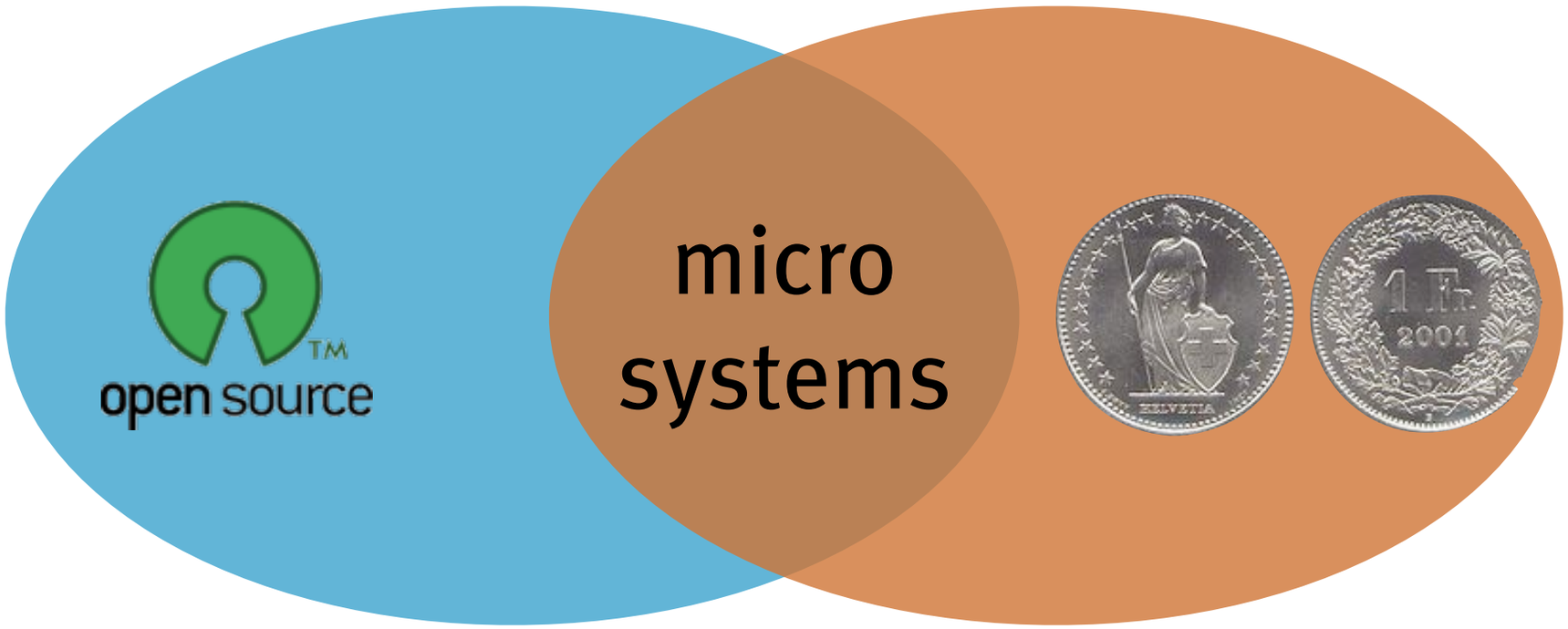
```
-- Marc Balmer, micro systems, <marc@msys.ch>
-- Prague PostgreSQL Developers Day 2011

-- http://www.p2d2.cz/


    CREATE OR REPLACE FUNCTION
        presentation()
    RETURNS void AS
    $$
    BEGIN
       RAISE NOTICE 'Asynchronous Notifications
            For Fun And Profit';
    END;
    $$ LANGUAGE plpgsql
```

micro systems

[Who we are and what we do]

Free Software, Commercial Applications



The Basel Zoo

LISTEN / NOTIFY

SQL Level Syntax

LISTEN

LISTEN channel

NOTIFY

NOTIFY channel [, payload]

UNLISTEN

UNLISTEN channel

LISTEN / NOTIFY

A demonstration

From C to Data via PL/pgSQL

Client: Calls a function (PL/pgSQL)

Server: Implementation of application logic

Helper-functions

Client: Calls a function (PL/pgSQL)

Server: Implementation of application logic

Helper-functions

PGresult *PQvexec(conn, fmt, ...)

