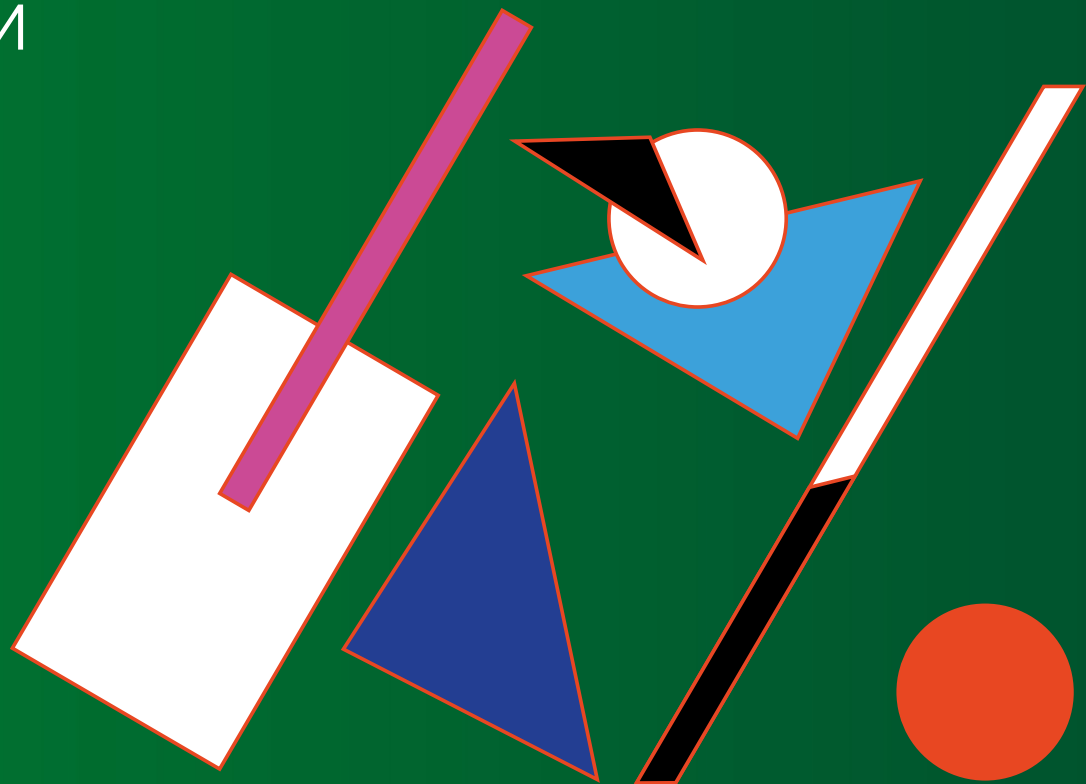


PGCONF.RUS SIA 2024

Как мы искали и
что мы нашли

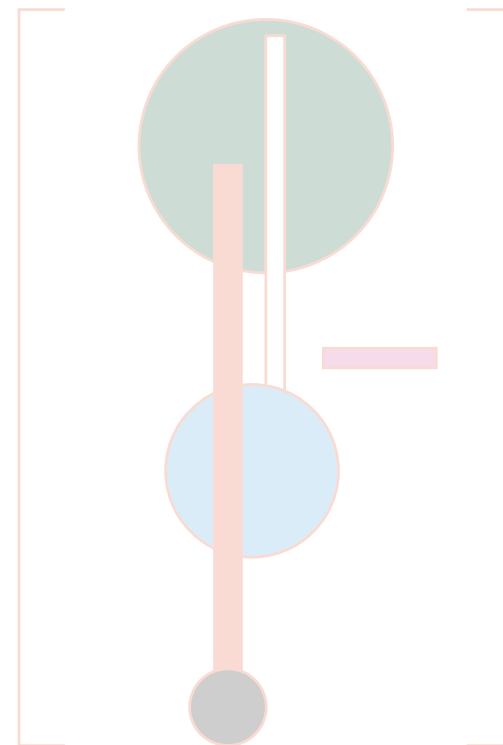
Николай Шаплов



Николай Шаплов

Отдел безопасной разработки

Подразделение фаззинга



PosgresPro SDL – Security Development Lifecycle

I. Планирование

...

