مهندسی معکوس برای مبتدیان

مقدمه
چندین معانی معروف از "مهندسی معکوس" وجود دارد : ۱) مهندسی معکوس نرم‌افزار : کاوش کردن برنامه‌های کامپایل شده; ۲) اسکن کردن ساختارهای 3D و دستکاری محصولات دیجیتالی پیشرو به منظور تکثیر و تولید آن ها; ۳) ساخت مجدد ساختار DBMS (سیستم مدیریت پایگاه داده). این کتاب در مورد معنی اول است.

مباحثی که به صورت عمیق مورد بحث قرار خواهند گرفت
.x86/x64, ARM/ARM64, MIPS, Java/JVM

مباحثی که به سرعت مورد بررسی قرار می‌گیرند
Itanium ، Oracle RDBM ، دانگل های محافظت از کپی ، LD_PRELOAD ، سرریز پشته، ELF 10 ، قالب فایل‌های قابل حمل و اجرایی win32 ، ساختار x86-64 ، بخش‌های مهم، فراخوان های سیستمی، TLS 11 ، کد مستقل از موقعیت (PIC 12) ، بهینه سازی پروفایل هدایت شده ، OpenMP ، C++ STL .SEH

تمرین‌ها و کارها
...همه آن‌ها به وب سایت جداگانه ای منتقل شده‌اند : http://challenges.re.

درباره نویسنده

دنیس پوریچو (Dennis Yurichev) یک برنامه نویس و مهندس معکوس با تجربه است. با او می‌توان از طریق ایمیل به آدرس **dennis@yurichev.com** یا اسکایپ **dennis.yurichev** ارتباط برقرار کرد.

تشکرها
برای صبر و حوصله به خاطر پاسخ دادن به تمام سؤالات من : آندری "herm1t" برانوویچ ، اسلاوا "Avid" کازیکو.
برای ارسال یادداشت‌ها در مورد اشتباهات و عدم دقت برای من : استانیسلاو "Beaver" بابریسکای ، الکساندر لیسنکو، شل راکت ، ژو رویجین ، چنگمین هور.
برای کمک به من به روش های دیگر : اندرو زوبینسکی ، آرنود پاتاد (با نام مستعار rtp در کانال #debian-arm در شبکه IRC) ، الیاکساندر اوتای.
برای ترجمه کتاب به زبان چینی ساده : شی آن کای.
برای ترجمه کتاب به زبان کره ای : بون های مین.
برای غلط گیری : الکساندر "Lstar" چرننکی ، ولادیمیر باتاو ، آندری بریزهیک، مارک "Logxen" کوپر، یوآن جوکن کانگ ، مال مالاکاو ، لوئیس پورتر.
"وسیل کویلو" مقدار زیادی کار را در زمینه غلط گیری و تصحیح اشتباهات فراوان انجام داد.

برای تصاویر و طرح جلد : اندی نیچایوسکی.
همچنین با تشکر از تمامی مردم در github.com که در یادداشت‌ها و اصلاحات به من کمک کردند.
من از بسیاری از بسته های LATEX استفاده کردم : همچنین می‌خواهم از سازندگان آن‌ها تشکر کنم.

0.0.1 کمک مالی

همانطور که معلوم است، نوشتن به صورت فنی زمان و تلاش زیادی را می گیرد.
این کتاب رایگان است و به صورت آزادانه و به شکل کد منبع (LaTeX) در دسترس است و باید برای همیشه به همین صورت باقی بماند.
همچنین این کتاب شامل هیچگونه تبلیغی نیست.
طرح فعلی من برای این کتاب اضافه کردن مقدار زیادی از اطلاعات در مورد موارد زیر است :

- Objective-C
- Visual Basic
- ترفندهای ضد اشکال زدایی
- اشکل زدایی هسته Windows NT
- .NET
- Oracle RDBMS

اگر می‌خواهید که من به نوشتن در مورد تمام این موضوعات ادامه بدهم، ممکن است که کمک مالی را در نظر بگیرید.
راه‌های اهداء کمک می‌توانند در آدرس beginners.re یافت شوند.
نام تمام کمک کننده ها در این کتاب گنجانده خواهد شد! واقعیت این است که کمک کنندگان مالی این حق را دارند که از من بخواهند موارد موجود در طرحم برای نوشتن را مجدداً مرتب سازی کنم.