PQXtRegister(ctx, conn, txt, func)

PQXtUnregister(conn, txt)

PQXtUnregisterAll(conn)

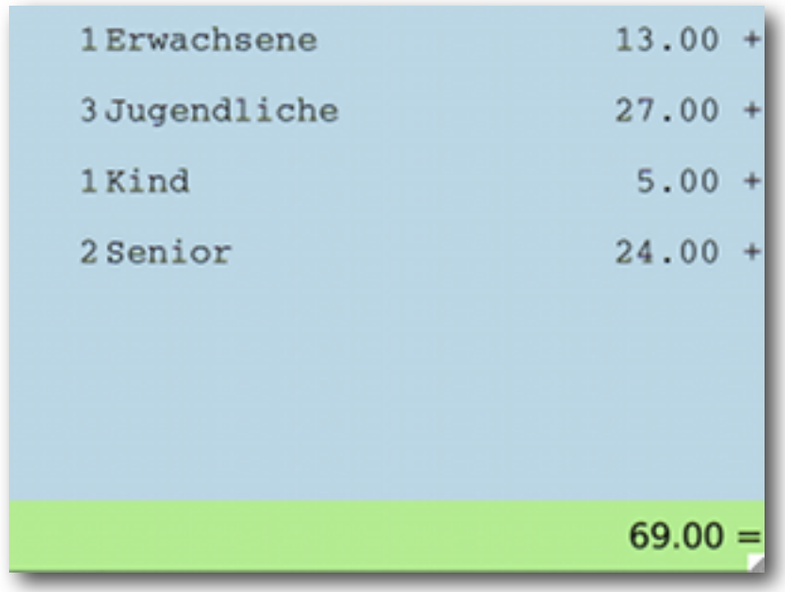
PQvexec

```
PGresult *res;

res = PQvexec(conn, „SELECT name, nr FROM stt_product
    WHERE i d= %d“, id);
```

Careful with string parameters!

