Посвящается Ноортье и Сиетсе

ОГЛАВЛЕНИЕ

СОДЕРЖАНИЕ

ЧАСТЬ IV. ПРИЛОЖЕНИЯ

От издательства

Отзывы и пожелания

Мы всегда рады отзывам наших читателей. Расскажите нам, что вы думаете об этой книге, – что понравилось или, может быть, не понравилось. Отзывы важны для нас, чтобы выпускать книги, которые будут для вас максимально полезны.

Вы можете написать отзыв на нашем сайте *www.dmkpress.com*, зайдя на страницу книги и оставив комментарий в разделе «Отзывы и рецензии». Также можно послать письмо главному редактору по адресу *dmkpress@gmail.com*; при этом укажите название книги в теме письма.

Если вы являетесь экспертом в какой-либо области и заинтересованы в написании новой книги, заполните форму на нашем сайте по адресу *http://dmkpress.com/authors/publish_book/* или напишите в издательство по адресу *dmkpress@gmail.com*.

Скачивание исходного кода примеров

Скачать файлы с дополнительной информацией для книг издательства «ДМК Пресс» можно на сайте *www.dmkpress.com* на странице с описанием соответствующей книги.

Список опечаток

Хотя мы приняли все возможные меры для того, чтобы обеспечить высокое качество наших текстов, ошибки все равно случаются. Если вы найдете ошибку в одной из наших книг, мы будем очень благодарны, если вы сообщите о ней главному редактору по адресу *dmkpress@gmail.com*. Сделав это, вы избавите других читателей от недопонимания и поможете нам улучшить последующие издания этой книги.

Нарушение авторских прав

Пиратство в интернете по-прежнему остается насущной проблемой. Издательства «ДМК Пресс» и No Starch Press очень серьезно относятся к вопросам защиты авторских прав и лицензирования. Если вы столкнетесь в интернете с незаконной публикацией какой-либо из наших книг, пожалуйста, пришлите нам ссылку на интернет-ресурс, чтобы мы могли применить санкции.

Ссылку на подозрительные материалы можно прислать по адресу электронной почты *dmkpress@gmail.com*.

Мы высоко ценим любую помощь по защите наших авторов, благодаря которой мы можем предоставлять вам качественные материалы.

Об авторе

Дэннис Эндриесс имеет степень доктора по безопасности систем и сетей, и анализ двоичных файлов – неотъемлемая часть его исследований. Он один из основных соразработчиков системы целостности потока управления PathArmor, которая защищает от атак с перехватом потока управления типа возвратно-ориентированного программирования (ROP). Также Эндриесс принимал участие в разработке атаки, которая положила конец P2P-сети ботов GameOver Zeus.

О технических рецензентах

Торстен Хольц (Thorsten Holz) – профессор факультета электротехники и информационных технологий в Рурском университете в Бохуме, Германия. Он занимается исследованиями в области технических аспектов безопасности систем. В настоящее время его интересуют обратная разработка, автоматизированное обнаружение уязвимостей и изучение последних векторов атак.

Tim Vidas – хакер-ас. На протяжении многих лет он возглавлял инфраструктурную команду в соревновании DARPA CGC, внедрял инновации в компании Dell Secureworks и курировал исследовательскую группу CERT в области компьютерно-технической экспертизы. Он получил степень доктора в университете Карнеги-Меллона, является частым участником и докладчиком на конференциях и обладает числом Эрдёша–Бейкона 4-3. Много времени уделяет обязанностям отца и мужа.

ВСТУПИТЕЛЬНОЕ СЛОВО

В наши дни нетрудно найти книги по языку ассемблера и даже описания двоичных форматов ELF и PE. Растет количество статей, посвященных прослеживанию потока информации и символическому выполнению. Но нет еще ни одной книги, которая вела бы читателя от основ ассемблера к выполнению сложного анализа двоичного кода. Нет ни одной книги, которая научила бы читателя оснащать двоичные программы инструментальными средствами, применять динамический анализ заражения (taint analysis) для прослеживания путей прохождений интересных данных по программе или использовать символическое выполнение в целях автоматизированного генерирования эксплойтов. Иными словами, нет книг, которые учили бы методам, инструментам и образу мыслей, необходимым для анализа двоичного кода. Точнее, до сих пор не было.

Трудность анализа двоичного кода состоит в том, что нужно разбираться в куче разных вещей. Да, конечно, нужно знать язык ассемблера, но, кроме того, понимать форматы двоичных файлов, механизмы компоновки и загрузки, принципы статического и динамического анализов, расположение программы в памяти, соглашения, поддерживаемые компиляторами, – и это только начало. Для конкретных задач анализа или оснащения могут понадобиться и более специальные знания. Конечно, для всего этого нужны инструменты. Многих эта перспектива настолько пугает, что они сдаются, даже не вступив в борьбу. Так много всего предстоит учить. С чего же начать?

