

Cuando la complejidad es mala y el código abierto bueno

ATAQUE DEL ASESINO FTRACE

Cuando un kernel en fase de pruebas empieza a destrozar tarjetas de red, la comunidad se pone manos a la obra. **POR KURT SEIFRIED**

La complejidad es una cosa asombrosamente terrible. Una de las primeras CPUs de Intel, llamada 4004 y lanzada en 1971, contenía 2.300 transistores, los microprocesadores Pentium 4, lanzados en el 2000, tenía 48 millones, y una moderna CPU Quad core, alrededor de 2 billones de transistores. En el año 2008, el kernel 2.6.27 de Linux sobrepasó los 10 millones de líneas de código. Personalmente, mi experiencia ha sido que más transistores significan un ordenador mejor. Sine embargo, a pesar de un hardware más rápido, Linux parece necesitar el mismo tiempo para arrancar y presentar un escritorio de trabajo.

La Noche de las Tarjetas de Red

El pasado Agosto de 2008 se empezaron a aparecer algunos informes relativos a tarjetas de red e1000e [1]. No se trataba del típico "Actualicé mi driver y ahora la red no me funciona", sino de verdaderas llamadas de socorro tipo "Mi tarjeta de red ha dejado de funcionar, no se enumera en

el arranque y ahora está completamente muerta".

Lo bueno es que el hardware es barato. Lo malo es que el hardware es barato.

En un esfuerzo para ampliar las capacidades del hardware y reducir los costes, muchos proveedores han dado firmware actualizable a dispositivos de hardware (tarjetas de red, etc). El firmware les permite arreglar dispositivos que ya han sido vendidos a y puestos en funcionamiento por los clientes (lo que resulta mucho más baratos que su retirada física); también les permite actualizar hardware y añadir nuevas capacidades al vuelo. Evidentemente, esto significa que un dispositivo con firmware corrupto es poco probable que funcione adecuadamente o en absoluto.

A pesar de que es bastante obvio que actualizar a un kernel de en fase de pruebas no debería matar nuestra tarjeta de red, ¿qué hacemos cuando ocurre? Si somos desarrolladores del kernel de Linux, habitualmente sacamos nuestro depurador de kernel favorito y volvemos a crear las circunstancias que originaron el problema,

mirando con atención los cuelgues para entender qué pasó. Desafortunadamente, cuando comprobamos un error que supone cargarse tarjetas de red, puede llegar a ser difícil comprobar el problema (especialmente cuando intentamos convencer a otros para que reproduzcan la prueba en sus propios sistemas explicando que el nuestro tiene una tarjeta de red en desahuciada que no responde).

Debido a la naturaleza imperceptible del error y a la dificultad para comprobarlo, los primeros intentos para restringirlo y arreglarlo no tuvieron éxito.

Si somos Intel y tenemos un sistema operativo importante cargándose nuestras tarjetas de red, podemos coger un montón de tarjetas de red y un número de ingenieros ligeramente mayor y encerrarlos en una habitación. Éstos comenzarán probando diferentes versiones y parches de Linux para ver donde ocurre el problema. Una vez que hayamos localizado al parche o los parches responsables, podemos analizar el código y determinar qué está fallando.

¿Qué Fue Mal?

Una de las cosas para las que fue diseñado *ftrace* fue para suministrar rastreo de llamada de función simple. En este modo de operación se produce una línea por cada función llamada así como el llamante de esa función. *ftrace* utiliza los mecanismos de perfilado compilados en GCC, los cuales añaden sucesivamente un *mcount()* a cada función, de manera que cuando se la llama, se registran los datos. Desafortunadamente, esto penaliza el rendimiento (especialmente porque la mayoría de los sistemas no usan funcionalidad *ftrace*). Para evitarlo, los desarrolladores del kernel usaron parchado de memoria para sustituir las llamadas a la función con no operaciones (noops), algo que la mayoría de las CPUs gestionan muy rápidamente.



Sin embargo, esta solución tiene un inconveniente importante: Debemos parchear el código del kernel mientras éste está ejecutándose, algo que puede acarrear graves consecuencias si no se hace completamente bien. Para lograr estos objetivos de manera segura y rápida, ftrace realiza un registro de cada llamada a la entrada `mcount()` y las resuelve. Sin embargo, hacer uno cada vez es muy lento, así que los desarrolladores del kernel adoptaron el inteligente método de procesar por lotes los sustitutos de `mcount()`. Como resultado, existe una ventana de tiempo en la que la llamada `mcount()` deja de existir en el mismo espacio de memoria; por lo tanto, si el kernel lo parchea, estará de hecho parcheando un lugar de memoria que puede contener virtualmente todo. Los desarrolladores del kernel de Linux lo tuvieron en consideración y decidieron usar `cmpxchg` (compara y cambia) [2], el cual examina la memoria y la compara. Si coincide con lo que se esperaba, parchea entonces la memoria; pero si el contenido no coincide (por ejemplo, no está presente ninguna llamada a `mcount()`), no la parchea. En teoría, esto es completamente seguro.