Real World Example:
ECR with Customer Display

ECR – Customer Display

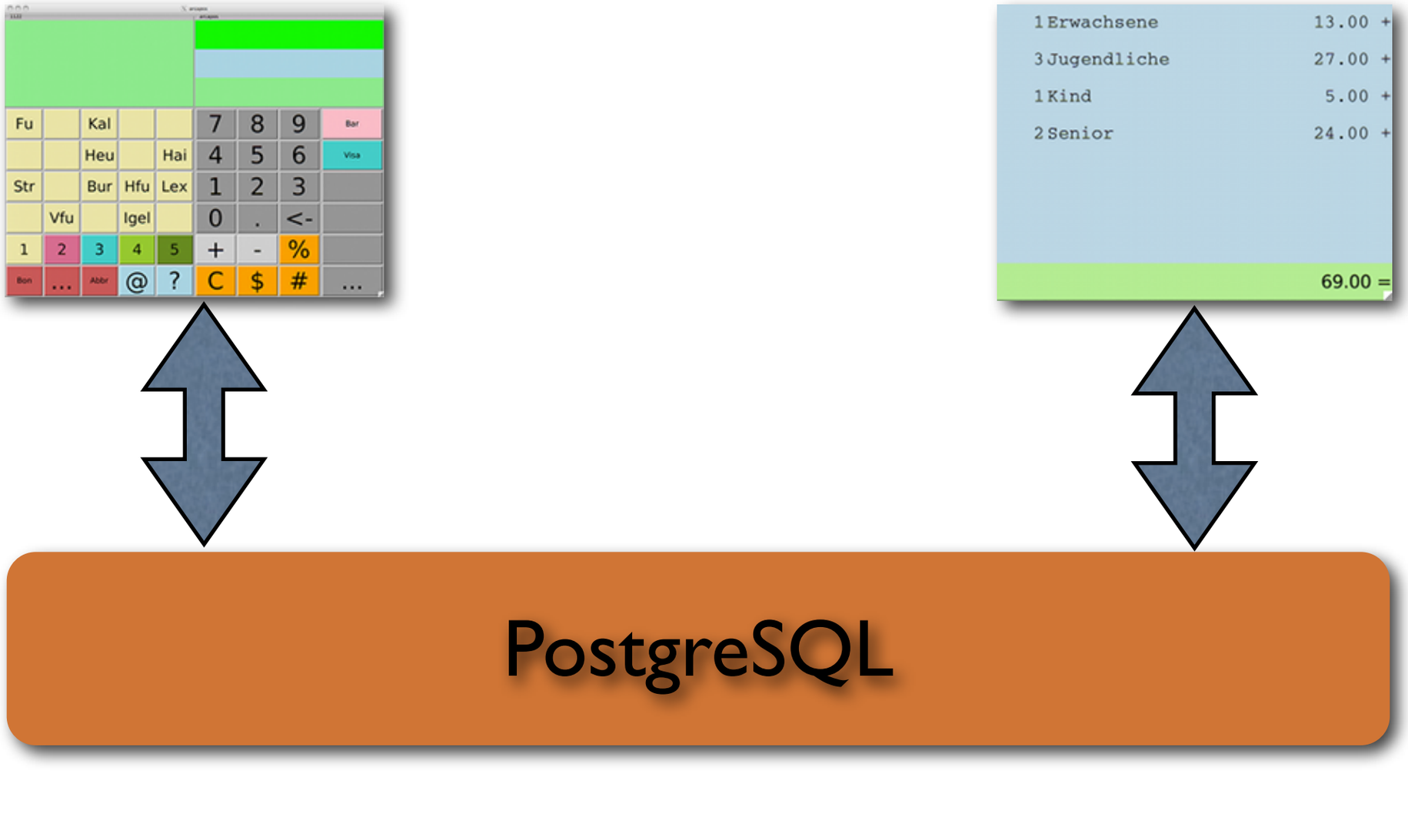


ECR – Customer Display in Action

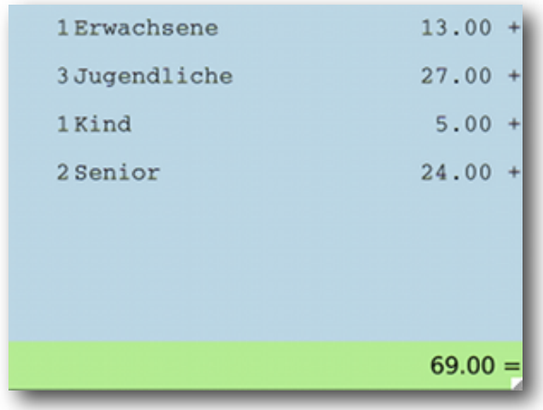
A demonstration

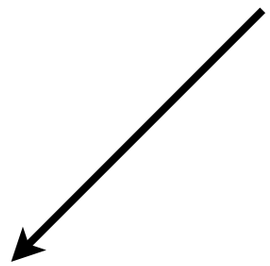
Direct Communication?

Decoupling by „Asynchronous Notifications“



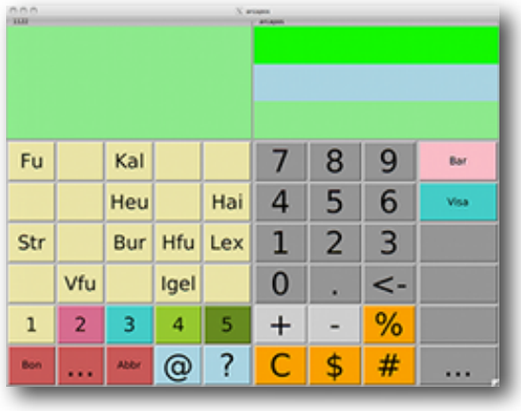
„Subscribe“ to an Event

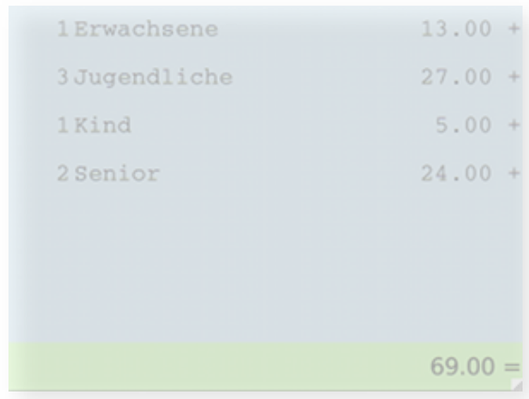




LISTEN „SALE“

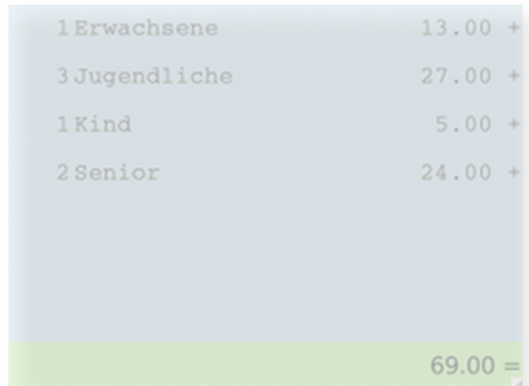
ECR changes data...





