1) Описание необходимых для понимания технических деталей.
В данном докладе речь пойдет об операционных системах на базе ядра Linux. Что такое операционная система? Набор программ, которые позволяют вам работать с вашим компьютером и запускать другие программы.

1.1) Хранение переменных окружения и аргументов программы.
В Linux существует такое понятие как процесс. Это некоторая единица активности ОС, в которой существуют последовательные действия, текущее состояние и набор связанных ресурсов. У процесса существует некоторое адресное пространство, которое делится на кучу, стек, код библиотек, сегмент кода и так далее. Нам в данном случае важен стек, поскольку именно в нем запущенная программа хранит переменные окружения и аргументы программы.
…
Дальше там на слайдах приводится то, где и как хранятся аргументы и переменные окружения.
….
Собственно, нам важно то, что переменные окружения и аргументы программы отделяются друг от друга NULLом, это как раз нам в дальнейшем и пригодится для осуществления эксплоита.

1.2) Привилегии пользователя.
Пару слов о привелегиях пользователя. Поскольку система Linux является многопользовательской, то в ней могут работать одновременно и поочереди несколько пользователей. Каждый из пользователей обладает своими правами доступа. Также помимо обычных пользователей существует суперпользователь, так называемый root, который может читать любые данные других пользователей, а также изменять, удалять, добавлять свои данные в их домашнюю директорию. Кроме того, он может изменять важные для системы ресурсы. Чтобы у каждого пользователя были разные права, в Linux есть такая вещь как авторизация, все с ней были знакомы еще на курсе веб программирования.
…
В дальнейшем будет рассмотрено как с помощью данной уязвимости можно избежать процесса авторизации и сразу стать суперпользователем.

1.3) Дистрибутивы Linux.
В GNU/Linux есть понятие дистрибутивов: это такие операционные системы, ядро в которых — Linux, в них используется бесплатное программное обеспечение GNU, а также какие-либо свои предустановленные пакеты. Так например в Debian во время установки мы выбираем конфигурацию системы, и в результате устанавливается необходимое для корректной работы данной конфигурации программное обеспечения, оно «связывается» с помощью различных конфигов и в результате мы получаем рабочую требуемую нам операционную систему.

1.4) Библиотека polkit.
Это набор утилит, которые предоставляют непривелигированным процессам возможности выполнения действия, требующих прав суперпользователя. Данная библиотека установлена на большинстве распространенных дистрибутивах, таких как Ubuntu, Debian, Fedora, CentOS и других, что означает, что до того, как данная уязвимость была исправлена, большинство машин, на которых установлены данные дистрибутивы, могли стать уязвимыми, а, учитывая то, что основная часть серверов сейчас работает под управлением Linux, это является большой проблемой.

2) Объяснение причин, из-за которых произошла данная уязвимость.
В дальнейшем мы будем рассматривать исходный код программы pkexec из пакета polkit: будет объяснено, в чем же заключается уязвимость, а также код связанных библиотек для лучшего понимания того, что происходит.
Отметим для чего нужна программа pkexec. На слайде 9 показано описание. Напоминает sudo.

Слайд 10. Теперь перейдем к самому эксплоиту. На 534 строчке мы видим, что к n, означающий номер аргументов, присваивается единица. Однако мы можем передать программе 0 аргументов, тогда на основании того, как устроена память, в нашей программе указатель будет показывать на 0 переменную окружения. Обычно 0 аргументом программы является ее название.
Слайд 11. Значит в случае, если у нас 0 аргументов, то на 610 строке мы записываем в path 0 переменную окружения.
Слайд 12. Если переменная окружения не начинается на `/`, то в случае, если программа будет найдена в PATH, в 0 переменную окружения запишется PATH.
Слайд 13. На 702 строке чистятся переменные окружения, поэтому необходимо реализовать уязвимость между кодом. Как
Слайд 14. В чем же проблема просто взять и не добавить переменную окружения при старте программе. Дело в том, что мы не можем передать некоторые переменные окружения, так называемые Insecure Environment Variables, поскольку они фильтруются библиотекой ld.so для программ, которые должны быть исполнены как другой пользователь. Таким образом, поскольку мы раннее показали, что можем записать любую переменную окружения с помощью Out of Bounds Write, то это эскалация привилегиями.
The environment of a process is only available to the user (euid) running the process.
Слайд 15. Вернемся обратно к pkexec.c. Нам необходимо запустить уязвимость между строчками 610 и 702. Для этого как раз подойдет функция validate_environment_variable. На 413 строке, в случае, если выполняется условие в скобочках, то вызывается функция gprinterr, она нам и нужна.
Слайд 16. Она работает по принципу, что если у нас кодировка UTF-8, то она печатает сообщение как обычно, а если нет, то будет искать с помощью переменной окружения GCONV_PATH модуль конвертации из одной кодировки к другой, а поскольку мы можем записать любую переменную окружения, то это и будет нашим эксплоитом.
Слайд 17. Пишем свой модуль конвертации, и в нем устанавливаем с помощью setuid uid суперпользователя.
Слайд 18. Программа, которую мы запускаем, чтобы получить права суперпользователя.
Слайд 20. Vulnerability resolved. По ссылке можно перейти и протестировать работоспособность уязвимости, до того, как она была решена.