کمک کنندگان مالی

25 * anonymous, 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 (e10+50), Sergey Lukianov (300 RUR),
Ludvig Gislason (200 SEK), Gérard Labadie (e40), Sergey Volchkov (10 AUD), Vankayala Vigneswararao ($50), Philippe Teuwen
($4), Martin Haeberli ($10), Victor Cazacov (e5), Tobias Sturzenegger (10 CHF), Sonny Thai ($15), Bayna AlZaabi ($75), Redfive
B.V. (e25), Joona Oskari Heikkilä (e5), Marshall Bishop ($50), Nicolas Werner (e12), Jeremy Brown ($100), Alexandre Borges
($25), Vladimir Dikovski (e50), 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 (e10), Salikov Alexander (500 RUR), Oliver Whitehouse
(30 GBP), Katy Moe ($14), Maxim Dyakonov ($3), Sebastian Aguilera (e20), Hans-Martin Münch (e15), Jarle Thorsen (100
NOK), Vitaly Osipov ($100), Yuri Romanov (1000 RUR), Aliaksandr Autayeu (e10), Tudor Azoitei ($40), Z0vsky (e10)

پرسش و پاسخ کوتاه

پرسش : چرا باید این روزها زبان اسمبلی را یاد بگیریم؟
پاسخ : تنها در صورتی نیاز به این کار ندارید که یک توسعه‌دهنده سیستم عامل باشید که احتمالاً نیازی به کد نویسی به زبان اسمبلی ندارید – کامپایلرهای مدرن بهینه سازی را نسبت به انسان بسیار بهتر انجام می دهند. همچنین پردازنده های امروزی دستگاه‌های بسیار پیچیده‌ای هستند و دانش اسمبلی واقعاً برای درک اجزای داخلی آن‌ها کمکی نمی کند.می شود گفت حداقل دو حوزه وجود دارند که درک خوب از زبان اسمبلی در آن‌ها می‌تواند مفید باشد : اولین و مهمترین آن پژوهش در زمینه امنیت/نرم افزارهای مخرب است. همچنین یک راه خوب برای به دست آوردن درک بهتری از کدهای کامپایل شده در هنگام اشکال زدایی نیز همین است. در نتیجه این کتاب برای کسانی است که می‌خواهند زبان اسمبلی را درک کنند نه کد نویسی در آن را و به همین دلیل مثال های متعددی از خروجی کامپایلر در این کتاب موجود است.

پرسش : من بر روی یک لینک در داخل سند PDF کلیک کردم، چگونه می‌توانم به عقب برگردم؟
پاسخ : در Adobe Acrobat Reader از کلید های Alt+Left Arrow استفاده کنید.

پرسش : کتاب شما حجیم است! آیا نمونه مختصری از آن وجود دارد؟
پاسخ: نسخه سبک تر آن در آدرس http://beginners.re/#lite وجود دارد.

پرسش : مطمئن نیستم که اگر برای یادگیری مهندسی معکوس تلاش کنم می‌توانم آن را یادبگیرم یا نه.
پاسخ : شاید زمان متوسط برای آشنا شدن با محتویات نسخه کوتاه ۱ یا ۲ ماه باشد.

پرسش : آیا امکان دارد که این کتاب را چاپ کنم؟ آیا می‌توانم از آن برای آموزش استفاده کنم؟
پاسخ : البته! به همین دلیل است که این کتاب تحت مجوز Creative Commons است. همچنین در صورت تمایل می‌توانید نسخه مربوط به خود را بسازید – برای یافتن اطلاعات بیشتر https://github.com/dennis714/RE-for-beginners/blob/master/HACKING.md را بخوانید.

پرسش : می‌خواهم کتاب شما را به زبان‌های دیگر ترجمه کنم.
پاسخ : به آدرس https://github.com/dennis714/RE-for-beginners/blob/master/Translation.md بروید.

پرسش : چگونه می‌توانم یک شغل در مورد مهندسی معکوس پیدا کنم؟
پاسخ : گهگاهی در reddit موضوعاتی در رابطه با استخدام به مهندسی معکوس اختصاص داده شده است (آدرس های http://go.yurichev.com/17333 و http://go.yurichev.com/17334 را ببینید).سعی کنید به آن‌ها نگاهی کنید. موضوعات تا حدودی مربوط به استخدام را می‌توان در netsec مربوط به reddit پیدا کرد : http://go.yurichev.com/17335

پرسش : من یک سؤال دارم...
پاسخ : آن را از طریق ایمیل به من بفرستید (dennis@yurichev.com) یا سؤال خود را در فروم وب سایت من مطرح کنید : forum.yurichev.com

