

**TAREA: 3\_6\_Comandos\_gcc\_gdb**

Manzo Torres Marcos

8° A Ing. Mecatrónica

Programación de sistemas embebidos Profesor: Carlos Moran Garabito

**GCC**

Lo primero de todo, GCC es un compilador portable —se ejecuta en la mayoría de las plataformas disponibles hoy, y puede producir salidas para muchos tipos de procesadores. Además de procesadores usados en ordenadores personales, también soporta microcontroladores, DSPs y CPUs de 64 bits.

GCC no es solo un compilador nativo —también puede *compilar cruzado* cualquier programa, produciendo ficheros ejecutables para un sistema diferente desde el que GCC está siendo usado. Esto permite compilar software para sistemas embebidos que no son capaces de ejecutar un compilador. GCC está escrito en C con un fuerte enfoque hacia la portabilidad, y puede compilarse a sí mismo, así puede ser adaptado a nuevos sistemas fácilmente. GCC tiene múltiples *frontends*, para parsear diferentes lenguajes. Los programas en cada lenguaje pueden ser compilados, o compilados de manera cruzada, para cualquier arquitectura. Por ejemplo, un programa en ADA puede ser compilado para un microcontrolador, o un programa en C para un supercomputador.

GCC tiene un diseño modular, permitiendo que el soporte para nuevos lenguajes y arquitecturas sea añadido. Añadir un nuevo front-end a GCC habilita el uso de este lenguaje en cualquier arquitectura y proporciona que estén disponibles facilidades (tales como librerías) en tiempo de ejecución. De manera similar, si se añade soporte para una nueva arquitectura éste se vuelve disponible para todos los lenguajes.

Para referirse exactamente a los compiladores de cada lenguaje, se habla de:

* GCC para C;
* G++ para C++;
* GCJ para Java;
* GNAT para Ada;
* Gfortran para Fortran;
* GPC para Pascal;
* GHDL para VHDL.

El compilador de C de GNU **gcc** es la aplicación que, dado un conjunto de ficheros de código C, genera un programa ejecutable. Como el resto de aplicaciones, se puede invocar desde el intérprete de comandos. Con la opción **--version** el compilador tan sólo muestra información sobre su versión. Prueba el siguiente comando en el terminal:

$ gcc --version

Como todo comando, **gcc** también tiene su página de manual en donde se explica su funcionamiento, así como las opciones con las que se puede modificar su comportamiento. Para ver la complejidad de esta herramienta mira esta página de manual con el comando:

$ man gcc

## De un fichero de código a un programa:

A continuación, ejecuta el comando para transformar este fichero de código en un programa:

$ gcc main\_es.c

Comprueba que en el directorio actual se ha creado el fichero a.out. El compilador está diseñado para que, si no se especifica el nombre del programa a crear, por defecto lo deja en a.out. Este fichero se puede ejecutar como cualquier otro comando, con la salvedad de que por estar ubicado en una carpeta del usuario y no en las que el sistema busca los programas, hay que ponerle el prefijo “./”, esto es:

$ ./a.out

La primera opción que vamos a utilizar es precisamente para poder escoger el nombre del ejecutable que produce el compilador. Esta opción es **-o** seguida (separado por un espacio) del nombre del fichero que queremos crear. Por ejemplo, si queremos llamar al programa main\_es, el comando a ejecutar es:

$ gcc -o main\_es main\_es.c

Tras ejecutar este comando comprueba que tienes **dos** ejecutables en el directorio actual. Borra el que ya no necesitas.

**GDB**

GDB (depurador de proyectos GNU) es un depurador de base de línea de comandos que es bueno para analizar programas en ejecución y de núcleo. De acuerdo con el manual del usuario, GDB admite C, C ++, D, Go, Objective-C, Fortran, Java, OpenCL C, Pascal, Rust, assembly, Modula-2 y Ada.

