Suscribete a news sobre mis otros artículos y publicaciones en mi blog:

- https://twitter.com/yurichev
- https://www.facebook.com/dennis.yurichev.5

Servicios de ingeniería inversa

Intenté muchos trabajos en mi vida, pero, sorprendentemente (incluso para mí), El trabajo del que estoy más orgulloso es haber re-escrito grandes piezas de código compilado y llevarlos de vuelta en C / C ++. Este es un proceso extremadamente aburrido y lento, una vez me pasé más de un año reescribiendo una DLL de 100KB a C puro, y fue como haber hecho un trabajo a tiempo completo. Y esto también es costoso.

Se han agregado muchos trucos a este libro como resultado de este trabajo.

Supongo que este servicio podría ser de interes para aquellos que heredaron algún código compilado sin código fuente.

Debe ser el propietario legal del producto de software.

También podría intentar auditar el código (binario). Puedo tratar de encontrar vulnerabilidades en tu software antes de que otros lo hagan. Esto es como una prueba de penetración. Puedo intentar trabajar con código binario sin código fuente.

También debes ser el propietario legal del producto de software.

E-Mail: `dennis(@)yurichev.com`.

Por favor done

...a este proyecto para que pueda seguir trabajando en este libro y otros artículos:
https://yurichev.com/donate.html.

Atención: Encuesta de opinión

Tengo una idea para reemplazar todos los ejemplos de OllyDbg del libro con ejemplos obtenidos con algún otro depurador. No tengo nada en contra de OllyDbg, pero tiene una GUI y usa fuentes pequeñas, y las capturas de pantalla son algo inadecuadas para el libro.

Tal vez, ¿podría usar GDB, rada.re, WinDbg, o tal vez algún otro depurador de consola?

¿Qué piensas al respecto? ¿Debo dejar los ejemplos con OllyDbg, o ejemplos con radare estarían bien?

E-Mail: `dennis(@)yurichev.com`.

52 65 76 65 72 73 65
45 6e 67 69 6e 65 65 72 69 6e 67
66 6f 72 20 42 65 67 69 6e 6e 65 72 73

44 65 6e 6e 69 73 20 59 75 72 69 63 68 65 76

Ingeniería Inversa para Principiantes

Dennis Yurichev
<dennis(@)yurichev.com>

Versión del texto (14 de noviembre de 2017).

La última versión (así como las versiones en inglés y ruso) de este texto está disponible en beginners.re.

La portada fue hecha por Andy Nechaevsky: facebook.

Contenidos abreviados

Índice general

Prólogo

Existen muchos significados populares para el término «reverse engineering»: 1) La ingeniería inversa de software: la investigación de programas compilados; 2) El escaneo de estructuras 3D y la manipulación digital subsecuente requerida para duplicarlas; 3) La recreación de la estructura de un DBMS[1]. Este libro es acerca del primer significado.

Ejercicios y tareas

...fueron movidos al sitio web: http://challenges.re.

Sobre el autor

Dennis Yurichev es un reverser y programador experimentado. Puede ser contactado por email: **dennis(@)yurichev.com**.

Elogios para *Ingeniería Inversa para Principiantes*

- «Now that Dennis Yurichev has made this book free (libre), it is a contribution to the world of free knowledge and free education.» Richard M. Stallman,
- «It's very well done .. and for free .. amazing.»[2] Daniel Bilar, Siege Technologies, LLC.
- «... excellent and free»[3] Pete Finnigan,gurú de seguridad en Oracle RDBMS.
- «... [the] book is interesting, great job!» Michael Sikorski, autor de *Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software*.
- «... my compliments for the very nice tutorial!» Herbert Bos, catedrático de tiempo completo en la Vrije Universiteit Amsterdam, coautor de *Modern Operating Systems (4th Edition)*.
- «... It is amazing and unbelievable.» Luis Rocha, CISSP / ISSAP, Technical Manager, Network & Information Security at Verizon Business.
- «Thanks for the great work and your book.» Joris van de Vis, especialista en SAP Netweaver & Security .
- «... [a] reasonable intro to some of the techniques.»[4] Mike Stay, profesor en el Federal Law Enforcement Training Center, Georgia, US.
- «I love this book! I have several students reading it at the moment, [and] plan to use it in graduate course.»[5] Sergey Bratus , Research Assistant Professor en el Departamento de Ciencias de la Computación en Dartmouth College
- «Dennis @Yurichev has published an impressive (and free!) book on reverse engineering»[6] Tanel Poder, experto en afinación de rendimiento de Oracle RDBMS .

[1] Database Management Systems
[2] twitter.com/daniel_bilar/status/436578617221742593
[3] twitter.com/petefinnigan/status/400551705797869568
[4] reddit
[5] twitter.com/sergeybratus/status/505590326560833536
[6] twitter.com/TanelPoder/status/524668104065159169

- «This book is a kind of Wikipedia to beginners...» Archer, Chinese Translator, IT Security Researcher.
- «[A] first-class reference for people wanting to learn reverse engineering. And it's free for all.» Mikko Hyppönen, F-Secure.

Agradecimientos

Por contestar pacientemente a todas mis preguntas: Andrey «herm1t» Baranovich, Slava «Avid» Kazakov, SkullC0DEr.

Por enviarme notas acerca de errores e inexactitudes: Stanislav «Beaver» Bobrytskyy, Alexander Lysenko, Alexander «Solar Designer» Peslyak, Federico Ramondino, Mark Wilson, Xenia Galinskaya, Razikhova Meiramgul Kayratovna, Anatoly Prokofiev, Kostya Begunets, Valentin "netch" Nechayev, Shell Rocket, Zhu Ruijin, Changmin Heo, Vitor Vidal, Stijn Crevits, Jean-Gregoire Foulon[7], Ben L., Etienne Khan, Norbert Szetei[8], Marc Remy, Michael Hansen, Derk Barten, The Renaissance[9], Hugo Chan..

Por ayudarme de otras formas: Andrew Zubinski, Arnaud Patard (rtp en #debian-arm IRC), noshadow en #gcc IRC, Aliaksandr Autayeu, Mohsen Mostafa Jokar.

Por traducir el libro a Chino Simplificado: Antiy Labs (antiy.cn), Archer.

Por traducir el libro a Coreano: Byungho Min.

Spanish text placeholder: Cedric Sambre (AKA Midas).

Spanish text placeholder: Diego Boy, Luis Alberto Espinosa Calvo, Fernando Guida, Diogo Mussi, Patricio Galdames.

Spanish text placeholder: Thales Stevan de A. Gois, Diogo Mussi.

Spanish text placeholder: Federico Ramondino[10], Paolo Stivanin[11], twyK.

Spanish text placeholder: Florent Besnard[12], Marc Remy[13], Baudouin Landais, Téo Dacquet[14].

: Dennis Siekmeier[15], Julius Angres[16], Dirk Loser[17], Clemens Tamme.

TBT[18]: Kateryna Rozanova, Aleksander Mistewicz.

Por corrección de pruebasAlexander «Lstar» Chernenkiy, Vladimir Botov, Andrei Brazhuk, Mark "Logxen" Cooper, Yuan Jochen Kang, Mal Malakov, Lewis Porter, Jarle Thorsen, Hong Xie.

Vasil Kolev[19] realizó una gran cantidad de trabajo en corrección de pruebas y corrección de muchos errores.

Por las ilustraciones y el arte de la portada: Andy Nechaevsky.

Gracias a toda la gente en github.com que ha contribuido con notas y correcciones[20].

Muchos paquetes de LaTeX fueron utiliados: quiero agradecer también a sus autores.