3) Обобщение проблемы.
Основная проблема, в результате которой стала возможность данная уязвимость — Out of Bounds Read/Write Operations. Она означает, что мы можем прочитать ту часть памяти, которую не хотели или нам не предназначалась, или записать данные за границу аллоцированной памяти или в другую часть аллоцированной памяти, что потенциально можно привести к arbitrary code execution, т.е произвольному исполнению кода, или к сбою. Возможно вам знакомы такие вещи как segmentation fault, buffer overflow. Они часто возникают в связи с Out of Bounds.

Данная причина связана с тем, как устроена память существующих компьютеров: ячейки памяти имеют свои адреса, сама память непрерывна.

Чтобы отлавливать данную ошибку на ранних этапах, в современных операционных системах память бьется на страницы, а также каждый процесс запускается в своем виртуальном адресном пространстве. Таким образом, если процесс попытается что-то сделать с памятью, которая ему не предназначается, то получит Segfault.

Данная проблема является довольно распространенной.

4) Принятые меры в связи с данным инцидентом (приветствуются диаграммы рассматриваемых процессов).

На самом деле тяжело говорить что-либо о принятых мерах конкретно в данном случае, поскольку данные инциденты являются довольно частыми, однако если говорить о тенденции индустрии в целом, то в последнее время все больше и больше приложений пишется:

- на более высокоуровневых языках программирования, где управление памятью сокрыто и отдается на интерпретатор или компилятор,
- Используем уже написанное и протестированное программное обеспечение для создания своих решений. (Java и Kotlin, Scala, Groovy, Clojure)
- Ограничение ручного управления памятью в языках программирования в «эффективных» языках программирования (Rust и Unsafe блоки кода)
- Современные компиляторы умеют проверять код на возможные уязвимости (Врубаем кучу флагов с вероятностными ошибками в C/C++ или Rust)
- Популяризация организаций, нацеленных на выявление, определение и каталогизирование публично раскрытых уязвимостей кибербезопасности (CVE — Common Vulnerabilities and Exposures)
- Появление платформ безопасности для разработчиков (Snyk): выявление уязвимостей на основании базы данных с открытым исходным кодом.

5) Мораль.

Пользователям:

- Используйте последние версии программного обеспечения, поскольку зачастую чем новее версия, тем меньше в ней ошибок и уязвимостей
- Не загружайте исполняемые файлы из незнакомых источников, так как они могут использовать уязвимости ПО для своих целей
- Перед использованием приложения можно прогнать его в контейнере и посмотреть на уязвимости, обнаруженные в данном приложении. Тогда можете с почти с безопасностью ставить его на основную машину.
- Используйте различные анализаторы исходных кодов и исполняемых программ для обнаружения уязвимостей, например, Snyk, Checkmarx, Sonatype и другие.
- мораль может быть адресована

Разработчикам:

- Также используйте различные анализаторы, профилировщики, санитайзеры.
- Смотрите на ошибки компиляторов
- Пишите на более высокоуровневых языках программирования, если не требуется обратного

Администраторам:

- Будьте вкурсе новостей уязвимостей ПО. Короче читайте CVE.