



SHELLCODE

Security Research Team – September 2006

Registration Weakness in Linux Kernel's Binary formats



Polluting `sys_execve()` in kernel space without depending on the `sys_call_table[]`

Table of Contents

[Disclaimer]	2
[Introduction]	3
[Description] <code>exec()</code> implementation	4
[Analysis] The Binary Format list	6
[POC] Inserting a new binary format.....	8
[Workaround] Kernel patching	11
[About]	16



Acknowledgements: Ignacio Marambio Catán, Jonathan Sarba



SHELLCODE

Security Research Team – September 2006

Registration Weakness in Linux Kernel's Binary formats



[Disclaimer]

La información del presente documento es provista sin ningún tipo de garantía. Ni SHELLCODE ni sus autores se harán responsable por el uso indebido del material presentado en el mismo.

Esta obra está licenciada bajo una Licencia Atribución 2.5 de Creative Commons. Para ver una copia de esta licencia, visite <http://creativecommons.org/licenses/by/2.5/> o envíenos una carta a Creative Commons, 543 Howard Street, 5th Floor, San Francisco, California, 94105, USA.



SHELLCODE

Security Research Team – September 2006

Registration Weakness in Linux Kernel's Binary formats



[Introduction]

El presente documento tiene como fin demostrar la debilidad de diseño encontrado en el manejo de listas simplemente enlazadas usada para la registraci3n de formatos de binario manejado por el kernel, el mismo afecta a toda la familia de kernel GNU/Linux (2.0/2.2/2.4/2.6), permitiendo la inserci3n de m3dulos de infecci3n en kernel-space que pueden ser utilizados con fines personales para la creaci3n de distintas herramientas de infecci3n, por ejemplo rootkits.

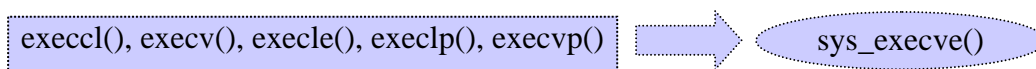
Si bi3n, la utilizaci3n de la t3cnica presentada en el presente documento podr3a ser utilizada con fines de protecci3n, prevenci3n, control en la ejecuci3n de binarios, y auditor3as; la contramedida podr3a verse afectada directamente por la misma t3cnica.

Cabe destacar que la t3cnica utilizada para manejar la lista simplemente enlazada la consideramos correcta, no as3 el contexto de las estructuras de datos involucradas.

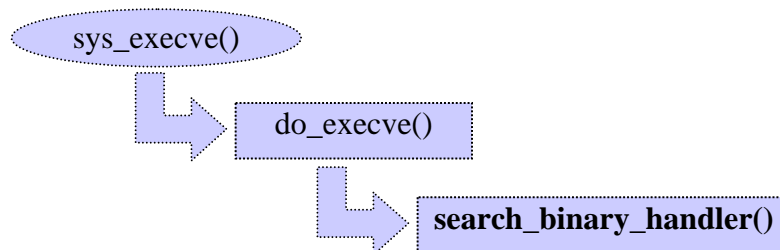


[Description] exec() implementation

Durante la creación de nuevos procesos la familia de funciones exec() será la encargada de reemplazar la imagen de un proceso por otro, en el momento que un proceso invoca alguna de las siguientes funciones, el binario especificado como argumento será el inicializado a través de su entry-point en la imagen ya reemplazada.



Las muchas funciones de la familia exec() existentes son pequeños wrappers a la llamada de sistema sys_execve()¹; esta pequeña función es la encargada de llamar a la función do_execve()¹ encargada de inicializar la estructura linux_binprm, el manejo de memoria y las credenciales para luego invocar a la función search_binary_handler()², función en donde nos detendremos.



Quizás resulte sorprendente para más de uno saber que el kernel mismo es el encargado de diferenciar los distintos tipos de binarios existentes en el sistema, reconociendolos a partir de los primeros 4 (cuatro) bytes del archivo.