Donadores

Aquellos que me apoyaron durante el tiempo que escribí una parte significativa del libro:

2 * Oleg Vygovsky (50+100 UAH), Daniel Bilar ($50), James Truscott ($4.5), Luis Rocha ($63), Joris van de Vis ($127), Richard S Shultz ($20), Jang Minchang ($20), Shade Atlas (5 AUD), Yao Xiao ($10), Pawel Szczur (40 CHF), Justin Simms ($20), Shawn the R0ck ($27), Ki Chan Ahn ($50), Triop AB (100 SEK), Ange Albertini (€10+50), Sergey Lukianov (300 RUR), Ludvig Gislason (200 SEK), Gérard Labadie (€40), Sergey

[7] https://github.com/pixjuan
[8] https://github.com/73696e65
[9] https://github.com/TheRenaissance
[10] https://github.com/pinkrab
[11] https://github.com/paolostivanin
[12] https://github.com/besnardf
[13] https://github.com/mremy
[14] https://github.com/T30rix
[15] https://github.com/DSiekmeier
[16] https://github.com/JAngres
[17] https://github.com/PolymathMonkey
[18] To be Translated. The presence of this acronym in this place means that the English version has some new/modified content which is to be translated and placed right here.
[19] https://vasil.ludost.net/
[20] https://github.com/dennis714/RE-for-beginners/graphs/contributors

Volchkov (10 AUD), Vankayala Vigneswararao ($50), Philippe Teuwen ($4), Martin Haeberli ($10), Victor Cazacov (€5), Tobias Sturzenegger (10 CHF), Sonny Thai ($15), Bayna AlZaabi ($75), Redfive B.V. (€25), Joona Oskari Heikkilä (€5), Marshall Bishop ($50), Nicolas Werner (€12), Jeremy Brown ($100), Alexandre Borges ($25), Vladimir Dikovski (€50), Jiarui Hong (100.00 SEK), Jim Di (500 RUR), Tan Vincent ($30), Sri Harsha Kandrakota (10 AUD), Pillay Harish (10 SGD), Timur Valiev (230 RUR), Carlos Garcia Prado (€10), Salikov Alexander (500 RUR), Oliver Whitehouse (30 GBP), Katy Moe ($14), Maxim Dyakonov ($3), Sebastian Aguilera (€20), Hans-Martin Münch (€15), Jarle Thorsen (100 NOK), Vitaly Osipov ($100), Yuri Romanov (1000 RUR), Aliaksandr Autayeu (€10), Tudor Azoitei ($40), Z0vsky (€10), Yu Dai ($10), Anonymous ($15), Vladislav Chelnokov (($25), Nenad Noveljic (($50).

¡Gracias a cada donante!

mini-FAQ

Q: ¿Por qué debería aprender lenguaje ensamblador hoy en día?

A: A menos que seas un desarrollador de SO[21], probablemente no necesitas programar en ensamblador—los compiladores modernos son mucho mejores generando optimizaciones que los humanos [22]. Además, los CPU[23]s modernos son dispositivos muy complejos y el conocimiento de ensamblador en realidad no ayuda a comprender su funcionamiento interno.

Una vez dicho eso, hay al menos dos áreas donde un buen entendimiento de ensamblador puede ser útil: Antes que nada, la investigación de seguridad/malware. También es una buena manera de obtener un mejor entendimiento de tu código compilado mientras es depurado.

Por lo tanto, este libro está dirigido a aquellos que desean comprender el lenguaje ensamblador en vez de codificar en él, razón por la cual contiene tantos ejemplos de código generado por compilador.

Q: Di click en un link dentro del documento PDF, ¿cómo regreso?

A: En Acrobat Reader, presiona Alt+FlechaIzquierda.

Q: ¿Puedo imprimir este libro / usarlo para enseñanza?

A: ¡Por supuesto! Por eso es que el libro está registrado bajo Creative Commons.

Q: ¿Cómo se consigue un trabajo en ingeniería inversa?

A: Existen threads de contratación que aparecen de vez en cuando en reddit, dedicados a reversing[24] (2016). Intenta buscando ahí.

Un thread en ocasiones relacionado con contrataciones puede ser encontrado en el subreddit «netsec»: 2016.

Q: Tengo una pregunta...

A: Envíamela por email (dennis(@)yurichev.com).

Acerca de la traducción al Coreano

En enero del 2015, la editorial Acorn (www.acornpub.co.kr) en Corea del Sur realizó una enorme cantidad de trabajo traduciendo y publicando mi libro (como era en agosto del 2014) en Coreano. Ahora se encuentra disponible en su sitio web.

El traductor es Byungho Min (twitter/tais9). El arte de la portada fue hecho por mi artístico amigo, Andy Nechaevsky facebook/andydinka. Ellos también poseen los derechos de autor de la traducción al coreano. Así que, si quieren tener un libro *real* en coreano en su estante y quieren apoyar mi trabajo, ya se encuentra disponible a la venta.

[21] Sistema Operativo
[22] Un buen texto acerca de este tema: [Agner Fog, *The microarchitecture of Intel, AMD and VIA CPUs*, (2016)]
[23] Central Processing Unit
[24] reddit.com/r/ReverseEngineering/

Capítulo 1

Patrones de código

1.1 Spanish text placeholder

Cuando el autor de este libro comenzó a aprender C y, más tarde, C++, él solía escribir pequeños trozos de código, compilarlos, y luego ver los resultados en lenguaje assembly. Esto lo hizo muy fácil para él entender lo que estaba pasando en el código que había escrito. [1]. Él lo hizo tantas veces que la relación entre el código C/C++ y lo que el compilador producido se imprimió profundamente en su mente. És fácil imaginar al instante un esbozo de la aparencia y función del código C. Quizás esta técnica podría ser útil para otra persona.

En ciertas partes, se han empleado aquí compiladores muy antiguas, con el fin de obtener lo mas corta (o simple) posible snippet. TBT.

Ejercicios

Cuando el autor de este libro estudió la lenguaje assembly, también con frecuencia compilaba pequeñas funciones en C, y reescribia gradualmente en assembly, tratando de hacer el código lo más pequeño posible. Probablemente no vale la pena hacer esto en escenarios reales actualmente, porque es dificil competir con los compiladores modernos en términos de eficiencia. Es, sin embargo, una muy buena manera de obtener una mejor compreensión de la assembly. Siéntase libre, por lo tanto, para tomar cualquier código de este libro y tratar de hacerlo más pequeño. Sin embargo, no se olvide de probar lo que has escrito.

Níveles de optimización y la información de depuración

El código fuente puede ser compilado por diferentes compiladores com varios niveles de optimización. Un compilador típico tiene alredor de tres de esos niveles, donde el nivel cero significa desactivar la optimización. La optimización también puede dirigirse hacia el tamaño del código o la velocidad de código. Un compilador sin optimización es más rápido y produce código más inteligible (aunque más grande), mientras un compilador con optimización es más lento y trata de producir un código que corre más rápido (pero no necesariamente más compacto). Además de los niveles y dirección de la otimización, el compilador puede incluir informaciones de depuración en el archivo resultante, produciendo así código para fácil depuración. Una de los características importantes del código de ´debug' és que puede contener enlaces entre cada línea del código fuente y las direcciones de código de máquina respectivos. Compiladores con optimización, por otro lado, tienden a producir una salida donde líneas enteras de código fuente pueden ser optimizados al punto de ser eliminados y por consiguiente no estar presentes en el código de máquina resultante. Ingenieros Inversos pueden encontrar ambas versiones, simplesmente porque alguns desarrolladores activan los flags de optimización del compilador, y otros no activan. Debido a esto, vamos a tratar de trabajar con ejemplos de ambas versiones de debug y release del código resaltado en este libro, cuando sea posible.

[1] De hecho, todavia lo hace cuando no puede entender lo que hace una determinada pieza de código.

1.2 Algunos conceptos basicos

1.2.1 Una breve introducción a la CPU

La CPU es el dispositivo que ejecuta el código de máquina que constituye un programa.

Un breve glosario:

Instrucción : Una primitiva CPU comando. Los ejemplos más simples incluyen: mover datos entre registros, trabajar con la memoria, operaciones aritméticas primitivas. Como regla general, cada CPU tiene su proprio conjunto de instrucciones (ISA[2]).

Spanish text placeholder : Código que la CPU procesa directamente. Cada instrucción generalmente se codifica por vários bytes.

Lenguaje assembly : Código mnemónico y algunas extensiones como macros que destinados a hacer la vida del programador más fácil.

Registros de la CPU : Cada CPU tiene un conjunto fijo de registros de propósito general (GPR[3]). ≈ 8 Spanish text placeholder x86, ≈ 16 Spanish text placeholder x86-64, ≈ 16 Spanish text placeholder ARM. La forma más fácil de entender un registro es pensar en ello como una variable temporal sin tipo. Imagine si estuviera trabajando con una LP[4] de alto nivel y sólo podría utilizar ocho variables de 32-bit (o de 64-bit). Sin embargo mucho se puede hacer usando sólo estos!

Uno podría perguntarse por qué es necessário que haya diferencia entre el código de la máquina y una lenguaje de programación de alto nivel. La respuesta está en el hecho de que los seres humanos y CPUs no son iguales—És mucho más fácil para los humanos utilizar un LP de alto nivel como C/C++, Java, Python, etc., pero és más fácil para una CPU utilizar un nivel mucho más bajo de abstración. Tal vez sería posible inventar una CPU que podría ejecutar código de LP de alto nivel, pero sería muchas veces más compleja que las CPUs que conocemos hoy. En uma manera similar, es muy incómodo para los seres humanos escribir en lenguaje assembly, debido a que es tan bajo nivel y difícil escribir sin hacer una gran cantidade de errores molestos. El programa que convierte el código de LP de alto nivel en assembly se llama *compiler*.

Algunas palabras sobre diferentes ISAs

El ISA x86 siempre ha tenido opcodes de tamaño variable, de modo que cuanco llegó la era de 64-bit, las extensiones x64 no impactan el ISA de manera muy significativa. De hecho, el ISA x86 aún contiene una gran cantidade de instrucciones que primero aparecieron en CPU 8086 16-bit, pero aún se encuentran en las CPUs de hoy. ARM és una CPU RISC[5] diseñado con la idea de opcodes con tamaño constante, que tenía algunas ventajas en el pasado. En el principio, todas las instrucciones ARM fueron codificados en 4 bytes[6]. Esto actualmente se conoce como «ARM mode». Entonces se llegó a la conclusión que no era tan económico como se imaginó al princípio. En realidad, la mayoría de las instrucciones de CPU utilizados [7] en aplicaciones del mundo real pueden ser codificados utilizando menos información. Por lo tanto añadieron otra ISA, llamado Thumb, donde cada instrucción fue codificada en sólo 2 bytes. Esto se conoce como «Thumb mode». No obstante, no todas las instrucciones ARM pueden ser codificadas en apenas 2 bytes, entonces el conjunto de instrucciones Thumb es algo limitada. Es importante destacar que el código compilado para el modo ARM y para el modo Thumb pueden, por supuesto, coexistir dentro de un solo programa. Los creadores de ARM concluyeron que se podría extender el Thumb, dando origem al Thumb-2, que apareció en el ARMv7. Thumb-2 sigue utilizando instrucciones de 2 bytes, pero tiene algunas nuevas instrucciones que tienen el tamaño de 4 bytes. Hay una idea errónea de que Thumb-2 es una mezcla de ARM y Thumb. Esto es incorrecto. Más bien, se extendió Thumb-2 para apoyar plenamente todas las características de processador por lo que podría competir con el modo ARM — un objetivo que se logró con claridad, ya que la mayoria de aplicacciones para iPod/iPhone/iPad son compmilados para el conjunto de instrucciones del Thumb-2 (la verdade es, en gran parte debido al hecho de que Xcode hace esto por defecto). Más tarde, el ARM 64-bit salió. Este ISA tiene opcodes de 4 bytes, y descarta la necesidade de cualquier modo Thumb adicional. Pero, los requisitos de 64-bit afectaron la ISA, resultando en ahora tenermos tres conjuntos de instrucciones ARM: ARM mode, Thumb mode (incluyendo Thumb-2)

[2] Instruction Set Architecture
[3] General Purpose Registers
[4] Lenguaje de Programación
[5] Reduced Instruction Set Computing
[6] Dicho sea de paso, las instrucciones de longitud fija son muy útiles porque se puede calcular la dirección de instrucción siguiente (o anterior) sin esfuerzo. Esta característica se discutirá en la sección de el operador switch() (**??** on page ??).
[7] Son estos MOV/PUSH/CALL/Jcc

y ARM64. Estos ISAs se intersectan parcialmente, pero puede ser más bien decir que son ISAs diferentes, en lugar de variaciones de lo mismo. Por lo tanto, nos gustaría intentar añadir fragmentos de código de los tres ISAs del ARM en este libro. Hay, por cierto, muchos otros RISC ISAs con opcodes de tamaño fijo de 32-bit, tales como MIPS, PowerPC Spanish text placeholder Alpha AXP.

1.2.2 Sistemas Numéricos

Los seres humanos se han acostumbrado a usar un sistema de numérico decimal, probablemente porque casi todos tenemos 10 dedos. Sin embargo, el número «10» no tiene un significado especial en ciencias y en matemáticas. El sistema de numérico natural en electrónica digital es el binario: 0 es por la ausencia de corriente en un cable y 1 por su presencia.

Si un sistema numérico tiene 10 dígitos, esto significa que un *radix* (o *base*) de 10. El sistema numérico binario tiene un *radix* de 2.

Cosas importantes para recordar:

Un *número* es un número tradicional, mientras que un *dígito* es un término empleado por los sistemas de escritura, y que corresponde a un carácter

2) El valor de un número no cambia cuando este se convierte a otro radix; solo su notación de escritura cambia (y por lo tanto la forma de representarlo en RAM[8]).

1.2.3 Conversión de una Base a Otra

La notación posicional se usa en casi todos los sistemas numéricos. Esto significa que un dígito tiene un peso relativo dependiendo del lugar donde este se ubica dentro del número. Si colocaramos un 2 en el lugar más a la derecha, su valor seria 2, en cambio si se colocara justo al lado izquierdo del dígito mas a la derecha, tendria el valor 20.

¿Qué significa 1234?

$10^3 \cdot 1 + 10^2 \cdot 2 + 10^1 \cdot 3 + 1 \cdot 4 = 1234$ o $1000 \cdot 1 + 100 \cdot 2 + 10 \cdot 3 + 4 = 1234$

Es la misma historia para los números binarios, pero la base es 2 en vez de 10. ¿Qué significa 0b101011?

$2^5 \cdot 1 + 2^4 \cdot 0 + 2^3 \cdot 1 + 2^2 \cdot 0 + 2^1 \cdot 1 + 2^0 \cdot 1 = 43$ o $32 \cdot 1 + 16 \cdot 0 + 8 \cdot 1 + 4 \cdot 0 + 2 \cdot 1 + 1 = 43$

Existe una notación no posicional, como es el sistema de numeración romano. [9]. Quizás, la humanidad cambió a la notación posicional porque le resultaba más fácil realizar operaciones básicas (adición, multiplicación, etc.) a mano sobre un papel.

Los números binarios se pueden sumar, restar, etc., de la misma manera como te enseñaron en la escuela, pero solo hay 2 dígitos disponibles.

Los números binarios son voluminosos cuando se representan en un código fuente y en un volcado, por lo que es aquí en donde el sistema numerico hexadecimal resulta ser de utilidad. Una base hexadecimal usa los dígitos 0..9 y también 6 caracteres latinos: A..F. Cada dígito hexadecimal toma 4 bits o 4 dígitos binarios, por lo que es muy fácil convertir un número binario a hexadecimal y viceversa, incluso mentalmente.

hexadecimal	binario	decimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
A	1010	10
B	1011	11
C	1100	12

[8] Random-Access Memory

[9] Acerca de la evolución del sistema numérico, consulte [Donald E. Knuth, *The Art of Computer Programming*, Volume 2, 3rd ed., (1997), 195-213.]

D	1101	13
E	1110	14
F	1111	15

¿Cómo se puede saber qué base se está utilizando en un ejemplo específico?

Los números decimales generalmente se escriben tal como son, es decir, 1234. Algunos ensambladores permiten agregar un identificador al número expresado en base decimal, en los que el número se escribiría con un sufijo ”d”: 1234d.

A veces, a un número binario se le antepone un prefijo ”0b”: 0b100110111 (GCC[10] tiene una extensión no estándar de idioma para esto [11]). También hay otra forma: usar un sufijo ”b”, por ejemplo: 100110111b. Este libro usará el prefijo ”0b” consistentemente para todos los números binarios.

A un número hexadecimal se le antepone un prefijo ”0x” en C/C++ y en otros **LP!**[12]s: 0x1234ABCD. Alternativamente, se les da un sufijo ”h”: 1234ABCDh. Esta es una forma común de representarlos en ensambladores y depuradores. En esta convención, si el número se inicia con un dígito latino (A..F), al inicio se le agrega un 0: 0ABCDEFh. También existio una convención muy popular en la era de las computadoras caseras de 8 bits, que usaba el prefijo $, como en $ABCD. A lo largo del libro intentaremos mantener el prefijo ”0x” para los números hexadecimales.

¿Debería aprender a convertir números mentalmente? Una tabla de números hexadecimales de 1 dígito se puede memorizar fácilmente. En cuanto a números más grandes, probablemente no valga la pena atormentarse.

Quizás los números hexadecimales más visibles estén en URL[13] s. Esta es la forma en que los caracteres no latinos están codificados. Por ejemplo: `https://en.wiktionary.org/wiki/na%C3%AFvet%C3%A9` es la URL del artículo de Wiktionary sobre la palabra «naïveté».

Base Octal

Otro sistema numérico muy empleado en el pasado para la programación de computadoras fue el octal. En octal hay 8 dígitos (0..7), y cada uno está mapeado a 3 bits, por lo que resulta ser fácil la conversión de números de base octal a binario y tambien en sentido contrario. Ha sido reemplazado ampliamente por el sistema hexadecimal, pero, sorprendentemente, hay una utilidad *NIX, utilizada a menudo por muchas personas, que toma números octales como argumento: `chmod`.

Como muchos usuarios de *NIX lo saben, el argumento `chmod` puede tener un número de 3 dígitos. El primer dígito representa los derechos del propietario del archivo (leer, escribir y/o ejecutar), el segundo son los derechos del grupo al que pertenece el archivo y el tercero es para todo el resto del mundo. Cada dígito que `chmod` toma puede representarse en forma binaria:

decimal	binario	significado
7	111	**rwx**
6	110	**rw-**
5	101	**r-x**
4	100	**r--**
3	011	**-wx**
2	010	**-w-**
1	001	**--x**
0	000	**---**

Entonces, cada bit está asignado a un indicador/bandera: leer/escribir/ejecutar.

La importancia de `chmod` aquí es que el número entero como argumento se puede representar como número octal. Tomemos, por ejemplo, 644. Cuando ejecuta `chmod 644 archivo`, se establecen los permisos de lectura/escritura para el propietario, los permisos de lectura para el grupo y nuevamente, los

[10] GNU Compiler Collection
[11] https://gcc.gnu.org/onlinedocs/gcc/Binary-constants.html
[12] **LP!**
[13] Uniform Resource Locator

permisos de lectura para todos los demás. Si convertimos el número octal 644 a binario, sería `110100100`, o, en grupos de 3 bits, `110 100 100`.

Ahora vemos que cada triplete describe los permisos para el propietario/grupo/otros: para el primero es `rw-`, para el segundo es `r--` y para el tercero es `r--`.

El sistema numerico octal también fue popular en las computadoras antiguas como PDP-8, porque la palabra podía ser de 12, 24 o 36 bits, y estos números son divisibles por 3, por lo que el sistema octal era natural en ese entorno. Hoy en día, todas las computadoras populares emplean tamaños de palabra/dirección de 16, 32 o 64 bits, y estos números son todos divisibles por 4, por lo que el sistema hexadecimal es más natural aquí.

El sistema de numérico octal es compatible con todos los compiladores estándar de C/C++. Esto a veces trae algo de confusión, ya que los números octales están codificados con un cero antepuesto, por ejemplo, 0377 es 255. A veces, esto puede conllevar un error tipográfico ya que podría escribir ”09” en vez de 9, y el compilador informaría un error. GCC podría informar algo como esto:
`error: dígito inválido "9" en constante octal`.

Además, el sistema octal es bastante popular en Java. Cuando el IDA muestra cadenas de caracteres de Java no imprimibles, estos están codificados en el sistema octal en vez del hexadecimal. El decompilador JAD de Java se comporta de la misma manera.

Divisibilidad

Cuando ves un número decimal como 120, puedes deducir rápidamente que es divisible por 10, porque el último dígito es cero. De la misma manera, 123400 es divisible por 100, ya que los dos últimos dígitos son ceros.

Del mismo modo, el número hexadecimal 0x1230 es divisible por 0x10 (o 16), 0x123000 es divisible por 0x1000 (o 4096), etc.

El número binario 0b1000101000 es divisible por 0b1000 (8), etc.

Esta propiedad a menudo se puede utilizar para que nos demos cuenta rápidamente si el tamaño de algún bloque en memoria se rellenará con algún límite. Por ejemplo, las secciones en archivos PE[14] casi siempre se inician en direcciones que terminan en 3 ceros hexadecimales: 0x41000, 0x10001000, etc. La razón de esto es el hecho de que casi todas las secciones PE se rellenan con un límite de 0x1000 (4096) bytes.

La base y la Aritmética de Multi-precisión

La aritmética de múltiple precisión puede usar números enormes, y cada uno puede almacenarse en varios bytes. Por ejemplo, las claves RSA, tanto públicas como privadas, abarcan hasta 4096 bits, y tal vez incluso más.

En [Donald E. Knuth, *The Art of Computer Programming*, Volume 2, 3rd ed., (1997), 265] encontramos la siguiente idea: cuando se almacena un número de múltiple precisión en varios bytes, el número completo puede representarse en una base de $2^8 = 256$, y cada dígito va al byte correspondiente. Del mismo modo, si se almacena un número de múltiple precisión en varios valores enteros de 32 bits, cada dígito va a cada ranura o slot de 32 bits, y puede pensarse que este número esta almacenado en base 2^{32}.