En este punto las cosas se tornan complejas (como si no lo estuvieran ya). En arquitecturas de 32 bits, la memoria devuelta por `vmalloc()` y `ioremap()` comparten el mismo espacio de memoria. El kernel usa `vmalloc` para módulos que pueden cargarse, aunque el driver `e1000e` utiliza `ioremap()` para mapear su espacio de memoria. Una vez que ha acabado el módulo de inicialización de ftrace, se eliminan de la memoria las funciones `—init` y se libera el espacio. El driver `e1000e` usa entonces este espacio libre. Desafortunadamente, el comportamiento de `cmpxchg`

no está definido cuando se parchea la memoria del driver. Como probablemente haya adivinado el lector avisado, esto nos permite escribir, incluso si la memoria no coincidió.

Ahora normalmente, escribir al espacio de dirección de memoria de un driver simplemente haría que el driver fallara catastróficamente, peor no afectaría al hardware. Pero en el caso del driver `e1000e`, deja el hardware abierto para escribir, permitiendo por tanto al kernel escribir datos de memoria a su firmware y estropeando el hardware (en algunos casos pudo recuperarse reescribiendo el firmware). Las buenas noticias son que Intel ha reparado el driver. En el momento del arranque, éste evita cualquier escritura al firmware de la tarjeta y ftrace se ha arreglado para que no sobrescriba la memoria errónea.

Lo bueno es que para la mayoría de nosotros, los procesos de desarrollo del kernel de Linux funcionaron. Se sustituyeron nuevas funcionalidades en un kernel de prueba, luego fue ejecutado por desarrolladores y otra gente con sistemas de última generación (Ubuntu Intrepid), los cuales encontraron un problema serio, que fue resuelto. Los proveedores más importantes nunca lanzaron este kernel como opción predeterminada (tienden a ser conservadores con las actualizaciones del kernel por esta misma razón) y muy poca gente se vio afectada. Además, herramientas tales como `ethtool` pueden usarse para recuperar tarje-



Figura 1: El bug `e1000e` en portada.

tas dañadas escribiendo una nueva imagen de firmware para ellas. Como muchos tuvieron acceso a este código fuente, pudieron aplicar o eliminar varios parches fácilmente para localizar exactamente qué fragmentos de código eran los que causaban los problemas. Hacer esto en un entorno de fuente cerrado sería imposible (todo el mundo estaría a merced de los proveedores). Más aún: en un caso como el que nos ocupa, con dos proveedores, (SO y hardware), tendrían que cooperar para localizar el problema exacto. Con código bajo licencia GPL, todo el mundo puede comprobar todo el código y localizar el problema.

Conclusión

Este problema fue (potencialmente) uno de los peores errores en Linux desde hace mucho tiempo. A pesar de ser una cuestión increíblemente compleja (derivada de dos defectos separados relativa a hardware específico) se atajó de manera relativamente rápida y fue restringida en gran parte a un pequeño subconjunto de versiones de kernel.

Además, como es un tema de firmware, fue posible recuperar todo el hardware dañado con el programa `ethtool`. Intel, según informa, ha realizado imágenes del firmware disponible que puede usarse para recuperar tarjetas estropeadas [3].

Como todos tenemos acceso a la fuente, no sólo podemos saber que debemos reparar algún componente de nuestros sistemas, sino que también tenemos la posibilidad de hacerlo. ■

¡Pero el Estándar Dijo que 32 como Máximo!

Otro ejemplo de complejidad que echa por tierra todos los planes es el estándar inalámbrico 802.11. Dice que el identificador de set de servicio (SSID o ESSID) puede contener entre 0 y 32 caracteres (en ASCII se permiten valores desde 0 a 255), permitiendo que el nombre de la red sea lo suficientemente largo como para que sea legible por los humanos ("Free Coffe Shop Wi-Fi" versus "1ad834d7"). Simple, ¿verdad?

Error. El encabezamiento del marco para un paquete 802.11 usa un valor no firmado de 8 bits para especificar el espacio actual disponible para los datos, lo que significa que el parámetro SSID puede estar entre 0 y 255 caracteres. A menos que cavemos profundamente en la especificación y estuviéramos buscando cosas que podrían ir mal, no conoceríamos este hecho. Como el SSID usualmente tiene asumido disponer de un máximo de 32 caracteres, a menudo se copia en buffers de longitud fijada sin ninguna comprobación de la longitud actual. Recientemente, esto le jugó una mala pasada a Linux, cuando `ndiswrapper` (un software que permite que se usen drivers de dispositivos inalámbricos de Windows en Linux) se encontró vulnerable a un desbordamiento de búfer que podría ser explotado remotamente por un solo paquete inalámbrico.

RECURSOS

- [1] Estado del bug `e1000e`: <http://lwn.net/Articles/301251/>
- [2] Código del bug `e1000e`: <http://lwn.net/Articles/304105/>
- [3] Causa del bug `e1000e`: <http://ostatic.com/174457-blog/likely-cause-of-intel-e1000e-bug-disco-vered>