تعریف و تمجید از کتاب مهندسی معکوس برای مبتدیان

- “این کتاب بسیار خوب نوشته شده و رایگان است...شگفت انگیزه”. دنیل بیلار از Siege Technologies, LLC.
- “عالی و رایگان”. پیت فینیگن متخصص امنیت Oracle RDBMS.
- “این کتاب جالب است و کار بزرگی است”. مایکل سیکورسکی نویسنده کتاب Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software.
- “تمجید از طرف من به خاطر آموزش بسیار خوب...”. هربرت باس، استاد تمام در دانشگاه Vrije آمستردام و یکی از نویسندگان کتاب Modern Operating Systems (4th Edition).
- “این کتاب شگفت انگیز و باورنکردنی است...”. لوئیس روچا دارنده مدرک CISSP / ISSAP ، مدیر فنی، شبکه و امنیت اطلاعات در شرکت Verizon.
- “تشکر به خاطر این کار بزرگ و کتاب شما”. جوریس ون دی ویس، SAP Netweaver و متخصص امنیت.
- “مقدمه ای معقول و منطقی برای برخی از تکنیک های دیگر...”. مایک استی، معلم در مرکز آموزش اجرای قانون فدرال در ایالت جورجیا.
- “این کتاب را دوست دارم! تعدادی دانشجو دارم که در حال حاضر آن را خوانده‌اند و برای استفاده از آن در دوره فوق لیسانس برنامه‌ریزی کرده اند”. سرگی براتس، استادیار تحقیقات در دپارتمان علوم کامپیوتر در کالج دارتموث.
- “دنیس یوریچو یک کتاب قابل توجه (و رایگان!) درباره مهندسی معکوس منتشر کرده است”. تانل پادر، متخصص تنظیم عمل‌کرد Oracle RDBMS.

درباره ترجمه کره ای

در ژانویه سال 2015 ، شرکت چاپ و نشر Acorn (بلوط) (www.acornpub.co.kr) در کره جنوبی برای ترجمه و چاپ کتاب من به زبان کره ای کار بزرگی انجام داد (که در ماه اوت سال 2014 بود).
این کتاب در حال حاضر در وب سایت آن‌ها به آدرس http://www.acornpub.co.kr/book/reversing-for-beginners موجود است.
مترجم این کتاب Byungho Min است (https://twitter.com/tais9).
جلد کتاب توسط دوست هنرمندم Andy Nechaevsky انجام شد. (https://www.facebook.com/andydinka)
آن‌ها همچنین از حق تکثیر ترجمه کره ای نگهداری می کنند.
بنابراین اگر می‌خواهید که یک کتاب واقعی به زبان کره ای در کتابخانه خود داشته باشید و از کار من نیز حمایت کنید، این کتاب در حال حاضر برای خرید در دسترس قرار دارد.

بخش اول
الگوهای کد

هنگامی که نویسنده این کتاب برای اولین بار شروع به یادگیری زبان C و بعد از آن C++ کرد ، او تکه‌ای کوچی از کد را نوشت، آن را کامپایل کرد و سپس به خروجی زبان اسمبلی نگاه کرد. این کار باعث شد که درک اینکه در کدی که او نوشته است چه می گذرد برایش بسیار آسان شود. او این کار را چندین بار انجام داد تا ارتباط بین کدهای C/C++ و آنچه که کامپایلر تولید می‌کند عمیقاً در ذهنش هک شود. تصور فوری و کلی از محتوای کدهای C و عمل‌کرد آن‌ها برایش آسان بود. شاید این روش بتواند برای افراد دیگر نیز مفید باشد.
گاهی اوقات در اینجا به منظور رسیدن به کوتاه ترین (یا ساده ترین) تکه کد ممکن از کامپایلرهای قدیمی استفاده می‌شود.

هنگامی که نویسنده این کتاب زبان اسمبلی را مطالعه می کرد، او اغلب توابع کوچک به زبان C را کامپایل کرده و سپس آن‌ها را به تدریج به زبان اسمبلی بازنویسی می‌کرد و تلاش می‌کرد که کدهای خود را تا آنجا که ممکن است کوتاه کند. امروزه احتمالاً این کار ارزش انجام در دنیای واقعی را ندارد چون از نظر بهره وری رقابت آن با کامپایلرهای مدرن سخت است. با این حال، این یک راه بسیار خوب برای به دست آوردن درک بهتری از زبان اسمبلی است. بنابراین، به راحتی می‌توانید هر کد به زبان اسمبلی که می‌خواهید را از این کتاب برداشته و سعی کنید که آن را کوتاه‌تر کنید. اما آزمایش کردن آنچه که نوشته‌اید را فراموش نکنید.