Para esto, la función register_binfmt()² realiza el agregado de los distintos tipos de formato de binario representados como una estructura linux_binfmt³, cada formato es insertado al principio de la lista global simplemente enlazada, denominada formats⁴. Esta función no ha sufrido cambios desde la familia de kernel

1 REF[ia32]: /arch/i386/kernel/process.c

2 /fs/exec.c

3 /include/linux/binfmt.h

4 /fs/exec.c



2.0 hasta la 2.6, exceptuando los spinlocks para lidiar con las plataformas SMP de la actualidad.

```
/fs/exec.c
int register_binfmt(struct linux_binfmt * fmt)
{
    struct linux_binfmt ** tmp = &formats;
    if (!fmt)
        return -EINVAL;
    if (fmt->next)
        return -EBUSY;
write_lock(&binfmt_lock);
    while (*tmp) {
        if (fmt == *tmp) {
            write_unlock(&binfmt_lock);
            return -EBUSY;
        }
        tmp = &(*tmp)->next;
    }
    fmt->next = formats;
    formats = fmt;
write_unlock(&binfmt_lock);
    return 0;
}
```

[bold] Solo kernel 2.4/2.6

Es importante remarcar que la función solo comprueba si el formato agregado existe o no en la lista, por consiguiente se desprende que el recorrido completo de la lista no es un problema en la solución propuesta.



[Analysis] The Binary Format list

Del análisis de la estructura `linux_binfmt5`, se destaca el puntero a la función `load_binary`, reconocida como la función encargada de realizar la ejecución del programa especificado originalmente en cualquier función de la familia `exec()`. El resto de los punteros se corresponden con distintas comprobaciones útiles para algunos formatos, como por ejemplo del tipo ELF.

Es decir, cada formato de binario posee su propia función que le permite ejecutar el programa especificado según el tipo de formato que tenga. Dicha función está especificada como `load_binary5` en la estructura `linux_binfmt5` y es llamada en una iteración por cada elemento de la lista `formats`.

/fs/exec.c

```
int search_binary_handler(struct linux_binprm *bprm, struct pt_regs *regs)
{
    int try, retval;
    struct linux_binfmt *fmt;
    [...]
    for (fmt = formats ; fmt ; fmt = fmt->next) {
        int (*fn)(struct linux_binprm *, struct pt_regs *) = fmt-
>load_binary;
        if (!fn)
            continue;
    [...]
}
```

La función encargada de realizar esta tarea es `search_binary_handler()`⁶, donde durante la iteración y ejecución de la función especificada en cada elemento de la lista, la interacción y ejecución continuará hasta el final de la lista siempre que el valor de retorno de la función especificada en `load_binary5` devuelva -ENOEXEC.

⁵ /include/linux/binfmt.h

⁶ /fs/exec.c



/fs/exec.c (search_binary_handler)

```
[...]  
        if (retval != -ENOEXEC || bprm->mm == NULL)  
            break;  
        if (!bprm->file) {  
            read_unlock(&binfmt_lock);  
            return retval;  
        }  
[...]
```

Al momento de declarar e insertar un nuevo formato, debemos hacerlo con su correspondiente función, dicha función será invocada siempre antes que cualquier otra, en cada llamada a `sys_execve()`; solo es importante devolver en la nueva función el valor `-ENOEXEC` para que el programa original ejecutado lo haga exitosamente.

7 REF[ia32]: /arch/i386/kernel/process.c



[POC] Inserting a new binary format

A continuación se detalla la prueba de concepto, para la inserción de un nuevo formato de binario utilizando un módulo de kernel. Dicho código demuestra las facilidades para la creación de nuevos rootkits utilizando esta técnica.