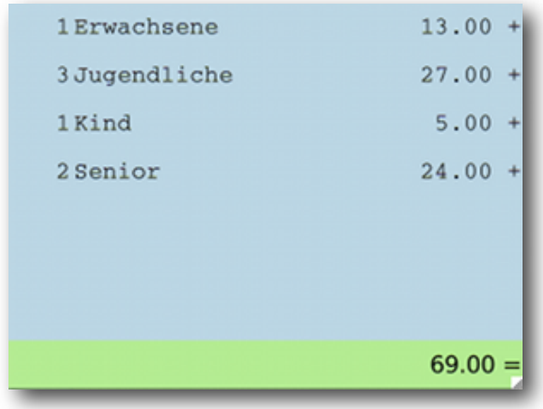
INSERT, UPDATE
DELETE

… DB change triggers a function



TRIGGER-FUNKTION

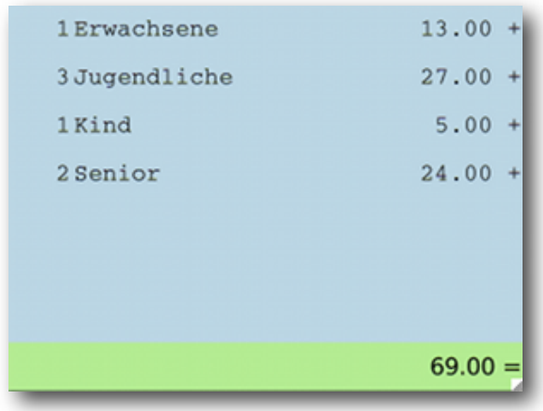
Tigger-function „reports“ Event



NOTIFY „SALE“

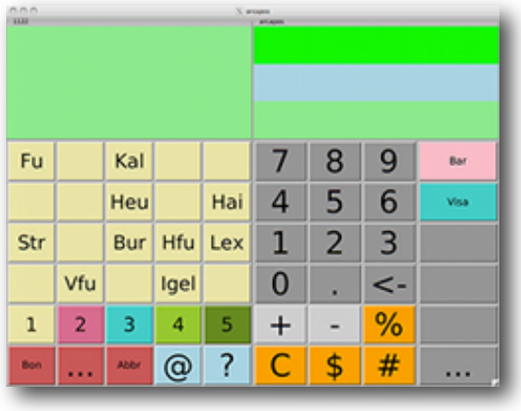
TRIGGER-FUNCTION

Display Shows New Data

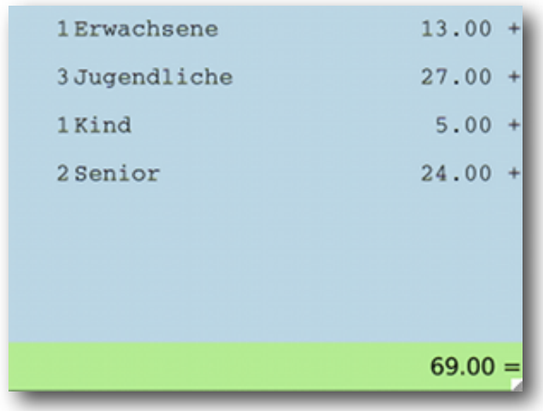


SELECT

Asynchronous Notifications



INSERT, UPDATE
DELETE



LISTEN, SELECT

TRIGGER: NOTIFY

Work the „Notifies“ Queue

```
PGnotify *pn;
if (PQstatus(conn) == CONNECTION_BAD)
   return;
PQconsumeInput(conn);
while (PQstatus(conn) == CONNECTION_OK
     && (pn = PQnotifies(conn)) != NULL) {
      ...
      pn->relname;
      ...
   PQfreemem(pn);
}
```

PostgreSQL and X Window

```
notify_input = XtAppAddInput(appctx,
    PQsocket(conn),
    (XtPointer)XtInputReadMask,
    async_notifies, conn);
```

PostgreSQL and X Window

```
static void
async_notifies(XtPointer client_data, int *source,
    XtInputId *id)
{
    PGconn *conn = (PGconn *)client_data;

    if (PQstatus(conn) == CONNECTION_BAD) {
        PQXtUnregisterAll(conn);
        return;
    }
    PQconsumeInput(conn);
    process_notifies(conn);
    if (PQstatus(conn) == CONNECTION_BAD)
        PQXtUnregisterAll(conn);
}
```

process_notifies()

```
static void
process_notifies(PGconn *conn)
{
     PGnotify *pn;
     struct notify *n;

     if (PQstatus(conn) == CONNECTION_BAD)
          return;
     PQconsumeInput(conn);
     while (PQstatus(conn) == CONNECTION_OK
         && (pn = PQnotifies(conn)) != NULL) {
          SLIST_FOREACH(n, &nhead, next) {
               if (!strcmp(n->name, pn->relname))
                    n->handler(n->name, n->user_data);
          }
          PQfreemem(pn);
     }
}
```

Queue with Event-Handlers

```
#include <sys/queue.h>

struct notify {
    SLIST_ENTRY(notify)next;
    char   *name;
    void   *user_data;
    void   (*handler)(char *name, void *user_data);
};
```

PqXtRegister()

```
int
PQXtRegister(XtAppContext app, PGconn *conn, char *name,
    void (*handler)(char *, void *), void *user_data)
{
    struct notify *n;
    ...
    n->user_data = user_data;
    n->handler = handler;
    SLIST_INSERT_HEAD(&nhead, n, next);
    PQclear(PQvexec(conn, "LISTEN \"%s\"", name));

    notify_input = XtAppAddInput(app, PQsocket(conn),
        (XtPointer)XtInputReadMask, async_notifies, conn);
```

Problem

Same socket is used for normal SQL operation and async notifications...