Cómo pronunciar números no decimales

Los números en una base no decimal generalmente se pronuncian dígito a dígito: “ uno-cero-cero-uno-uno -... ”. Palabras como “ diez ” y “ miles ” generalmente no se pronuncian, para evitar confusiones con el sistema de base decimal.

Números de punto flotante

Para distinguir los números de coma flotante de los enteros, generalmente estos se escriben con “ .0 ” al final, como 0.0, 123.0, etc.

[14] Portable Executable

1.3 La función más simple

La función más simple posible es sin duda uno que simplemente devuelve un valor constante:

Ejemplo:

Listing 1.1: Spanish text placeholder

```
int f()
{
        return 123;
};
```

Vamos a compilar!

1.3.1 x86

Esto es lo que optimiza el GCC y los compiladores de MSVC los cuales se producen en la plataforma x86:

Listing 1.2: Con optimización GCC/MSVC (salida del ensamblador)

```
f:
        mov     eax, 123
        ret
```

Hay solo dos instrucciones: los primeros lugares el valor 123 en el registro `EAX`, que se utiliza por convención para almacenar el valor de retorno y el segundo es `RET`, que devuelve la ejecución al llamado.

La persona que llama tomar el resultado del registro `EAX`.

1.3.2 ARM

Hay algunas diferencias en la plataforma ARM:

Listing 1.3: Con optimización Keil 6/2013 (Modo ARM) ASM Output

```
f       PROC
        MOV      r0,#0x7b ; 123
        BX       lr
        ENDP
```

ARM utiliza el registro `R0` para el retorno de los resultados de las funciones, por lo que se copia 123 en `R0`.

La dirección que retorna no es guardada en la pila local de la ARM ISA, sino más bien en el registro de enlace, entonces la instrucción `BX LR` provoca que la ejecución de un salto a la dirección eficazmente retornando la ejecución a donde fue llamada(caller “nuestro llamador”).

Vale la pena señalar que `MOV` es un nombre engañoso para la instrucción tanto en x86 y ARM ISAs.

Los datos de hecho, no se *mueven*, sino que se *copian*.

1.3.3 MIPS

Hay dos convenciones de nomenclatura utilizadas en el mundo de MIPS al nombrar registros: por número (de $0 a $31) o por seudónimo ($V0, $A0, etc.).

El ensamblado de GCC produce una salida en listando los registros por numeros:

Listing 1.4: Con optimización GCC 4.4.5 (salida del ensamblador)

```
        j       $31
        li      $2,123                  # 0x7b
```

...Mientras que IDA[15] lo hace por sus seudónimos:

Listing 1.5: Con optimización GCC 4.4.5 (IDA)

```
jr      $ra
li      $v0, 0x7B
```

El $2 (o $V0) registro es usado para almacenar el valor devuelto por la función. `LI` significa "Load Immediate" ("carga inmediata") y es el MIPS equivalente a `MOV`.

La otra instrucción es el jump ("salto") que es (J o JR) el cual retorna la ejecución fluida para el llamado, da un salto a la dirección del registro $31 (o $RA).

Este es el registro análoga a LR[16] en ARM.

Es posible que se pregunte por qué posiciones de la instrucción de la carga (LI) y el salto instrucciones (J y JR) se intercambian.Esto es debido a una característica llamada RISC ("ranura de retardo rama").

La razón por la que esto sucede es una peculiaridad en la arquitectura RISC de algunas ISA y no es importante para nuestros propósitos –– sólo tenemos que recordar que en MIPS, la instrucción que sigue a un salto o instrucción de salto se ejecuta antes de la instrucción / rama salto en sí.

Como consecuencia, las instrucciones de ramificación siempre intercambiar lugares con la instrucción que debe ser ejecutado de antemano.

Una nota acerca de la instrucción MIPS nombres / Registro

Registros y nombre de instrucciones en el mundo de MIPS tradicionalmente se escriben en minúsculas. Sin embargo, en aras de la coherencia, que me quedo con el uso de letras mayúsculas, ya que es la convención seguida por el resto de las ISAs destacados aquí.

1.4 Prologo y epilogo de funciones

El prologo de una funcion es una secuencia de instrucciones al inicio de esta. Por lo general luce mas o menos como el siguiente fragmento de codigo:

```
push    ebp
mov     ebp, esp
sub     esp, X
```

Lo que estas instrucciones hacen es: guardan el valor en el registro `EBP`, establece el valor del registro `EBP` al valor del registro `ESP` y luego asigna espacio en la pila para variables locales.

El valor de `EBP` permanece igual durante el periodo de ejecucion de la funcion, y es usado para variables locales y acceso a los argumentos. Para los mismos fines uno puede usar `ESP`, pero como este cambia con el tiempo, este enfoque no es muy conveniente.

El epilogo de la funcion libera el espacio asignado en la pila, coloca el valor del registro `EBP` vuelta su estado inicial y retorna el control de flujo a la la funcion llamada:

```
mov     esp, ebp
pop     ebp
ret     0
```

Los prologos y epilogos de funciones usualmente son detectados por desensambladores para la delimitacion de funciones.

1.4.1 Recursión

Epilogos y prologos pueden afectar negativamente el rendimiento de la recursion. Mas acerca de la recursion en este libro: **??** on page ??.

[15] Desensamblador Interactivo y depurador desarrollado por Hex-Rays
[16] Link Register

1.5 scanf()

1.5.1 Ejercicio

- http://challenges.re/53

1.6 switch()/case/default

1.6.1

```
#include <stdio.h>

void f (int a)
{
    switch (a)
    {
    case 0: printf ("zero\n"); break;
    case 1: printf ("one\n"); break;
    case 2: printf ("two\n"); break;
    default: printf ("something unknown\n"); break;
    };
};

int main()
{
    f (2); // test
};
```

Conclusión

listado.**??**.

1.6.2 Ejercicios

Ejercicio #1

1.7 Bucles

1.7.1 Ejercicios

- http://challenges.re/54
- http://challenges.re/55
- http://challenges.re/56
- http://challenges.re/57

1.8 Spanish text placeholder

1.8.1 strlen()

```
int my_strlen (const char * str)
{
        const char *eos = str;

        while( *eos++ ) ;
```

```
        return( eos - str - 1 );
}

int main()
{
        // test
        return my_strlen("hello!");
};
```

ARM

1.9 Substituición de instrucciones aritméticas por otras

1.9.1 Ejercicio

- http://challenges.re/59

1.10 Matriz

1.10.1

```
#include <stdio.h>

int main()
{
    int a[20];
    int i;

    for (i=0; i<20; i++)
        a[i]=i*2;

    for (i=0; i<20; i++)
        printf ("a[%d]=%d\n", i, a[i]);

    return 0;
};
```

1.10.2

1.10.3 Ejercicios

- http://challenges.re/62
- http://challenges.re/63
- http://challenges.re/64
- http://challenges.re/65
- http://challenges.re/66

1.11 Estructuras

1.11.1 UNIX: struct tm

1.11.2

1.11.3 Ejercicios

- http://challenges.re/71
- http://challenges.re/72

1.12

1.12.1

Listing 1.6: DEQuellcode stammt aus der Wikipedia: http://go.yurichev.com/17364

```
 * and that int is 32 bits. */
float sqrt_approx(float z)
{
    int val_int = *(int*)&z; /* Same bits, but as an int */
    /*
     * To justify the following code, prove that
     *
     * ((((val_int / 2^m) - b) / 2) + b) * 2^m = ((val_int - 2^m) / 2) + ((b + 1) / 2) * 2^m)
     *
     * where
     *
     * b = exponent bias
     * m = number of mantissa bits
     *
     * .
     */

    val_int -= 1 << 23; /* Subtract 2^m. */
    val_int >>= 1; /* Divide by 2. */
    val_int += 1 << 29; /* Add ((b + 1) / 2) * 2^m. */

    return *(float*)&val_int; /* Interpret again as float */
}
```

1.13

1.13.1

Dominic Sweetman, *See MIPS Run, Second Edition*, (2010).