MÓDULO: ghost.c

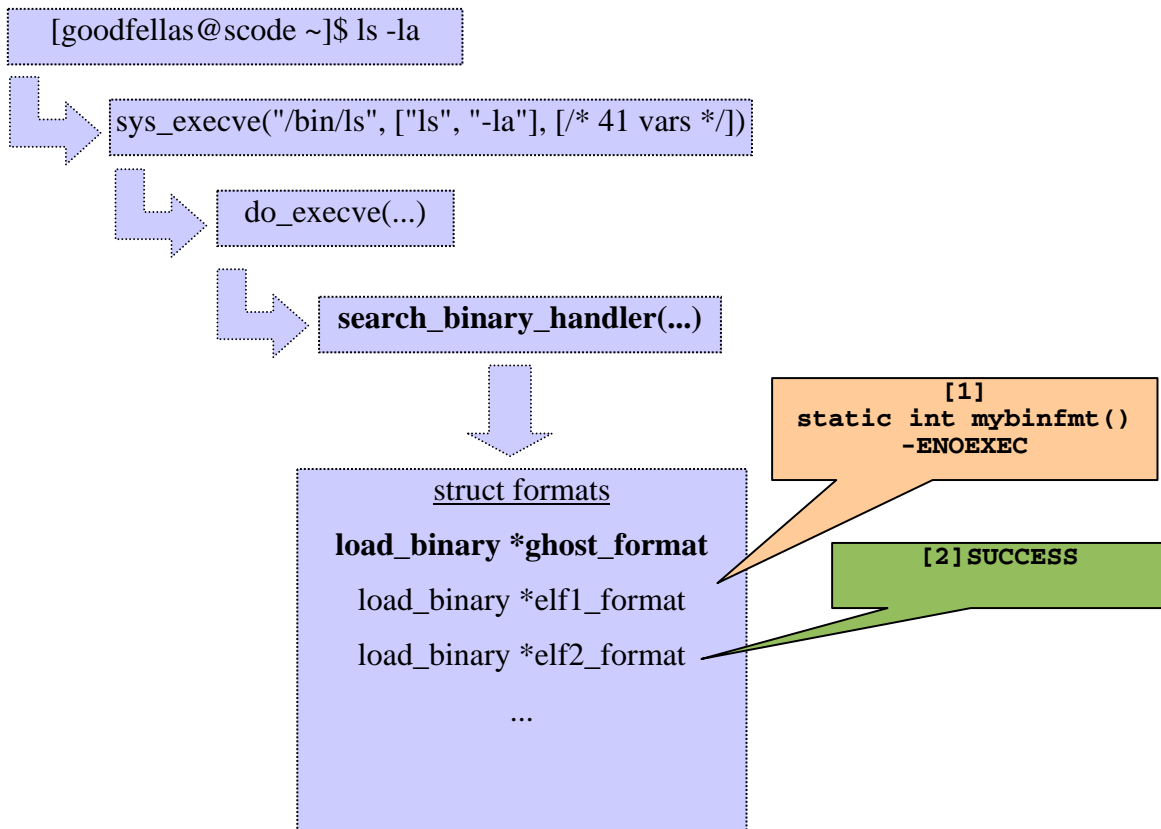
```
#define DRIVER_AUTHOR "SHELLCODE Security Research Team"
#define DRIVER_DESC  "Registration Weakness in Kernel's Binary format"
MODULE_LICENSE("GPL");
static int mybinfmt(struct linux_binprm *bprm,struct pt_regs *regs){
    printk(KERN_ALERT "Hello not printable characters!\n");
    return -ENOEXEC;
}
static struct linux_binfmt ghost_format = {
    .module          = THIS_MODULE,
    .load_binary     = mybinfmt,
};
static int __init ghost_init(void){
    printk("Loading ghost\n");
    return register_binfmt(&ghost_format);
}
static void __exit ghost_exit(void){
    printk(KERN_ALERT "Unloading ghost\n");
    unregister_binfmt(&ghost_format);
}
module_init(hello_3_init);
module_exit(hello_3_exit);
```

Kernel: 2.6.x



- La técnica utilizada podrá realizarse solo con privilegios de root.
- Los parches de kernel del tipo GRSEC8, y las implementación de seguridad a nivel de kernel del tipo SELINUX9, no contemplan la inserción de binarios no válidos ni las referencias de funciones en kernel-space para esta lista de formatos de binario.
- La demostración se realizó efectivamente para los kernel 2.4.x y 2.6.x, utilizando las siguientes versiones de kernel: 2.4.33 y 2.6.17.11

Se detalla un pequeño flujo de ejecución una vez cargado el módulo ghost.c utilizado durante las pruebas de concepto en nuestro laboratorio.



8 <http://www.grsecurity.net/>

9 <http://www.nsa.gov/selinux/>



SHELLCODE

Security Research Team – September 2006

Registration Weakness in Linux Kernel's Binary formats



En el flujo de ejecución puede verse, que ante la ejecución de cualquier comando, siempre es ejecutada la función inyectada previamente como parte del nuevo format de binario. Esto no puede detectarse utilizando herramientas del tipo strace/ltrace, ya que la función actua directamente en kernel-space desprendiendose directamente de la función `search_binary_handler()`¹⁰.

Esto afecta no solo a aquellos comandos ejecutados manualmente, a su vez estará afectando a todos los procesos del sistema, considerando que la llamada `sys_execve()`¹¹ puede ser invocada frecuentemente.

¹⁰ /fs/exec.c

¹¹ /arch/i386/kernel/process.c



[Workaround] Kernel patching

A continuación exponemos una de las posibles soluciones aplicadas a nivel kernel para la familia 2.0, 2.2, 2.4 y 2.6, cabe destacar que dicha solución requiere la recompilación del kernel en cuestión y conjuntamente el reinicio del equipo.

Las modificaciones fueron realizadas directamente sobre la función `register_binfmt()`¹², en el manejo de la lista simplemente enlazada llamada `formats`, haciendo que los nuevos elementos agregados a la lista ingresen en la última posición de esta.

Las diferencias presentadas fueron realizadas solo para las últimas versiones de kernel existentes al momento de la finalización del presente documento, a saber:

- **Kernel 2.0.40**
- **Kernel 2.2.26**
- **Kernel 2.4.33**
- **Kernel 2.6.17.13**

La implementación del parche se realiza copiando el archivo `diff` correspondiente dentro del directorio que contiene el kernel, y ejecutando la herramienta `patch -p0 < archivo-diff`.

Puede resultar que la aplicación de los siguientes parches apliquen a versiones anteriores de su propia familia de kernel.

¹² /fs/exec.c



PATCH: kernel 2.0.40

```
*** fs/exec.c 2006-09-26 18:17:52.000000000 -0300
--- fs/exec_new.c 2006-09-26 19:05:21.000000000 -0300
*****
*** 86,98 ****
        return -EINVAL;
        if (fmt->next)
            return -EBUSY;
!       while (*tmp) {
            if (fmt == *tmp)
                return -EBUSY;
            tmp = &(*tmp)->next;
        }
!       fmt->next = formats;
!       formats = fmt;
        return 0;
    }

--- 86,102 ----
        return -EINVAL;
        if (fmt->next)
            return -EBUSY;
!       if (*tmp == NULL) {
!           formats = fmt;
!           return 0;
!       }
!       while ((*tmp)->next) {
            if (fmt == *tmp)
                return -EBUSY;
            tmp = &(*tmp)->next;
        }
!       fmt->next = NULL;
!       (*tmp)->next = fmt;
        return 0;
    }
```



PATCH: kernel 2.2.26

```
*** fs/exec.c 2006-09-26 18:59:30.000000000 -0300
--- fs/exec_new.c 2006-09-26 19:00:54.000000000 -0300
*****
*** 94,106 ****
                return -EINVAL;
            if (fmt->next)
                return -EBUSY;
!         while (*tmp) {
                if (fmt == *tmp)
                    return -EBUSY;
                tmp = &(*tmp)->next;
            }
!         fmt->next = formats;
!         formats = fmt;
            return 0;
        }

--- 94,110 ----
                return -EINVAL;
            if (fmt->next)
                return -EBUSY;
!         if (*tmp == NULL) {
!             formats = fmt;
!             return 0;
!         }
!         while ((*tmp)->next) {
                if (fmt == *tmp)
                    return -EBUSY;
                tmp = &(*tmp)->next;
            }
!         fmt->next = NULL;
!         (*tmp)->next = fmt;
            return 0;
        }
```



PATCH: kernel 2.4.33

```
*** fs/exec.c 2005-01-19 11:10:10.000000000 -0300
--- fs/exec_new.c 2006-09-26 19:08:53.000000000 -0300
*****
*** 65,79 ****
    if (fmt->next)
        return -EBUSY;
    write_lock(&binfmt_lock);
!   while (*tmp) {
        if (fmt == *tmp) {
            write_unlock(&binfmt_lock);
            return -EBUSY;
        }
        tmp = &(*tmp)->next;
    }
!   fmt->next = formats;
!   formats = fmt;
    write_unlock(&binfmt_lock);
    return 0;
}
--- 65,84 ----
    if (fmt->next)
        return -EBUSY;
    write_lock(&binfmt_lock);
!   if (*tmp == NULL) {
!       formats = fmt;
!       write_unlock(&binfmt_lock);
!       return 0;
!   }
!   while ((*tmp)->next) {
        if (fmt == *tmp) {
            write_unlock(&binfmt_lock);
            return -EBUSY;
        }
        tmp = &(*tmp)->next;
    }
!   fmt->next = NULL;
!   (*tmp)->next = fmt;
    write_unlock(&binfmt_lock);
    return 0;
}
```



PATCH: kernel 2.6.17.13

```
*** fs/exec.c 2006-09-09 00:23:25.000000000 -0300
--- fs/exec_new.c 2006-09-26 19:10:52.000000000 -0300
*****
*** 76,90 ****
    if (fmt->next)
        return -EBUSY;
    write_lock(&binfmt_lock);
!   while (*tmp) {
        if (fmt == *tmp) {
            write_unlock(&binfmt_lock);
            return -EBUSY;
        }
        tmp = &(*tmp)->next;
    }
!   fmt->next = formats;
!   formats = fmt;
    write_unlock(&binfmt_lock);
    return 0;
}
--- 76,95 ----
    if (fmt->next)
        return -EBUSY;
    write_lock(&binfmt_lock);
!   if (*tmp == NULL) {
!       formats = fmt;
!       write_unlock(&binfmt_lock);
!       return 0;
!   }
!   while ((*tmp)->next) {
        if (fmt == *tmp) {
            write_unlock(&binfmt_lock);
            return -EBUSY;
        }
        tmp = &(*tmp)->next;
    }
!   fmt->next = NULL;
!   (*tmp)->next = fmt;
    write_unlock(&binfmt_lock);
    return 0;
}
```



SHELLCODE

Security Research Team – September 2006

Registration Weakness in Linux Kernel's Binary formats



[About]

SHELLCODE

Somos una empresa especializada en seguridad informática y comunicaciones que inició sus actividades en el año 1999. Nuestro equipo esta formado por expertos con más de 10 años de experiencia en el tratamiento de complejas infraestructuras tecnológicas y redes de alto rendimiento.

Actuamos en el campo de la seguridad de la información con una estructura dedicada a la investigación y al desarrollo, a la prestación de servicios profesionales y a la provisión de soluciones relacionadas.

Shellcode Security Research Team

GoodFellas es el equipo de trabajo dedicado a la investigación y desarrollo de nuevos conceptos, detección de vulnerabilidades y publicación de teorías relacionadas con la seguridad informática en las diferentes plataformas existentes en el mercado de las tecnologías de la información.

Puede contactarse con nuestro equipo, enviando sus consultas, comentarios y colaboraciones a GoodFellas@shellcode.com.ar.