Ответ: с этой книги. Здесь все необходимое излагается последовательно, логично и доступно. И интересно к тому же! Даже если вы ничего не знаете о том, как выглядит двоичная программа, как она

загружается и что происходит во время ее выполнения, в книге вы найдете отлично продуманное введение во все эти понятия и инструменты, с помощью которых сможете не только быстро освоить теоретические основы, но и поэкспериментировать в реальных ситуациях. На мой взгляд, это единственный способ приобрести глубокие знания, которые не забудутся на следующий день.

Но и в том случае, если у вас есть богатый опыт анализа двоичного кода с помощью таких инструментов, как Capstone, Radare, IDA Pro, OllyDbg или что там у вас стоит на первом месте, вы найдете здесь материал по душе. В поздних главах описываются продвинутые методики создания таких изощренных инструментов анализа и оснащения, о существовании которых вы даже не подозревали.

Анализ и оснащение двоичного кода – увлекательные, но трудные техники, которыми в совершенстве владеют лишь немногие опытные хакеры. Но внимание к вопросам безопасности растет, а вместе с ним и важность этих вопросов. Мы должны уметь анализировать вредоносные программы, чтобы понимать, что они делают и как им в этом воспрепятствовать. Но поскольку все больше вредоносных программ применяют методы обфускации и приемы противодействия анализу, нам необходимы более хитроумные методы.

Мы также все чаще анализируем и оснащаем вполне благопристойные программы, например, чтобы затруднить атаки на них. Так, может возникнуть желание модифицировать существующий двоичный код программы, написанной на C++, с целью гарантировать, что все вызовы (виртуальных) функций обращаются только к допустимым методам. Для этого мы должны сначала проанализировать двоичный код и найти в нем методы и вызовы функций. Затем нужно добавить оснащение, сохранив при этом семантику оригинальной программы. Все это проще сказать, чем сделать.

Многие начинают изучать такие методы, столкнувшись с интересной задачкой, оказавшейся не по зубам. Это может быть что угодно: как превратить игровую консоль в компьютер общего назначения, как взломать какую-то программу или понять, как работает вредонос, проникший в ваш компьютер.

К своему стыду, должен сознаться, что лично для меня все началось с желания снять защиту от копирования с видеоигр, покупка которых была мне не по средствам. Поэтому я выучил язык ассемблера и стал искать проверки в двоичных файлах. Тогда на рынке правил бал 8-разрядный процессор 6510 с аккумулятором и двумя регистрами общего назначения. Хотя для того чтобы использовать все 64 КБ системной памяти, требовалось исполнять танцы с бубнами, в целом система была простой. Но поначалу все было непонятно. Со временем я набирался ума от более опытных друзей, и постепенно туман стал рассеиваться. Маршрут был, без сомнения, интересным, но долгим, трудным и иногда заводил в тупик. Многое я бы отдал тогда за путеводитель! Современные 64-разрядные процессоры x86 не в пример сложнее, как и компиляторы, которые генерируют двоичный код. Понять, что делает код, гораздо труднее, чем раньше. Специалист, кото-

рый покажет путь и прояснит сложные вопросы, которые вы могли бы упустить из виду, сделает путешествие короче и интереснее, превратив его в истинное удовольствие.

Дэннис Эндриесс – эксперт по анализу двоичного кода и в доказательство может предъявить степень доктора в этой области – буквально. Однако он не академический ученый, публикующий статьи для себе подобных. Его работы по большей части сугубо практические. Например, он был в числе тех немногих людей, кто сумел разобраться в коде печально известной сети ботов GameOver Zeus, ущерб от которой оценивается в 100 миллионов долларов. И более того, он вместе с другими экспертами безопасности принимал участие в возглавляемой ФБР операции, положившей конец деятельности этой сети. Работая с вредоносными программами, он на практике имел возможность оценить сильные стороны и ограничения имеющихся средств анализа двоичного кода и придумал, как их улучшить. Новаторские методы дизассемблирования, разработанные Дэннисом, теперь включены в коммерческие продукты, в частности Binary Ninja.

Но быть экспертом еще недостаточно. Автор книги должен еще уметь излагать свои мысли. Дэннис Эндриесс обладает этим редким сочетанием талантов: он эксперт по анализу двоичного кода, способный объяснить самые сложные вещи простыми словами, не упуская сути. У него приятный стиль, а примеры ясные и наглядные.

Лично я давно хотел иметь такую книгу. В течение многих лет я читаю курс по анализу вредоносного ПО в Амстердамском свободном университете без учебника ввиду отсутствия такового. Мне приходилось использовать разнообразные онлайновые источники, пособия и эклектичный набор слайдов. Когда студенты спрашивали, почему бы не использовать печатный учебник (как они привыкли), я отвечал, что хорошего учебника по анализу двоичного кода не существует, но если у меня когда-нибудь выдастся свободное время, я его напишу. Разумеется, этого не случилось.

Это как раз та книга, которую я надеялся написать, но так и не собрался. И она лучше, чем мог бы написать я сам.

Приятного путешествия.

Герберт Бос

ПРЕДИСЛОВИЕ

Анализ двоичного кода – одна из самых увлекательных и трудных тем в хакинге и информатике вообще. Она трудна и для изучения, в немалой степени из-за отсутствия доступной информации.

В книгах по обратной разработке и анализу вредоносного ПО недостатка нет, но этого не скажешь о таких продвинутых вопросах анализа двоичного кода, как оснащение двоичной программы, динамический анализ заражения или символическое выполнение. Начинающий аналитик вынужден выискивать информацию по темным закоулкам интернета, в устаревших, а иногда и откровенно вводящих в заблуждение сообщениях в форумах или в непонятно написанных статьях. Во многих статьях, а также в академической литературе по анализу двоичного кода предполагаются обширные знания, поэтому изучение предмета по таким источникам становится проблемой курицы и яйца. Хуже того, многие инструменты и библиотеки для анализа плохо или никак не документированы.

Я надеюсь, что эта книга, написанная простым языком, сделает анализ двоичного кода более доступным и станет практическим введением во все важные вопросы данной области. Прочитав эту книгу, вы будете в достаточной степени экипированы, чтобы ориентироваться в быстро меняющемся мире анализа двоичного кода и отважиться на самостоятельные исследования.

БЛАГОДАРНОСТИ

Прежде всего я хочу поблагодарить свою жену Ноортье и нашего сына Сиетсе за поддержку во время работы над книгой. Это было невероятно горячее время, но вы видели только мою спину.

Я также благодарен всем сотрудникам издательства No Starch Press, которые помогли воплотить эту книгу в реальность, а особенно Биллу Поллоку и Тайлеру Ортману, предоставившим мне возможность взяться за нее, и Анни Чой, Райли Хоффмана и Ким Уимспетт за отличную работу по редактированию и подготовке книги к печати. Спасибо моим техническим рецензентам, Торстену Хольцу и Тиму Видасу, за пространные отзывы, которые способствовали улучшению текста.

Спасибо Бену Грасу, который помог перенести библиотеку `libdft` на современную версию Ubuntu, Джонатану Сэлуэну за замечания о главах, посвященных символическому выполнению, а также Лоренцо Кавалларо, Эрику ван дер Коуве и всем остальным, кто готовил слайды, легшие в основу приложения, касающегося языка ассемблера.

Наконец, я признателен Герберту Босу, Эйше Словинска и всем моим коллегам, которые создали замечательную среду для научной работы, благодаря чему у меня и появилась идея написать эту книгу.

ВВЕДЕНИЕ

Подавляющее большинство компьютерных программ написаны на языках высокого уровня типа C или C++, которые компьютер не может исполнять непосредственно. Такие программы необходимо откомпилировать, в результате чего создаются *двоичные исполняемые файлы*, содержащие машинный код, – его компьютер уже может выполнить. Но откуда мы знаем, что откомпилированная программа имеет такую же семантику, как исходная? Ответ может разочаровать – *а мы и не знаем*!

Существует семантическая пропасть между языками высокого уровня и двоичным машинным кодом, и как ее преодолеть, знают немногие. Даже программисты в большинстве своем плохо понимают, как их программы работают на самом нижнем уровне, и просто верят, что откомпилированная программа делает то, что они задумали. Поэтому многие дефекты компилятора, тонкие ошибки реализации, потайные ходы на двоичном уровне и другие вредоносные паразиты остаются незамеченными.

Хуже того, существует бесчисленное множество двоичных программ и библиотек – в промышленности, в банках, во встраиваемых системах, – исходный код которых давно утерян или является коммерческой собственностью. Это означает, что такие программы и библиотеки невозможно исправить или хотя бы оценить их безопасность на уровне исходного кода с применением традиционных методов. Это реальная проблема даже для крупных программных компаний, свидетельство чему – недавний выпуск компанией Microsoft созданного с большим трудом двоичного исправления ошибки переполнения буфера в программе «Редактор формул», являющейся частью Microsoft Office[1].

[1] *https://0patch.blogspot.nl/2017/11/did-microsoft-just-manually-patch-their.html*.