Capítulo 2

Fundamentos importantes

2.1 Representación de números con signo

Hay distintos métodos para representar números con signo[1], pero el «complemento a dos» es el más popular en las computadoras.

Aquí hay una tabla con los valores de algunos bytes:

binario	hexadecimal	sin signo	con signo
01111111	0x7f	127	127
01111110	0x7e	126	126
...			
00000110	0x6	6	6
00000101	0x5	5	5
00000100	0x4	4	4
00000011	0x3	3	3
00000010	0x2	2	2
00000001	0x1	1	1
00000000	0x0	0	0
11111111	0xff	255	-1
11111110	0xfe	254	-2
11111101	0xfd	253	-3
11111100	0xfc	252	-4
11111011	0xfb	251	-5
11111010	0xfa	250	-6
...			
10000010	0x82	130	-126
10000001	0x81	129	-127
10000000	0x80	128	-128

La diferencia entre números con signo y sin signo está en que si representamos `0xFFFFFFFE` y `0x00000002` sin signo, el primero (4294967294) es mayor que el segundo (2). Pero si representamos ambos con signo, el primero se vuelve -2, y es menor que el segundo (2). Esa es la razón por la que se tienen saltos condicionales (**??** on page ??) tanto para operaciones con signo (e.g. `JG`, `JL`) como sin signo (`JA`, `JB`).

Con el fin de la simplicidad, esto es lo que uno debe de saber:

- Los números pueden ser con signo y sin signo.
- Tipos con signo en C/C++:
  - `int64_t` (-9,223,372,036,854,775,808 .. 9,223,372,036,854,775,807) (- 9.2.. 9.2 billones) Spanish text placeholder `0x8000000000000000..0x7FFFFFFFFFFFFFFF`),
  - *int* (-2,147,483,648..2,147,483,647 (- 2.15.. 2.15Gb) Spanish text placeholder `0x80000000..0x7FFFFFFF`),
  - *char* (-128..127 Spanish text placeholder `0x80..0x7F`),
  - `ssize_t` .

  Sin signo:

  - `uint64_t` (0..18,446,744,073,709,551,615 (18 billones) Spanish text placeholder `0..0xFFFFFFFFFFFF`
  - `unsigned int` (0..4,294,967,295 (4.3Gb) Spanish text placeholder `0..0xFFFFFFFF`),
  - `unsigned char` (0..255 Spanish text placeholder `0..0xFF`),
  - `size_t` .
- En los tipos con signo, el signo se encuentra en el bit más significativo: 1 significa «menos», 0 significa «más».
- Promover a un tipo de dato más grande es sencillo: **??** on page ??.

[1] wikipedia

- La negación es simple: sólo invierte todos los bits y suma 1. Podemos recordar que un número con signo contrario se localiza en el lado opuesto a la misma distancia de cero. La suma de 1 es necesaria porque el cero se localiza en medio.
- Las operaciones de suma y resta funcionan bien para valores tanto con signo como sin signo. Sin embargo, para las operaciones de multiplicación y división x86 tiene instrucciones distintas: `IDIV` / `IMUL` para números con signo Spanish text placeholder `DIV` / `MUL` para números sin signo.
- Éstas son algunas otras instruccciones que funcionan con números con signo: `CBW/CWD/CWDE/CDQ/CDQE` (**??** on page ??), `MOVSX` (**??** on page ??), `SAR` (**??** on page ??).

2.1.1 AND/OR/XOR como MOV

`OR reg, 0xFFFFFFFF` establece todos los bits a 1, por lo tanto, no importa lo que estaba antes en el registro, este será establecido a -1. `OR reg, -1` es más corto que `MOV reg, -1`, así que MSVC usa el OR en vez del MOV.

por ejemplo **??** on page ??.

Así, `AND reg, 0` siempre resetea todos los bits, por lo tanto, actúa como un `MOV reg, 0`.

`XOR reg, reg`, resetea todos los bits, sin importar lo que estuviese antes en el registro, y también es quivalente a un `MOV reg, 0`.

2.2 Endianness

Endianness es una forma de representar valores en la memoria.

2.2.1 Big-endian

El valor `0x12345678` es representado en la memoria como:

dirección en memoria	valor del byte
+0	0x12
+1	0x34
+2	0x56
+3	0x78

Entre los CPU big-endian se encuentran Motorola 68k, IBM POWER.

2.2.2 Little-endian

El valor `0x12345678` es representado en la memoria como:

Dirección en memoria	valor del byte
+0	0x78
+1	0x56
+2	0x34
+3	0x12

Entre los CPU little-endian tenemos el Intel x86.

2.2.3 Ejemplo

Tomemos el MIPS big-endian para Linux instalado y listo en QEMU [2].

Y compilemos este ejemplo sencillo:

[2] Disponible para descargar aquí: http://go.yurichev.com/17008

```
#include <stdio.h>

int main()
{
        int v, i;

        v=123;

        printf ("%02X %02X %02X %02X\n",
                *(char*)&v,
                *(((char*)&v)+1),
                *(((char*)&v)+2),
                *(((char*)&v)+3));
};
```

Después de correrlo obtenemos:

```
root@debian-mips:~# ./a.out
00 00 00 7B
```

Eso es todo. 0x7B es 123 en decimal. En las arquitecturas little-endian, 7B es el primer byte (puedes checarlo en x86 o x86-64), pero aquí es el último, porque el byte más alto va primero.

Por eso hay distribuciones separadas de Linux para MIPS («mips» (big-endian) Spanish text placeholder «mipsel» (little-endian)). Es imposible que un binario compilado para un endianness trabaje en un SO con diferente endianness.

En este libro hay otro ejemplo de big-endianness en MIPS: **??** on page ??.

2.2.4 Bi-endian

CPUs que puede cambiar entre endianness son ARM, PowerPC, SPARC, MIPS, IA64[3], etc.

2.2.5 Convirtiendo datos

La instrucción `BSWAP` puede ser utilizada para conversiones.

Los paquetes de datos en redes TCP/IP utilizan convenciones big-endian, por eso un programa corriendo en una arquitectura little-endian tiene que convertir los valores.

Las funciones `htonl()` Spanish text placeholder `htons()` son usadas generalmente.

En TCP/IP, big-endian también es llamado «orden de bytes de red», mientras que el orden de bytes en la computadora se conoce como «orden de bytes de host». «orden de bytes de host» es little-endian en Intel x86 y otras arquitecturas little-endian, pero es big-endian en IBM POWER, asi que `htonl()` y `htons()` no cambian el orden de los bytes en ésta última.

2.3 Memoria

Existen 3 tipos principales de memoria:

- Memoria global AKA[4] «asignación estática de memoria». No hay necesidad de asignarla explícitamente, la asignación es realizada al declarar variables/arreglos globales. Estas variables globales residen en los segmentos de datos o de constantes. Están disponibles globalmente (por lo tanto, se consideran un anti-patrón). No son convenientes para buffers/arreglos porque deben tener un tamaño fijo. Los desbordamientos de buffer que occurren aquí usualmente sobreescriben variables o buffers que residen junto a ellos en memoria. En este libro hay un ejemplo: **??** on page ??.

[3] Intel Architecture 64 (Itanium)
[4] - (También Conocido Como)

- Pila AKA «asignación en pila». La asignación se realiza al declarar variables/arreglos dentro de una función. Son usualmente variables locales a la función. Algunas veces estas variables locales también estan disponibles para funciones descendientes (funciones llamadas, si aquel que la llama le pasa un apuntador a una de sus variables). La asignación y desasignación son muy rápidas, sólo necesita que SP[5] sea ajustado. Spanish text placeholder Los desbordamientos de buffer suelen reescribir estructuras importantes en la pila: 1.10.2 on page 9.
- Heap AKA «asignación dinámica de memoria». La asignación/desasignación es realizada llamando a `malloc()/free()` Spanish text placeholder `new/delete` Spanish text placeholder C++. Éste es el método más conveniente: el tamaño del bloque puede establecerse en tiempo de ejecución. Cambiar el tamaño es posible (usando `realloc()`), pero puede ser lento. Ésta es la forma más lenta de asignar memoria: el asignador de memoria debe suportar y actualizar todas las estructuras de control mientras se asigna y desasigna. Los desbordamientos de buffer suelen sobreescribir estas estructuras. Las asiganciones en el heap también son el origen de problemas de fuga de memoria: cada bloque de memoria tiene que ser desasgnado explícitamente, pero uno puede olvidarse de ello, o hacerlo de manera incorrecta. Otro problema es el «uso después de la liberación»—usar un bloque de memoria después de que `free()` ha sido llamado en él, lo cual es muy peligroso. Un ejemplo en este libro: **??** on page ??.

2.4 CPU

2.4.1 Predictores del saltos

Algunos compiladores modernos intentan deshacerse de las instrucciones de saltos condicionales. Ejemplos en este libro son: **??** on page ??, **??** on page ??, **??** on page ??.

Esto se debe a que el predictor de saltos no siempre es perfecto, por lo tanto los compiladores tratan de evitar los saltos condicionales, de ser posible.

Las instrucciones condicionales en ARM (como ADRcc) son una forma de hacerlo, otra es el conjunto de instrucciones x86 CMOVcc.

2.4.2 Dependencias de datos

Los CPUs modernos son capaces de ejecutar instrucciones de manera simultanea (OOE[6]), pero para poder lograrlo, los resultados de una instrucción en un grupo no debe influenciar la ejecución de otras. Como consecuencia, el compilador se esfuerza en hacer uso de instrucciones que tengan una influencia mínima en el estado del CPU.

Por eso la instrucción `LEA` es tan popular, porque no modifica las banderas del CPU, mientras que otras instrucciones aritméticas sí lo hacen.

2.5 Funciones hash

Un ejemplo muy sencillo es CRC32, un algoritmo que provee una «fuerte» suma de comprobación para propósitos de comprobación de integridad. Es imposible reestablecer el texto original a partir de su valor hash, tiene mucho menos información: la entrada puede ser larga, pero el resultado de CRC32 siempre está limitado a 32 bits. Pero CRC32 no es criptográficamente seguro: es sabido cómo alterar un texto de tal modo que el valor del hash CRC32 resultante sea el que necesitemos. Las funciones hash criptográficas están protegidas de esto.

Tales funciones son MD5, SHA1, etc., y son utilizadas ampliamente para obtener el hash de contraseñas de usuarios para almacenarlas en las bases de datos. De hecho, la base de datos de un foro en internet no puede contener las contraseñas de los usuarios (una base de datos robado puede comprometer las contraseñas de todos los usuarios) sino únicamente hashes (un cracker no puede recuperar las contraseñas). Además, el motor de un foro de internet no es consciente de tu contraseña, sólo debe

[5] stack pointer. SP/ESP/RSP en x86/x64. SP en ARM.
[6] Out-of-Order Execution

comprobar si su hash es el mismo que aquel almacenado en la base de datos, y darte acceso si concuerdan. Uno de los métodos de cracking de contraseñas más simple es tratar de obtener los hashes de todas las contraseñas posibles para ver cuál concuerda con el resultado que necesitamos. Otros métodos son mucho más complejos.

2.5.1 ¿Cómo trabajan las funciones de una vía?

Las funciones de una vía son funciones capces de transformar un valor en otro, a la vez que es imposible (o muy difícil revertirlo.) Algunas personas tienen dificultad entendiendo cómo puede ser esto posible. Consideremos una demostración simple.

Tenemos un vector de 10 números en el rango 0..9, cada una presente una sola vez, por ejemplo:

```
4 6 0 1 3 5 7 8 9 2
```

El algoritmo de la función de una vía más simple es:

- toma el número en la posición cero (4 en nuestro caso);
- toma el número en la primera posición (6 en nuestro caso);
- intercambia los números en las posiciones 4 y 6.

Marquemos los números en las posiciones 4 y 6:

```
4 6 0 1 3 5 7 8 9 2
      ^   ^
```

Intercambiémolos y tenemos el resultado:

```
4 6 0 1 7 5 3 8 9 2
```

Mientras vemos el resultado, incluso si conocemos el algoritmo, no podemos enumerar sin ambigüedad el conjunto inicial porque los primeros dos números puedieron haber sido 0 y/o 1, y puedieron haber participado en el proceso de intercambio.

Este ejemplo fue demasiado simplificado para efectos de desmostración. Las funciones de una vía reales pueden llegar a ser muy complejas.

Capítulo 3

Capítulo 4

4.1 Linux

4.2 Windows NT

4.2.1 Windows SEH

SEH

[Matt Pietrek, *A Crash Course on the Depths of Win32™ Structured Exception Handling*, (1997)][1], [Igor Skochinsky, *Compiler Internals: Exceptions and RTTI*, (2012)] [2].

[1] También disponible como http://go.yurichev.com/17293
[2] También disponible como http://go.yurichev.com/17294

Capítulo 5

Herramientas

5.1 Desensamblador

5.1.1 IDA

Una versión freeware anterior está disponible para descargar [1].

Cheatsheet de teclas de acceso rápido: **??**

5.2 Depurador

5.2.1 OllyDbg

Un depurador muy popular para win32 en modo usuario: ollydbg.de.

Cheatsheet de teclas de acceso rápido: **??**

5.2.2 GDB

No es muy popular entre reversers, aunqu es muy cómoda.

Algunos comandos: **??**.

5.2.3 tracer

El autor suele utilizar *tracer* [2] english link (recurso en inglés) en vez de un depurador.

El autor de estas líneas eventualmente dejó de utilizar un depurador, ya que lo único que necesita es hallar los argumentos de las funciones durante la ejecución, o el estado de los registros en algún punto. Cargar un depurador en cada ocasión es demasiado, y fue así como nació una utilería llamada *tracer*. Funciona a través de la línea de comandos, permitiendo interceptar la ejecucón de una función, colocar breakpoints en lugares arbitrarios, leer y cambiar el estado de los registros, etc.

Sin embargo, con fines de aprendizaje es altamente recomendable trazar el código manualmente en un depurador, observar cómo cambia el estado de los registros (e.g. los clásicos SoftICE, OllyDbg, WinDbg subrayan los registros modificado), de las banderas, de los datos, modificarlos, observar la reacción, etc.

[1] hex-rays.com/products/ida/support/download_freeware.shtml
[2] yurichev.com

5.3 Trazado de llamadas al sistema

strace / dtruss

Muestra cuáles llamadas al sistema (llamadas al sistema(**??**)) son llamadas por un proceso en este momento.