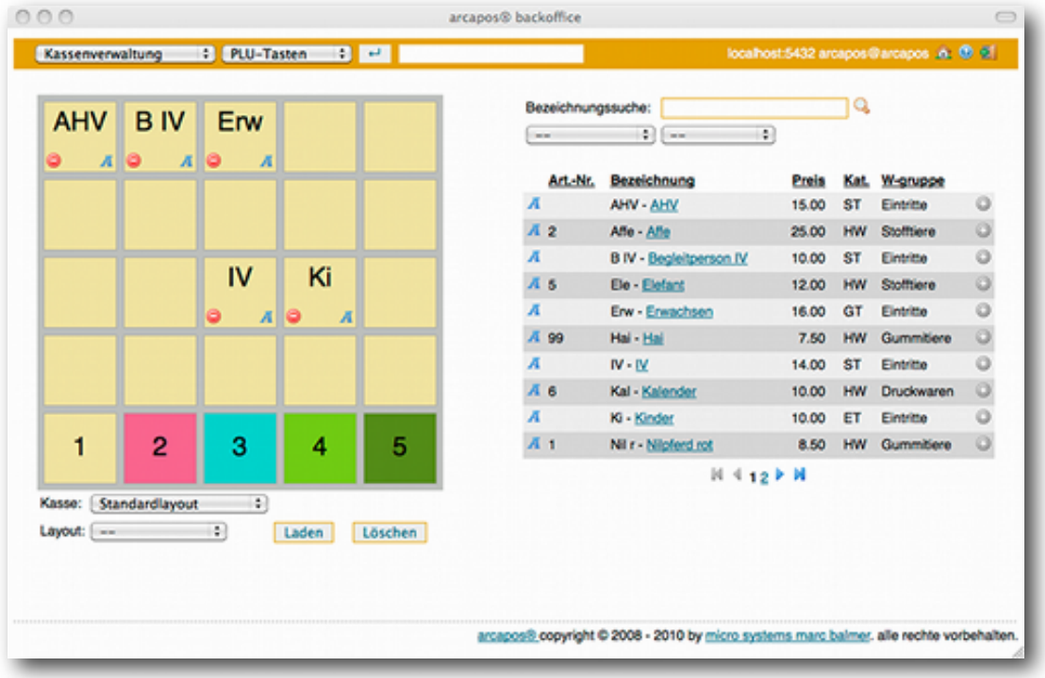
Turn of the X11 Input before sending an SQL command

Turn it back on after result is reveived

Idea

Use two connections, one for normal operation, one to LISTEN on

ECR – Backoffice



ECR LISTENs, too

A demonstration

Security

[To trust in god is not good enough…]

No Security on Notifications

Any user can send any message

Any user can listen to any message

Absolutely no rights to GRANT / REVOKE

An (Exploitable) Bug

```
scroll_stop();
res = PQvexec(conn, "SELECT max(id) FROM sat_sale "
        "WHERE terminal = '%s' AND state = 'S'", hostname);

sale = atol(PQgetvalue(res, 0, 0));

PQclear(PQvexec(conn, "LISTEN \"sale_upd:%ld\"", sale));
PQclear(PQvexec(conn, "LISTEN \"sale_del:%ld\"", sale));

update_sale();
```

An Exploitable Bug

```
void
async_notification(XtPointer client_data, int *source,
XtInputId *id)
{
     PGnotify *notify;
     char *p;

     PQconsumeInput(conn);
     while ((notify = PQnotifies(conn)) != NULL) {

          if (!strcmp(notify->relname, "sale_del"))
               clear_sale();
```

Don‘ t Trust Notifications

A demonstration

Security Remarks

Be careful with the payload feature („Payload-Injection-Attack“)

Use notifications only as a trigger, always access the database

DOS attacks?

Idea

Use the payload for a cryptographic hash (a shared secret)

UNLISTEN „Talk“

[Questions?]

In god we trust, in C we code!

Marc Balmer

marc@msys.ch, m@x.org,

mbalmer@NetBSD.org

www.msys.ch, www.arcapos.com