سطوح بهینه سازی و اشکال زدایی اطلاعات
کد منبع (Source code) را می‌توان با کامپایلرهای متفاوت با سطوح مختلف بهینه سازی کامپایل کرد. معمولا کامپایلرها در حدود سه سطح دارند که سطح صفر به معنی غیرفعال کردن بهینه سازی است. همچنین بهینه سازی می‌تواند حجم کد یا سرعت کد را هدف قرار دهد.
کامپایلر غیر بهینه سازی شده سریعتر است و کدهای قابل فهم تری (البته طولانی) را تولید می کند، در حالی که یک کامپایلر بهینه سازی شده کندتر است و تلاش می‌کند کدهایی را تولید کند که سریع‌تر اجرا می‌شوند (اما لزوماً فشرده تر نیست).
علاوه بر سطوح بهینه سازی و هدایت، کامپایلر می‌تواند شامل برخی از اطلاعات اشکال زدایی در فایل نتیجه باشد که در نتیجه کد تولید شده را برای اشکال زدایی آسان می کند.
یکی از ویژگی‌های مهم اشکال زدایی کد این است که ممکن است شامل ارتباطی بین هر خط از کد منبع و آدرس کد ماشین مربوطه باشد. از طرف دیگر، بهینه سازی کامپایلرها تمایل به تولید خروجی دارد که در آن تمام خطوط کد منبع می‌توانند به طور پیوسته بهینه سازی شوند و حتی کد ماشین نیز در نتیجه حاصل از آن وجود نداشته باشد.
مهندسان معکوس می‌توانند با هر دو نسخه مواجه شوند زیرا برخی از توسعه دهندگان پرچم های مربوط به بهینه سازی کامپایلرها را روشن کرده و برخی دیگر این کار را نمی کنند. به همین دلیل ما سعی خواهیم کرد که در مثال‌های موجود در این کتاب تا جایی که امکان دارد هم بر روی نسخه اشکال زدایی و هم بر روی نسخه آزاد کار کنیم.

فصل 1

یک مقدمه کوتاه برای CPU

CPU دستگاهی است که کد ماشین که یک برنامه شامل آن است را اجرا می کند.

واژه نامه کوتاه :

دستور (Instruction) : یک دستور ابتدایی CPU است. ساده‌ترین نمونه ها عبارت اند از : انتقال داده بین ثبات ها، کار با حافظه و محاسبات اولیه. به عنوان یک قانون، هر CPU مجموعه دستورهای معماری (ISA) مربوط به خود را دارد.

کد ماشین (Machine code) : کدی که CPU به طور مستقیم آن را پردازش می کند. هر دستور معمولاً با چند بایت کد گذاری می شود.

زبان اسمبلی (Assembly language) : کدهای مربوط به حافظه و برخی از الحاقات مانند ماکروها که برای آسان‌تر کردن کار برنامه نویس در نظر گرفته شده اند.

ثبات CPU (CPU register) : هر پردازنده مجموعه‌ای ثابت از ثبات های همه منظوره (GPR) را دارد. در معماری x86 تعداد آن 8 عدد ، در معماری x86-64 تعداد آن 16 عدد و در معماری ARM نیز تعداد آن 16 عدد است. ساده ترین راه برای درک یک ثبات این است که به آن به عنوان یک متغیر موقت و بدون نوع فکر کنیم. تصور کنید که در حال کار با یک زبان برنامه نویسی سطح بالا هستید و تنها می‌توانید از هشت متغیر 32 بیتی (یا 64 بیتی) استفاده کنید. با این حال می‌توان با استفاده از آن‌ها کارهای بسیاری را انجام داد!

ممکن است تعجب کنید که چرا باید تفاوت بین کد ماشین و یک زبان برنامه نویسی وجود داشته باشد. پاسخ در این واقعیت نهفته است که انسان‌ها و پردازنده ها یکسان نیستند – برای انسان‌ها استفاده از یک زبان برنامه نویسی سطح بالا شبیه به C/C++، Java, Python و غیره خیلی ساده‌تر است اما برای یک CPU استفاده از یک سطح بسیار پایین آسان‌تر است. شاید اختراع یک CPU که بتواند کدهای زبان برنامه نویسی سطح بالا را اجرا کند امکان‌پذیر باشد اما این امر می‌تواند خیلی پیچیده‌تر از پردازنده هایی که ما امروزه می‌شناسیم باشد. در یک روش مشابه، برای انسان‌ها برنامه نویسی به زبان اسمبلی به علت اینکه بسیار سطح پایین و مشکل است بدون مقدار زیادی از اشتباهات آزاردهنده بسیار ناخوشایند است. برنامه ای که کدهای نوشته شده به زبان برنامه نویسی سطح بالا را به اسمبلی تبدیل می‌کند یک کامپایلر نامیده می شود.