Por ejemplo:

```
# strace df -h

...

access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
open("/lib/i386-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\1\1\1\0\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0\220\232\1\0004\0\0\0"..., 512) = 512
fstat64(3, {st_mode=S_IFREG|0755, st_size=1770984, ...}) = 0
mmap2(NULL, 1780508, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0xb75b3000
```

Mac OS X tiene dtruss para hacer lo mismo.

Cygwin también tiene strace, para hasta donde se sabe, sólo trabaja con archivos .exe compilados para el mismo ambiente cygwin.

5.4 Decompiladores

Existe un solo decompilador a código C de alta calidad y disponible públicamente: Hex-Rays: hex-rays.com/product

5.5 Otras herramientas

- Microsoft Visual Studio Express[3]: La versión mínima y gratuita de Visual Studio, conveniente para experimentos sencillos. Algunas opciones útiles: **??**.
- Hiew[4]: para realizar modificaciones de código pequeñas en archivos binarios.
- binary grep: una pequeña utilería para buscar cualquier secuencia de bytes en un montón de archivos, incluyendo archivos no ejecutables: GitHub.

5.6

5.7

Pierre Capillon – Black-box cryptanalysis of home-made encryption algorithms: a practical case study.

[3] visualstudio.com/en-US/products/visual-studio-express-vs
[4] hiew.ru

Capítulo 6

Capítulo 7

Libros/blogs que merecen lectura

7.1 Libros

7.1.1 Reverse Engineering

- Eldad Eilam, *Reversing: Secrets of Reverse Engineering*, (2005)
- Bruce Dang, Alexandre Gazet, Elias Bachaalany, Sebastien Josse, *Practical Reverse Engineering: x86, x64, ARM, Windows Kernel, Reversing Tools, and Obfuscation*, (2014)
- Michael Sikorski, Andrew Honig, *Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software*, (2012)
- Chris Eagle, *IDA Pro Book*, (2011)

7.1.2 Windows

- Mark Russinovich, *Microsoft Windows Internals*

Blogs:

- Microsoft: Raymond Chen
- nynaeve.net

7.1.3 C/C++

- Brian W. Kernighan, Dennis M. Ritchie, *The C Programming Language*, 2ed, (1988)
- *ISO/IEC 9899:TC3 (C C99 standard)*, (2007)[1]
- Bjarne Stroustrup, *The C++ Programming Language, 4th Edition*, (2013)
- C++11 standard[2]
- Agner Fog, *Optimizing software in C++* (2015)[3]
- Marshall Cline, *C++ FAQ*[4]
- Dennis Yurichev, *C/C++ programming language notes*[5]
- JPL Institutional Coding Standard for the C Programming Language[6]
- Intel manuals[7]

[1] También disponible como http://go.yurichev.com/17274
[2] También disponible como http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2013/n3690.pdf.
[3] También disponible como http://agner.org/optimize/optimizing_cpp.pdf.
[4] También disponible como http://go.yurichev.com/17291
[5] También disponible como http://yurichev.com/C-book.html
[6] También disponible como https://yurichev.com/mirrors/C/JPL_Coding_Standard_C.pdf
[7] También disponible como http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html

- AMD manuals[8]
- Agner Fog, *The microarchitecture of Intel, AMD and VIA CPUs*, (2016)[9]
- Agner Fog, *Calling conventions* (2015)[10]
- [*Intel® 64 and IA-32 Architectures Optimization Reference Manual*, (2014)]
- [*Software Optimization Guide for AMD Family 16h Processors*, (2013)]

7.1.4 ARM

- Manuales de ARM[11]
- *ARM(R) Architecture Reference Manual, ARMv7-A and ARMv7-R edition*, (2012)
- [*ARM Architecture Reference Manual, ARMv8, for ARMv8-A architecture profile*, (2013)][12]
- Advanced RISC Machines Ltd, *The ARM Cookbook*, (1994)[13]

7.1.5 Java

[Tim Lindholm, Frank Yellin, Gilad Bracha, Alex Buckley, *The Java(R) Virtual Machine Specification / Java SE 7 Edition*] [14].

7.1.6 UNIX

Eric S. Raymond, *The Art of UNIX Programming*, (2003)

7.1.7 Criptografía

- Bruce Schneier, *Applied Cryptography*, (John Wiley & Sons, 1994)
- (Free) lvh, *Crypto 101*[15]
- (Free) Dan Boneh, Victor Shoup, *A Graduate Course in Applied Cryptography*[16].

7.2 Otros

Henry S. Warren, *Hacker's Delight*, (2002).

Existen dos excelentes subreddits relacionados con RE[17] en reddit.com: reddit.com/r/ReverseEngineering/ Spanish text placeholder reddit.com/r/remath (en los tópicos de la intersección de RE y matemáticas).

También hay una sección sobre RE en el sitio web de Stack Exchange:

reverseengineering.stackexchange.com.

En IRC hay un canal ##re en FreeNode[18].

[8] También disponible como http://developer.amd.com/resources/developer-guides-manuals/
[9] También disponible como http://agner.org/optimize/microarchitecture.pdf
[10] También disponible como http://www.agner.org/optimize/calling_conventions.pdf
[11] También disponible como http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.subset.architecture.reference/index.html
[12] También disponible como http://yurichev.com/mirrors/ARMv8-A_Architecture_Reference_Manual_(Issue_A.a).pdf
[13] También disponible como http://go.yurichev.com/17273
[14] También disponible como https://docs.oracle.com/javase/specs/jvms/se7/jvms7.pdf; http://docs.oracle.com/javase/specs/jvms/se7/html/
[15] También disponible como https://www.crypto101.io/
[16] También disponible como https://crypto.stanford.edu/~dabo/cryptobook/
[17] Reverse Engineering
[18] freenode.net

Spanish text placeholder

7.3 Spanish text placeholder

No dudes en enviar cualquier pregunta por email al autor: `<dennis(@)yurichev.com>`. ¿Tienes alguna sugerencia sobre nuevas cosas que puedan agregarse al libro? Por favor, no dudes en enviar correcciones (incluyendo gramática), etc.

El autor está trabajando arduamente en el libro, así que los números de página, de listado, etc., cambian con frecuencia. Por favor, no utilices como referencia números de página o de listado en tus emails. Existe un método mucho más simple: toma una impresión de pantalla de la página, subraya el lugar donde se encuentra el error en un editor de gráficos, y envíalo. De este modo será arreglado más rápido. Y si estás fmiliarizado con git y LaTeX puedes arreglar el error directo en el código fuente:

GitHub.

No te preocupes por escribirme acerca de errores insignificantes que encuentras, incluso si no te sientes seguro. Estoy escribiendo para principiantes, después de todo, por lo tanto la opinión de los principiantes y sus comentarios son cruciales para mi trabajo.

Acrónimos utilizados

SO Sistema Operativo vi

LP Lenguaje de Programación 2

PE Portable Executable 5

SP stack pointer. SP/ESP/RSP en x86/x64. SP en ARM. 15

LR Link Register 7

IDA Desensamblador Interactivo y depurador desarrollado por Hex-Rays 7

AKA - (También Conocido Como) 14

CPU Central Processing Unit vi

RISC Reduced Instruction Set Computing 2

DBMS Database Management Systems iv

ISA Instruction Set Architecture 2

RAM Random-Access Memory 3

GCC GNU Compiler Collection 4

IA64 Intel Architecture 64 (Itanium) 14

OOE Out-of-Order Execution 15

GPR General Purpose Registers 2

RE Reverse Engineering 21

TBT To be Translated. The presence of this acronym in this place means that the English version has some new/modified content which is to be translated and placed right here. v

URL Uniform Resource Locator 4