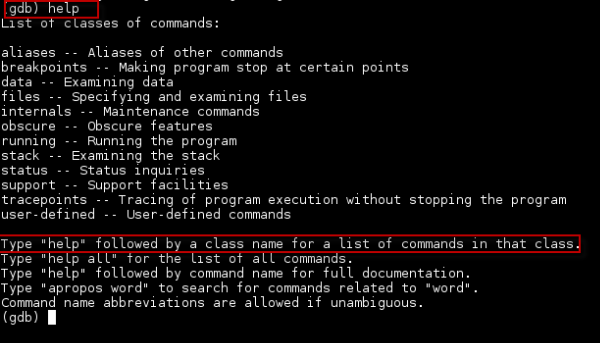
GDB tiene el mismo conjunto de funciones que la mayoría de los depuradores, pero es diferente de la mayoría de los que he usado, ya que todo se basa en los comandos de escritura en lugar de hacer clic en los elementos de la GUI. Algunas de estas características incluyen:

* Establecer puntos de ruptura
* Imprimiendo el valor de las variables.
* Establecer el valor de las variables para probar cosas.
* Viendo la pila

GDB ofrece la posibilidad de trazar y modificar la ejecución de un programa. El usuario puede controlar y alterar los valores de las [variables](https://es.wikipedia.org/wiki/Variable_(programaci%C3%B3n)) internas del programa.

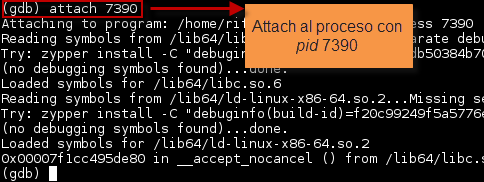
La diversidad de códigos maliciosos ha conllevado a que los analistas deban enfocarse en diversas plataformas para realizar los análisis correspondientes. En este caso, profundizaremos sobre GDB, el debugger por defecto presente en las distintas distribuciones Linux.

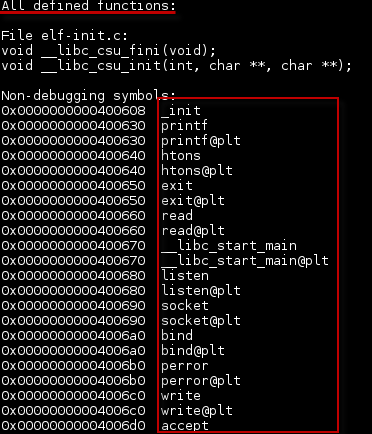
Anteriormente cubrimos temas referentes a herramientas para análisis dinámico de malware, en este caso profundizaremos aquello referido a análisis estático en Linux. Así como Ollydbg suele utilizarse en entornos Windows a la hora de analizar un malware que afecte a sistemas Linux será necesario utilizar un debugger como GDB. Para iniciar el debugger basta con ejecutar el comando gdb. De la misma manera, es posible ver las opciones a través del comando help. A continuación, puede observarse una imagen correspondiente a la ayuda:

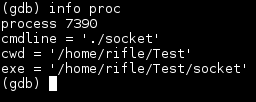
[](https://www.welivesecurity.com/wp-content/uploads/es-la/2013/08/Ayuda-Editada.png)

Si se desea obtener ayuda específica sobre uno de los comandos específicos, es posible adjuntar el comando help con el comando del cual se desea obtener más información.

**Funcionalidades**

[](https://www.welivesecurity.com/wp-content/uploads/es-la/2013/08/Proceso-de-attach-editada.png)Con gdb es posible ejecutar el archivo que se desea analizar dentro del contexto del propio debugger, o directamente realizar un attach de un proceso que ya está ejecutándose en memoria. En el caso de querer correr un determinado programa en el entorno del debugger, solo hace falta ejecutar gdb seguido de la aplicación que se desea analizar. De la misma manera, una vez que se ejecuta gdb, mediante el comando attach seguido del pid (id del proceso) es posible comenzar a analizar el archivo procesado. En primera instancia, es posible que el analista desee averiguar a qué funciones del sistema operativo utiliza la aplicación que se encuentra bajo análisis. Mediante el comando info functions es posible obsérvalo:

[](https://www.welivesecurity.com/wp-content/uploads/es-la/2013/08/info-functions-editado.png)

[](https://www.welivesecurity.com/wp-content/uploads/es-la/2013/08/info-proc.png)En caso de querer conocer desde donde se está ejecutando el proceso así como también obtener mayor información de cómo se inició el mismo, esto es posible mediante el comando info proc: