ГЛАВА 7

Программные уязвимости в образах контейнеров

Исправление уязвимостей в ПО — с давних пор важный аспект сопровождения безопасности развернутого кода. В мире контейнеров эта проблема не теряет своей значимости, но, как вы увидите в данной главе, процесс исправления был полностью переосмыслен. Но сначала посмотрим, что такое программные уязвимости и как публикуется и отслеживается информация о них.

Исследования уязвимостей

Уязвимость — это известный изъян во фрагменте программного обеспечения, с помощью которого злоумышленник может произвести какие-либо вредоносные действия. В общем случае чем сложнее фрагмент ПО, тем вероятнее, что в нем есть изъяны, такие, что по крайней мере некоторыми из них может воспользоваться злоумышленник.

А когда в широко используемом программном обеспечении существует уязвимость, злоумышленники могут задействовать ее повсюду, где развернуто это ПО. Как следствие, существует целая сфера исследований, посвященная поиску и публикации информации об уязвимостях в общедоступном ПО, особенно в пакетах для операционных систем и библиотеках языков программирования. Наверное, вы слышали о некоторых наиболее катастрофических уязвимостях, таких как Shellshock, Meltdown и Heartbleed, имеющих не только название, но иногда даже и эмблему. Но эти рок-звезды в мире уязвимостей составляют лишь малую толику из тысяч проблем, о которых сообщается ежегодно.

После того как уязвимость будет выявлена, начинается «забег» по скорейшей публикации исправлений, чтобы пользователи могли задействовать их прежде, чем уязвимость задействуют злоумышленники. Если сразу же

оповещать широкую публику о новых проблемах в ПО, то перед злоумышленниками откроются широкие возможности по использованию этих проблем. Во избежание этого была принята методология ответственного подхода к раскрытию проблем безопасности. Обнаружив уязвимость, аналитик в области безопасности связывается с разработчиком или поставщиком соответствующего ПО и договаривается о сроке, по истечении которого может публиковать полученные результаты. Эти сроки давят на поставщика в положительном смысле, заставляя его как можно быстрее выпустить исправление, ведь как для него, так и для его пользователей будет лучше, если исправление будет доступно до публикации результатов.

Каждой новой проблеме присваивается уникальный идентификатор, начинающийся с букв CVE (Common Vulnerabilities and Exposures — распространенные уязвимости и дефекты), за которыми следует год. Например, уязвимость Shellshock была открыта в 2014 году и официально называется CVE-2014-6271. Организация, которая заведует этими идентификаторами, называется MITRE (https://mitre.org/). Она курирует несколько комитетов по нумерации CVE (CVE Numbering Authority, CNA), имеющих право присваивать идентификаторы CVE в определенных рамках. Некоторые крупные поставщики программного обеспечения — например, Microsoft, Red Hat и Oracle — являются CNA с правом присваивать идентификаторы уязвимостям в их собственных программных продуктах. GitHub стал CNA в конце 2019 года.

Эти идентификаторы CVE используются в Национальной базе уязвимостей (National Vulnerability Database, NVD) (https://nvd.nist.gov/) для отслеживания пакетов и версий программ, затронутых каждой из уязвимостей. На первый взгляд кажется, что на этом все и заканчивается — есть список всех затронутых версий пакетов, так что если у вас установлена одна из этих версий, значит, ваша система уязвима для атаки. К сожалению, не все так просто, поскольку в зависимости от используемого дистрибутива Linux может существовать исправленная версия соответствующего пакета.

Уязвимости, исправления и дистрибутивы

Возьмем для примера Shellshock — критически важную уязвимость, затронувшую пакет `bash` GNU. На странице NVD, посвященной CVE-2014-6271 (https://oreil.ly/XGgEb), перечислен длинный список уязвимых версий, от 1.14.0 до 4.3. Если вы работаете на очень старой версии Ubuntu, 12.04, и обнаружили, что версия bash на вашем сервере — 4.2-2ubuntu2.2, то можете подумать,

что она уязвима, поскольку основана на `bash` 4.2, включенной в список NVD для уязвимости Shellshock.

На самом же деле согласно инструкции по безопасности Ubuntu для этой уязвимости (https://oreil.ly/IEUqF) к данной конкретной версии уже применено исправление от данной уязвимости, и она безопасна. Дело в том, что специалисты по сопровождению Ubuntu решили, что лучше задействовать исправление уязвимости и опубликовать исправленную версию, чем заставлять всех работающих на 12.04 обновляться до совершенно новой младшей версии `bash`.

Чтобы составить полную картину того, уязвимы ли установленные на сервере пакеты, необходимо заглянуть не только в NVD, но и в инструкции по безопасности для конкретного дистрибутива.

До сих пор в этой главе шла речь о пакетах (например, `bash` в предыдущем примере), распространяемых в двоичном виде с помощью таких систем управления пакетами, как `apt`, `yum`, `rpm` или `apk`. Эти пакеты используются всеми приложениями, присутствующими в файловой системе, и данный факт приводит к бесчисленным проблемам на серверах и виртуальных машинах: приложение может зависеть от определенной версии пакета, несовместимой с другим приложением, которое должно работать на той же машине. Эта проблема управления зависимостями — одна из тех, в решении которых могут помочь контейнеры благодаря отдельной корневой файловой системе каждого из них.

Уязвимости уровня приложения

Встречаются и уязвимости на уровне приложений. Большинство приложений используют сторонние библиотеки, устанавливаемые обычно с помощью системы управления пакетами, ориентированной на конкретный язык. В Node.js применяется `npm`, в Python — `pip`, в Java — `Maven` и т. д. Сторонние библиотеки, установленные с помощью этих утилит, — еще один потенциальный источник уязвимостей.

В компилируемых языках, например в Go, C и Rust, сторонние зависимости либо устанавливаются в виде совместно используемых библиотек, либо подключаются к исполняемому файлу во время сборки.

У автономных двоичных исполняемых файлов по определению (в соответствии с эпитетом «автономный») нет внешних зависимостей. У них могут быть зависимости от сторонних библиотек, встроенные тем не менее в эти исполняемые файлы. В таком случае можно создать образ контейнера на основе пустого базового образа, содержащего только нужный двоичный исполняемый файл.

Если у приложения нет зависимостей, то его нельзя просканировать на предмет известных уязвимостей. Но оно все равно может включать изъяны, делающие его уязвимым для злоумышленников, которые мы обсудим в разделе «Уязвимости нулевого дня» на с. 132.

Управление рисками, связанными с уязвимостями

Решение проблемы уязвимостей программного обеспечения — один из важных аспектов управления рисками. Любое развертываемое нетривиальное ПО, скорее всего, содержит некоторое количество уязвимостей, и существует риск атаки на приложение с их помощью. Чтобы управлять этим риском, необходимо идентифицировать существующие уязвимости и оценить степень их серьезности, распределить по степени приоритетности и наладить процессы их исправления либо снижения негативных последствий.

Сканеры уязвимостей автоматизируют процесс идентификации уязвимостей и предоставляют информацию о степени серьезности проблем, а также версиях пакетов ПО, в которых они были исправлены (если исправление было опубликовано).

Сканирование на уязвимости

Если поискать в Интернете, то можно найти огромное множество утилит для сканирования на уязвимости, охватывающих разнообразные методики, включая такие утилиты сканирования портов, как `nmap` и `nessus`, предназначенные для внешнего поиска уязвимостей в системе, работающей в продакшене. Это ценный подход, но в данной главе нас больше интересуют

утилиты, с помощью которых можно найти уязвимости в ПО, установленном в корневой файловой системе.

Чтобы идентифицировать присутствующие в системе уязвимости, сначала необходимо установить, какое программное обеспечение в ней есть. Существует несколько различных механизмов установки ПО:

- корневая файловая система, основанная на корневой файловой системе дистрибутива Linux, в которой могут быть уязвимости;
- установленные через систему управления пакетами Linux (например, `rpm` или `apk`) системные пакеты, а также ориентированные на конкретный язык программирования пакеты, установленные с помощью таких утилит, как `pip` и `RubyGems`;
- программное обеспечение, установленное непосредственно с помощью `wget`, `curl` или даже FTP.

Некоторые сканеры зависимостей запрашивают у системы управления пакетами список установленного программного обеспечения. В случае использования одной из подобных утилит следует избегать установки ПО напрямую, поскольку оно не будет просканировано на предмет уязвимостей.

Установленные пакеты

Как вы видели в главе 6, любой образ контейнера может включать дистрибутив Linux, возможно, с какими-либо уже установленными пакетами, помимо кода приложения. Может быть запущено много экземпляров каждого контейнера, причем каждый — со своей копией файловой системы образа контейнера, включая все потенциально уязвимые пакеты, содержавшиеся в нем. На рис. 7.1 показана подобная ситуация: два экземпляра контейнера X и один экземпляр контейнера Y. Кроме того, показаны дополнительные пакеты, установленные непосредственно на хост-компьютере.

Установка пакетов непосредственно на хосте — обычное дело, на самом деле именно их исправления системные администраторы всегда должны были устанавливать из соображений безопасности. Для этого нередко администраторы подключались к каждому хосту по SSH и устанавливали исправленный пакет. В эпоху же ориентации на облачные сервисы это не приветствуется, поскольку изменение состояния машины вручную подобным образом озна-

чает невозможность автоматического восстановления ее в том же состоянии. Вместо этого лучше либо создать новый образ машины с обновленными пакетами, либо обновить сценарии автоматизации, служащие для подготовки образов, чтобы новые установки включали обновленные пакеты.

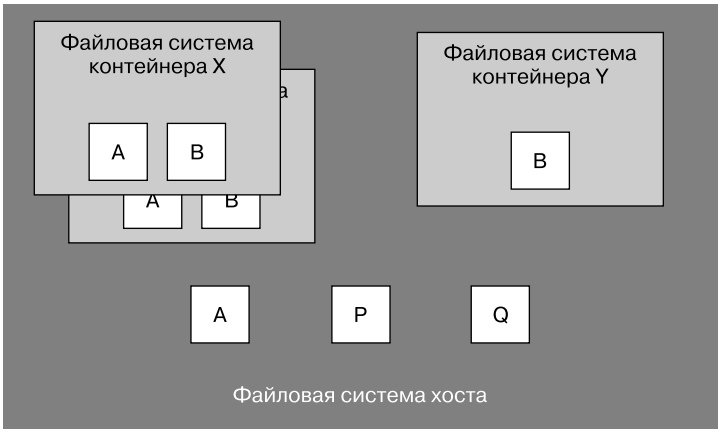

Рис. 7.1. Пакеты на хосте и в его контейнерах

Сканирование образов контейнеров

Чтобы знать, не запущены ли в развернутой системе контейнеры с уязвимым программным обеспечением, необходимо просканировать в них все зависимости. Для этого существует несколько различных подходов.

Представьте утилиту, сканирующую все запущенные на хосте (или в системе, развернутой на нескольких хостах) контейнеры. В современных системах, развертываемых в облаках, можно нередко видеть сотни экземпляров контейнеров, созданных на основе одного образа контейнера, и просматривать одни и те же зависимости сотни раз — очень неэкономично. Гораздо более рациональным будет просканировать образ контейнера, на котором основаны все эти контейнеры.

Однако данный подход требует, чтобы в контейнерах присутствовало только программное обеспечение из образа контейнера и ничего более. Работающий в каждом из контейнеров код должен быть *неизменяемым* (immutable). Посмотрим, каковы преимущества таких неизменяемых контейнеров.

Неизменяемые контейнеры

(Обычно) ничто не мешает контейнеру после запуска скачивать дополнительное программное обеспечение и размещать его в своей файловой системе. И действительно, на заре развития контейнеров подобный паттерн встречался довольно часто, поскольку считался удобным способом обновления ПО контейнера до последних версий, который не нуждался в повторной сборке образа контейнера. Если до сих пор эта идея не приходила вам в голову, то забудьте ее сразу же, она считается исключительно неудачной по многим причинам, включая представленные ниже.

- Если контейнер скачивает код во время выполнения, то в различных экземплярах контейнера могут работать разные версии этого кода, причем отследить, в каком экземпляре работает та или иная версия, будет непросто. В отсутствие сохраненной версии кода из данного контейнера будет сложно (или даже невозможно) воссоздать его идентичную копию, что создаст проблемы при попытках воспроизвести проблемы, возникшие при эксплуатации.
- Труднее контролировать и гарантировать происхождение программного обеспечения, работающего в каждом из контейнеров, если оно может скачиваться в любой момент времени и откуда угодно.
- Сборку образа контейнера и сохранение его в реестре можно с легкостью автоматизировать в конвейере CI/CD. Совсем нетрудно также добавить в тот же конвейер дополнительные проверки безопасности — например, сканирование на уязвимости или проверку цепочки поставок программного обеспечения.

Во многих системах, развертываемых в промышленной эксплуатации, контейнеры считаются неизменяемыми, просто в порядке практической рекомендации, но никаких мер для обеспечения этого не предпринимается. Существуют утилиты, способные автоматически обеспечить неизменяемость контейнеров с помощью запрета на запуск в контейнере отсутствовавших там на момент сканирования исполняемых файлов. Данная методика, обсуждаемая подробнее в главе 13, называется *предотвращением отклонений* (drift prevention).

Еще один способ реализовать неизменяемость — запустить контейнер с файловой системой, предназначенной только для чтения. Если коду приложения требуется доступ к открытому для записи локальному хранилищу, то можно смонтировать открытую для записи временную файловую систему. Здесь

может понадобиться вносить изменения в приложение, чтобы оно записывало данные лишь в эту временную файловую систему.

Если контейнеры неизменяемые, то достаточно просканировать только образ, чтобы найти все уязвимости, которые могут встретиться во всех контейнерах, основанных на нем. Но, к сожалению, просканировать их однократно недостаточно. Обсудим, почему необходимо регулярное сканирование.

Регулярное сканирование

Как обсуждалось в начале данной главы, по всему миру есть множество исследователей, занимающихся безопасностью, которые ищут неизвестные ранее уязвимости в существующем коде. Иногда эти люди находят проблемы, существовавшие уже многие годы. Один из самых ярких примеров — HeartBleed — критическая уязвимость в широко используемом пакете OpenSSL, основанная на проблеме в потоке heartbeat-запросов и ответов, поддерживающем TLS-соединение в активном состоянии. Эта уязвимость, обнаруженная в апреле 2014-го, позволяла злоумышленнику отправлять heartbeat-запрос, составленный специальным образом, на небольшое количество данных в большом буфере. Отсутствие проверки длины в коде OpenSSL приводило к тому, что в ответ отправлялось небольшое количество данных, а остальная часть буфера заполнялась содержимым используемой в текущий момент памяти, которое могло включать конфиденциальные данные, попадавшие таким образом к злоумышленнику. Было установлено, что эта уязвимость стала причиной серьезных утечек данных, связанных с потерей паролей, номеров социального страхования и медицинских данных.

Столь серьезные случаи, как HeartBleed, редки, но лучше всегда предполагать, что в любой сторонней зависимости рано или поздно может обнаружиться новая уязвимость. И, к сожалению, неизвестно, когда это произойдет. Даже если сам код не меняется, в его зависимостях могут обнаружиться новые уязвимости.

При регулярном сканировании образов контейнеров утилита сканирования может проверять их содержимое на актуальной базе знаний уязвимостей (из NVD и прочих источников инструкций по безопасности). Очень часто все развернутые образы сканируют каждые 24 часа, помимо сканирования новых образов при сборке, в качестве составной части автоматизированного конвейера CI/CD.