В этой книге вы научитесь анализировать и даже модифицировать программы на двоичном уровне. Будь вы хакер, специалист по безопасности, аналитик вредоносного кода, программист или просто любопытствующий, эти методы позволят вам лучше понимать и контролировать двоичные программы, которые вы создаете и используете каждый день.

Что такое анализ двоичных файлов, и зачем он вам нужен?

Анализ двоичных файлов, или просто *двоичный анализ*, – это наука и искусство анализа свойств двоичных компьютерных программ, а также машинного кода и данных, которые они содержат. Короче говоря, цель анализа двоичных файлов – определить (и, возможно, модифицировать) истинные свойства двоичных программ, т. е. понять, что они делают в действительности, а не доверяться тому, что они, по нашему мнению, должны делать.

Многие отождествляют двоичный анализ с обратной разработкой и дизассемблированием, и отчасти они правы. Дизассемблирование – важный первый шаг многих видов двоичного анализа, а обратная разработка – типичное приложение двоичного анализа и зачастую единственный способ документировать поведение проприетарного или вредоносного программного обеспечения. Однако область двоичного анализа гораздо шире.

Методы анализа двоичных файлов можно отнести к одному из двух классов, хотя возможны и комбинации.

Статический анализ В этом случае мы рассуждаем о программе, не выполняя ее. У такого подхода несколько преимуществ: теоретически возможно проанализировать весь двоичный файл за один присест, и для его выполнения не нужен процессор. Например, можно статически проанализировать двоичный файл для процессора ARM на компьютере с процессором x86. Недостаток же в том, что в процессе статического анализа мы ничего не знаем о состоянии выполнения двоичной программы, что сильно затрудняет анализ.

Динамический анализ Напротив, в случае динамического анализа мы запускаем программу и анализируем ее во время выполнения. Часто этот подход оказывается проще статического анализа, потому что нам известно все состояние программы, включая значения переменных и выбор ветвей при условном выполнении. Однако мы видим лишь тот код, который выполняется, поэтому можем пропустить интересные части программы.

И у статического, и у динамического анализов есть плюсы и минусы. В этой книге мы расскажем об обоих направлениях. Помимо пассивного двоичного анализа, вы узнаете о методах *оснащения* дво-

ичных файлов, которые позволяют модифицировать двоичные программы, не имея исходного кода. Оснащение двоичных файлов опирается на такие методы анализа, как дизассемблирование, но и само оно может помочь при проведении двоичного анализа. Из-за такого симбиоза приемов двоичного анализа и оснащения в этой книге рассматриваются как те, так и другие.

Я уже говорил, что анализ можно использовать, чтобы документировать или тестировать на отсутствие уязвимостей программы, для которых у вас нет исходного кода. Но даже если исходный код имеется, такой анализ может быть полезен для поиска тонких ошибок, которые более отчетливо проявляются на уровне двоичного, а не исходного кода. Многие методы анализа двоичных файлов полезны также для отладки. В этой книге рассматриваются методы, применимые во всех этих ситуациях, и не только.

В чем сложность анализа двоичных файлов?

Двоичный анализ – вещь гораздо более трудная, чем эквивалентный анализ на уровне исходного кода. На самом деле многие задачи в этом случае принципиально неразрешимы, т. е. невозможно сконструировать движок, который всегда возвращает правильный результат! Чтобы вы могли составить представление о том, с какими проблемами предстоит столкнуться, ниже приведен список некоторых вещей, затрудняющих анализ двоичных файлов. Увы, этот список далеко не исчерпывающий.

Отсутствует символическая информация В исходном коде, написанном на языке высокого уровня типа C или C++, мы даем осмысленные имена переменным, функциям, классам и т. п. Эти имена мы называем *символической информацией*, или просто *символами*. Если придерживаться хороших соглашений об именовании, то понять исходный код будет гораздо проще, но на двоичном уровне имена не имеют ни малейшего значения. Поэтому из двоичных файлов информация о символах часто удаляется, из-за чего понять код становится гораздо труднее.

Отсутствует информация о типах Еще одна особенность программ на языках высокого уровня – наличие у переменных четко определенных типов, например `int`, `float`, `string` или более сложных структурных типов. На двоичном же уровне типы нигде явно не упоминаются, поэтому понять структуру и назначение данных нелегко.

Отсутствуют высокоуровневые абстракции Современные программы состоят из классов и функций, но компиляторы отбрасывают эти высокоуровневые конструкции. Это означает, что двоичный файл выглядит как огромный «комок» кода и данных, а не хорошо структурированный код, и восстановить высокоуровневую структуру трудно и чревато ошибками.