1.1 چند کلمه در مورد ISAهای مختلف

همیشه x86 ISA یک آپ کد (opcode یا کد عمل‌کرد) با طول متغیر دارد، بنابراین هنگامی که عصر پردازنده های 64 بیتی آمد پسوند x64 تأثیر بسیار قابل توجهی بر ISA نگذاشت. درواقع x86 ISA هنوز شامل مقدار زیادی از دستورها بود که برای اولین بار در پردازنده های 16 بیتی 8086 ظاهر شده بودند

که هنوز هم در پردازنده های امروزی وجود دارند.

ARM یک پردازنده بر اساس معماری کم دستور (RISC) است که با در نظر گرفتن آپ کد با طول ثابت طراحی شده است که در گذشته مزایایی را داشت. در ابتدا تمام دستورهای ARM در ۴ بایت کد گذاری می شدند.امروزه این "حالت ARM" نامیده می شود.

سپس به این فکر کردند که این به اندازه‌ای که آن‌ها در ابتدا تصور می‌کردند به صرفه نیست. درواقع، بیشتر دستورهای پردازنده که برنامه‌های کاربردی از آن استفاده می کنند را می‌توان با استفاده از اطلاعات کمتر رمز گذاری کرد. بنابراین آن‌ها ISA دیگری را اضافه کردند که Thumb نامیده می‌شد که در آن هر دستور تنها در ۲ بایت رمز گذاری می شد. اکنون به عنوان " حالت Thumb" نامیده می شود. با این حال، تمام دستورهای ARM را نمی‌توان تنها در دو بایت رمزگذاری کرد و مجموعه دستورهای Thumb تا حدودی محدود است. شایان ذکر است که کد کامپایل شده برای حالت ARM و حالت Thumb ممکن است همزمان در یک برنامه وجود داشته باشند.

سازندگان ARM فکر می‌کردند که Thumb می‌تواند توسعه داده شود و به همین دلیل Thumb-2 به وجود آمد که در ARMv7 ظاهر شد.Thumb-2 هنوز از دستورهای ۲ بایتی استفاده می‌کرد اما برخی از دستورهای جدید اندازه ۴ بایتی داشتند. یک تصور غلط و رایج این است که Thumb-2 ترکیبی از ARM و Thumb است. این اشتباه است. بلکه Thumb-2 به طور کامل از تمام ویژگی‌های پردازنده پشتیبانی کرده و بنابراین می‌تواند با حالت ARM رقابت کند – یک هدف که به وضوح به دست آمده است این است که اکثر برنامه‌های کاربردی برای iPod/iPhone/iPad با مجموعه دستورهای Thumb-2 کامپایل شده‌اند (مسلماً و عمدتا به دلیل این واقعیت است که Xcode به طور پیش‌فرض این کار را انجام می دهد). بعدا نسخه ۶۴ بیتی ARM عرضه شد. این ISA آپ کدهای ۴ بایتی دارد و نیاز به هیچ حالت Thumb اضافی ندارد. با این حال، نیازهای ۶۴ بیتی ISA را تحت تأثیر قرار می‌دهد که منجر به این شده است که ما در حال حاضر سه مجموعه دستور ARM داشته باشیم : حالت ARM ، حالت Thumb (از جمله Thumb-2) و ARM64. این ISAها تا حدودی همدیگر را پوشش می دهند اما می‌توان گفت به جای اینکه انواع مختلفی از یک مدل باشند آن‌ها ISAهای مختلفی هستند. بنابراین ما باید سعی کنیم که قطعاتی از کد را برای هر سه ARM ISA در کتاب اضافه کنیم.

با این حال بسیاری از ISAها از نوع RISC و با طول ثابت آپ کد ۳۲ بیتی وجود دارند، مانند MIPS, PowerPC و Alpha AXP.

فصل 2

ساده ترین تابع

ساده‌ترین تابع ممکن مسلماً تابعی است که تنها یک مقدار ثابت را برگرداند.

در اینجا یکی وجود دارد :

مثال 2.1 : کد C/C++

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

اجازه بدهید که آن را کامپایل کنیم!

x86 2.1

در اینجا خروجی تولید شده بر روی پلتفرم x86 را برای هر دو کامپایلر بهینه سازی شده GCC و MSVC می‌بینیم :

مثال 2.2 : GCC/MSVC بهینه سازی شده (خروجی اسمبلی)

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

تنها دو دستور در اینجا وجود دارد : خط اول مقدار 123 را به داخل ثبات EAX منتقل می‌کند که طبق قرارداد برای ذخیره سازی مقدار بازگشتی استفاده می‌شود و خط دوم RET است که برگشت به فراخوان را انجام می دهد.