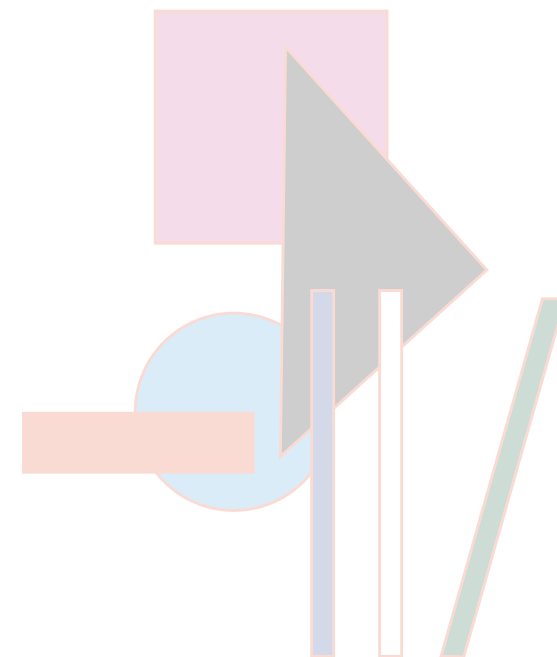
IV. Реализация

V. Верификация

- статический анализ;
- динамический анализ;
 - **фаззинг-тестирование.**

VI. Релиз

VII. Реагирование

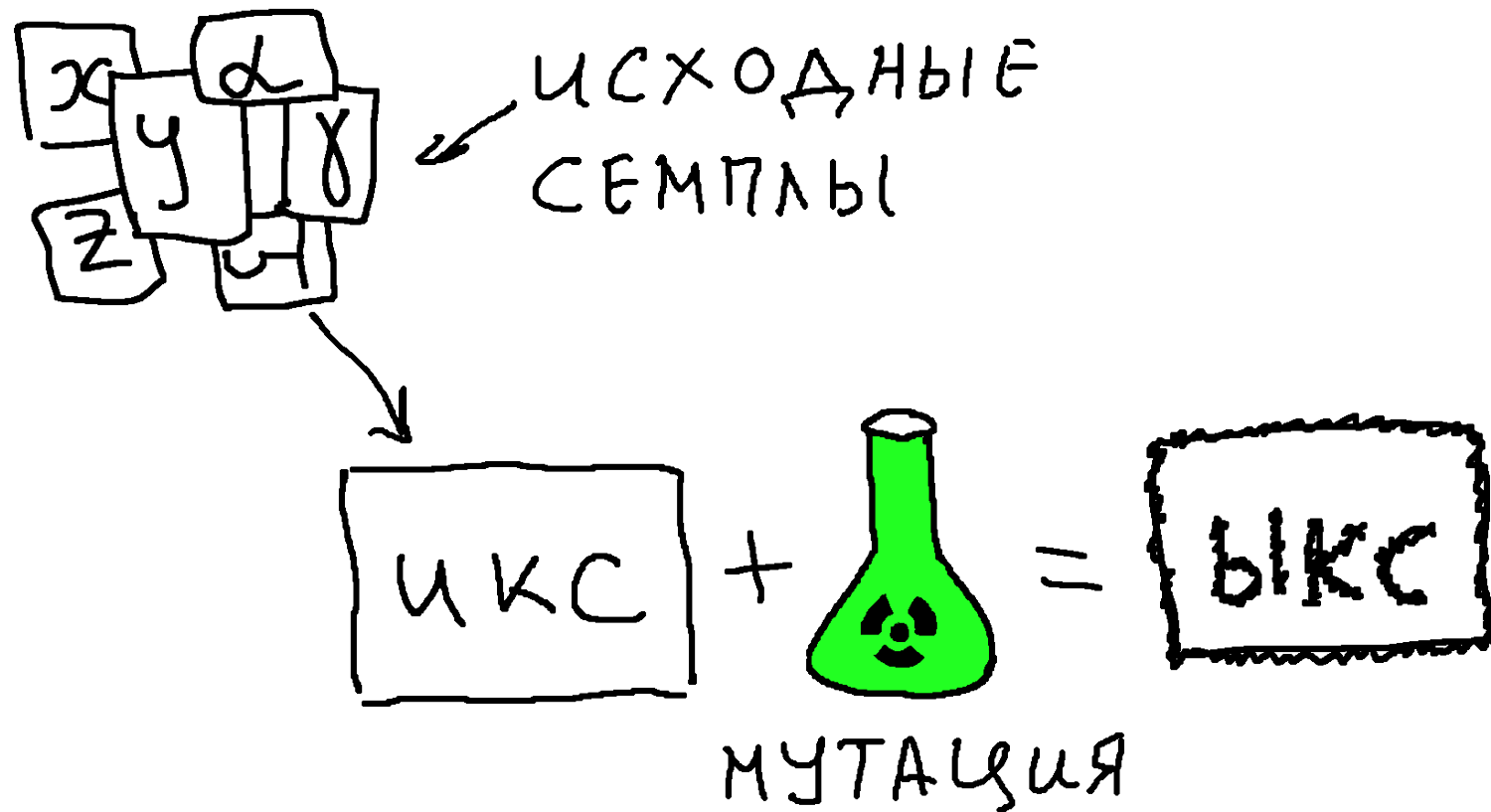


PosgresPro Фаззинг в двух словах

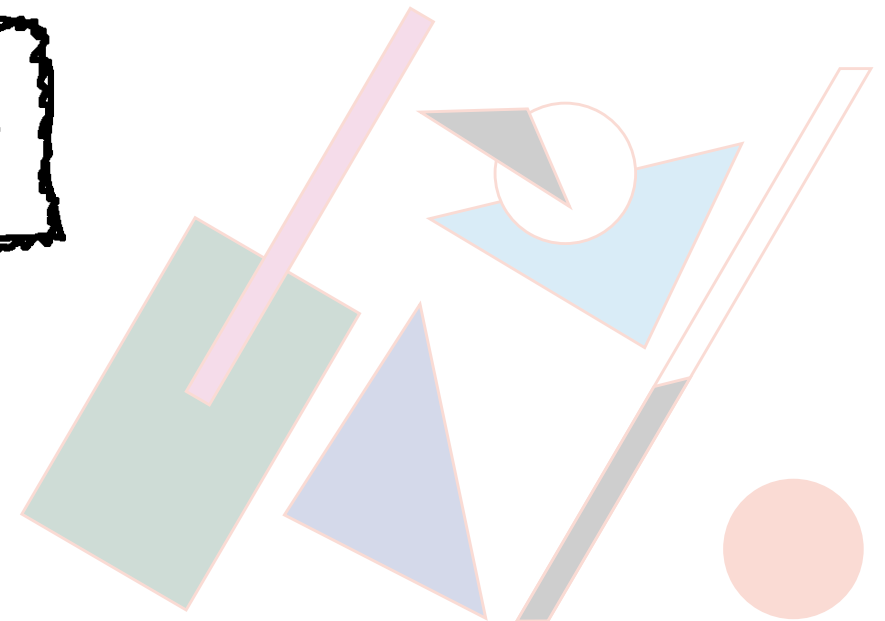
Поддаем на вход
системе случайные
(на самом деле вовсе не случайные)
данные и смотрим,
что получится



