D. in Computer Engineering, all from Busan National University, in 1985, 1991, and 1996, respectively. He is a member of IEEE and ACM. He has been an associate Professor in the Department of Computer Engineering at Dongeui University since 1996. He was a member of ETRI(Electronic and Telecommunication Research Institute) in Daejon, Korea, from 1987 to 1996, and developed the National Administration Multiprocessor Minicomputer during those years. His current research interests include fault-tolerant computing systems, distributed systems in the UNIX Operating Systems, multimedia operating systems, security system, and parallel algorithms.