Средства сканирования

Существует множество утилит для сканирования образов контейнеров, начиная от реализаций с открытым исходным кодом, таких как Trivy (https://oreil.ly/SxKQT), Clair (https://oreil.ly/avK-2) и Anchore (https://oreil.ly/7rFFt), и до коммерческих решений от таких компаний, как JFrog, Palo Alto и Aqua. Во многих реестрах образов контейнеров, например Docker Trusted Registry (https://docs.docker.com/ee/dtr), и в проекте Harbor (https://goharbor.io/) от CNCF, а также в реестрах от основных поставщиков облачных сервисов есть встроенные возможности сканирования.

К сожалению, выдаваемые различными сканерами результаты сильно разнятся, и имеет смысл обсудить почему.

Источники информации

Как уже обсуждалось ранее в этой главе, есть множество источников информации об уязвимостях, включая инструкции по безопасности различных дистрибутивов. У Red Hat их даже несколько (https://oreil.ly/jE4ad) — их лента новостей OVAL включает только уязвимости, для которых существует исправление, и не включает уже опубликованные, но пока не исправленные.

Если сканер не использует данные из ленты новостей по безопасности дистрибутива и основывается лишь на данных NVD, то будет, вероятно, выдавать множество ложнопозитивных результатов для образов, в основе которых лежит этот дистрибутив. Если вы предпочитаете конкретный дистрибутив Linux в своих образах или программные решения наподобие distroless, то убедитесь, что сканер его поддерживает.

Устаревшие источники

Иногда персонал, отвечающий за сопровождение дистрибутивов, меняет способ извещать об уязвимостях. Относительно недавно это произошло с дистрибутивом Alpine: вместо информации об уязвимостях на alpine-secdb (https://oreil.ly/lVHll) теперь используется новая система на aports (https://oreil.ly/J1-YA). На момент написания данной книги есть сканеры, которые до

сих пор используют только данные из старой ленты новостей alpine-secdb, не обновлявшейся уже несколько месяцев.

Не все уязвимости исправляются

Иногда персонал, отвечающий за сопровождение дистрибутивов, решает не исправлять конкретную уязвимость (например, поскольку риск пренебрежимо мал, а исправить проблему непросто либо потому, что способ взаимодействия с прочими пакетами на их платформе делает невозможным использование этой уязвимости).

А раз уязвимость исправлять никто не собирается, то у разработчиков сканеров возникает философский вопрос: отображать ли уязвимость в списке результатов, если ничего все равно сделать нельзя? Мы, в Aqua, слышали от некоторых клиентов, что они не хотели бы видеть данную категорию результатов, поэтому мы предоставили пользователю возможность выбора. Все это свидетельствует о том, что не существует «правильных» наборов результатов, когда речь идет о сканировании на уязвимости.

Уязвимости подпакетов

Иногда система управления пакетами устанавливает пакет и сообщает о нем как о едином целом, однако на самом деле он состоит из нескольких подпакетов. Хороший пример: пакет `bind` в Ubuntu. Иногда он устанавливается с одним только подпакетом `docs`, включающим, как легко догадаться, лишь документацию. Некоторые сканеры предполагают, что, если система управления пакетами сообщает об установленном пакете, значит, установлен весь пакет (со всеми возможными подпакетами). Это может привести к ложнопозитивным результатам, когда сканер сообщает об уязвимостях, которых нет и быть не может, поскольку соответствующий подпакет вообще не установлен.

Различия названий пакетов

Имя источника для пакета может включать двоичные файлы с совершенно различными названиями. Например, в Debian пакет `shadow` (https://oreil.ly/SpPXQ) включает двоичные файлы `login`, `passwd` и `uidmap`. Если сканер это не учитывает, то может выдавать ложнонегативные результаты.

Дополнительные возможности сканирования

Некоторые сканеры образов способны выявлять и другие проблемы, помимо уязвимостей, например:

- известное вредоносное программное обеспечение, содержащееся в образе;
- исполняемые файлы с установленным битом `setuid` (что, как вы видели в главе 2, может позволить повысить полномочия);
- образы, конфигурация которых предусматривает выполнение от имени суперпользователя;
- секретные учетные данные, например токены или пароли;
- конфиденциальные данные, например номера платежных карт, номера социального страхования или что-то в этом роде.

Ошибки сканеров

Надеюсь, материал данного раздела позволяет понять, что поиск уязвимостей — не такая простая задача, как может показаться на первый взгляд. Поэтому весьма вероятно, что при использовании любого сканера будут встречаться ложнопозитивные или ложнонегативные результаты из-за ошибок в сканере или изъянов в информации об уязвимостях, которую он задействует.

Тем не менее со сканером лучше, чем без него. Если у вас нет сканера или вы не используете его регулярно, то никак не узнаете, не станет ли ваше программное обеспечение легкой добычей злоумышленников. Время в этом смысле плохой лекарь — критическая уязвимость Shellshock была обнаружена в коде, существовавшем уже десятилетия. Если использовать запутанные зависимости, то следует ожидать, что рано или поздно в них обнаружатся какие-либо уязвимости.

Ложнопозитивные результаты раздражают, но некоторые утилиты позволяют заносить отдельные отчеты об уязвимостях в белый список, поэтому вы можете сами решить, следует ли на них реагировать.

Будем считать, что мы убедили вас в преимуществах использования сканера, и рассмотрим возможные варианты включения его в ваш технологический процесс.

Сканирование в конвейере CI/CD

Пройдем по конвейеру CI/CD слева направо, где этап написания кода — крайний слева, а этап развертывания в промышленной эксплуатации — крайний справа, как показано на рис. 7.2. В данном конвейере желательно устранять проблемы как можно раньше — это проще и дешевле, точно так же, как поиск и исправление программных ошибок требует намного больше времени и затрат после развертывания, чем во время разработки.

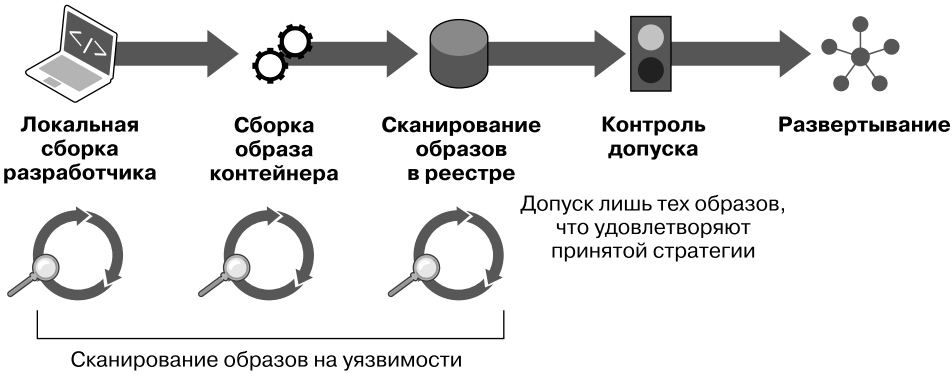

Рис. 7.2. Сканирование на уязвимости в конвейере CI/CD

При обычном развертывании на хосте все работающее на нем программное обеспечение использует одни и те же пакеты. За их регулярное обновление путем установки исправлений безопасности в организации обычно отвечает группа специалистов по компьютерной безопасности. Данная деятельность практически не зависит от этапов разработки и тестирования жизненного цикла приложения и располагается ближе к правому краю конвейера развертывания. Нередко возникают проблемы, связанные с совместным использованием одного и того же пакета различными приложениями, которым нужны разные его версии. Такие проблемы требуют продуманного управления зависимостями и в некоторых случаях изменений кода.

И напротив, как вы видели в главе 6, при развертывании на основе контейнеров все образы включают собственные зависимости, поэтому контейнеры для различных приложений могут включать различные версии пакетов, если это необходимо. Не нужно волноваться о совместимости кода приложения и набора используемых зависимостей. Это, с учетом существования утилит