PosgresPro Фаззинг: подробности по ссылке

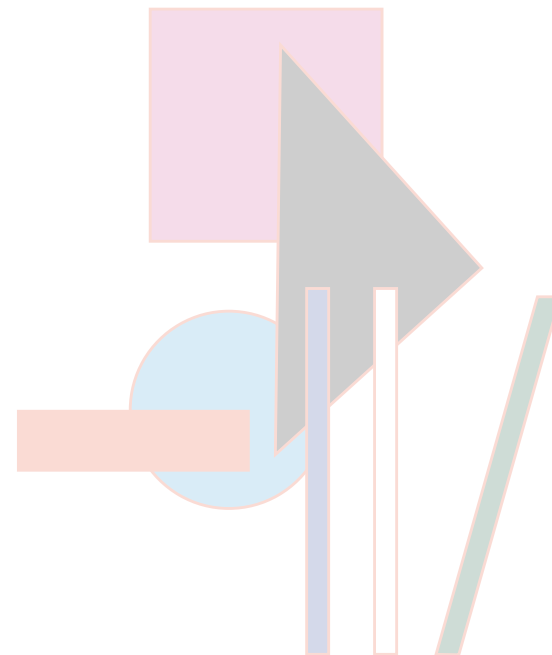


youtube.com/watch?v=fbhuzSpyUJc



PosgresPro План

- **Фаззинг input-функций;**
 - Создание цели через фаззинг-модуль;
- **Фаззинг op-функций;**
 - Structure-Aware фаззинг;
- **Фаззинг сетевого соединения;**
 - Создание цели через врезку;
 - Подмена сетевых POSIX функций;
 - Система восстановления дискового контекста.





Фаззинг input-функций



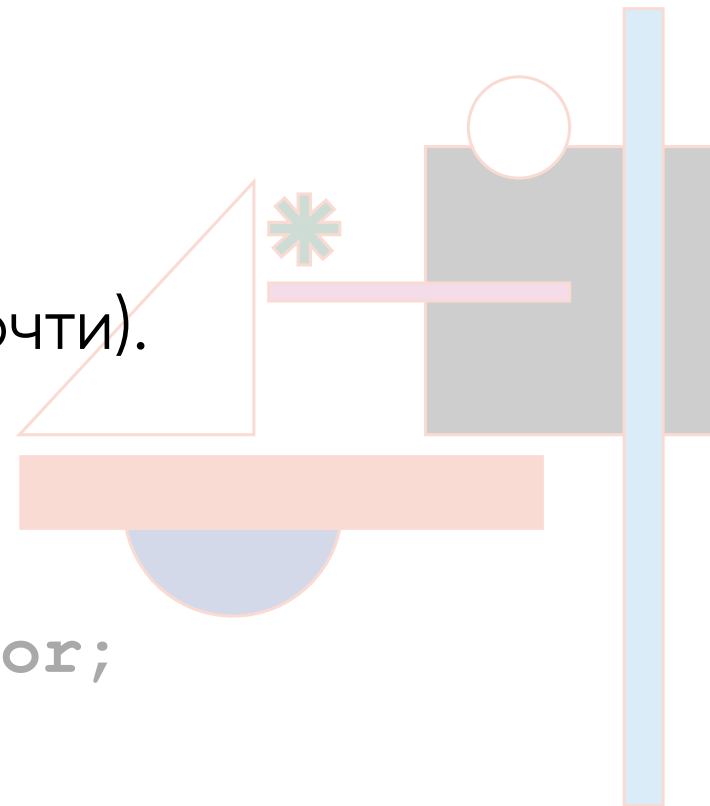
PostgresPro Input-функция

Input-функция – это pg-функция, занимающаяся преобразованием данных определенного типа из текстового представления во внутреннее.

- Простая;
- Входит в поверхность атаки;
- Стандартизованный интерфейс;
- Отчуждаемая от остального Постгреса (ну почти).

```
SELECT 'a:1A fat:2B,4C cat:5D'::tsvector;
```

Внутри вызывается `tsvectorin()`



PostgresPro Создание цели в форме фаззинг-модуля

- Берем исследуемую функцию;
- Линкуем к ней пол-Постгреса;
- Минимально инициализируем окружение;
- Говорим фаззеру «Фас!»



PostgresPro Создание цели в форме фаззинг-модуля

#

```
#include "postgres.h"
#include "utils/memutils.h"
//....
int main(int argc, char* argv[])
{
    Datum out;
    MemoryContextInit();
    while (__AFL_LOOP(4000))
    {
        char * in = slurp_file(argv[1]);
        out = DirectFunctionCall1Coll(tsvectorin,
                                     InvalidOid, CStringGetDatum(in));
        //... освобождаем память
    }
}
```

PostgresPro Создание цели в форме фаззинг-модуля

#

```
#include "postgres.h"
#include "utils/memutils.h"
//....
int main(int argc, char* argv[])
{
    Datum out;
    MemoryContextInit();
    while (__AFL_LOOP(4000))
    {
        char * in = slurp_file(argv[1]);
        out = DirectFunctionCall1Coll(tsvectorin,
                                     InvalidOid, CStringGetDatum(in));
        //... освобождаем память
    }
}
```

PosgresPro Создание цели в форме фаззинг-модуля

```
#include "postgres.h"
#include "utils/memutils.h"
//....
int main(int argc, char* argv[])
{
    Datum out;
    MemoryContextInit();
    while (__AFL_LOOP(4000))
    {
        char * in = slurp_file(argv[1]);
        out = DirectFunctionCall1Coll(tsvectorin,
                                     InvalidOid, CStringGetDatum(in));
        //... освобождаем память
    }
}
```

PosgresPro Создание цели в форме фаззинг-модуля

```
#include "postgres.h"
#include "utils/memutils.h"
//....
int main(int argc, char* argv[])
{
    Datum out;
    MemoryContextInit();
    while (__AFL_LOOP(4000))
    {
        char * in = slurp_file(argv[1]);
        out = DirectFunctionCall1Coll(tsvectorin,
                                     InvalidOid, CStringGetDatum(in));
        //... освобождаем память
    }
}
```

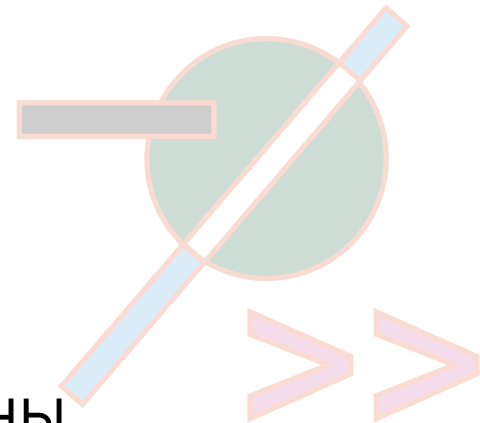
PosgresPro Создание цели в форме фаззинг-модуля

```
#include "postgres.h"
#include "utils/memutils.h"
//....
int main(int argc, char* argv[])
{
    Datum out;
    MemoryContextInit();
    while (__AFL_LOOP(4000))
    {
        char * in = slurp_file(argv[1]);
        out = DirectFunctionCall1Coll(tsvectorin,
                                     InvalidOid, CStringGetDatum(in));
        //... освобождаем память
    }
}
```

PosgresPro Создание цели в форме фаззинг-модуля

```
#include "postgres.h"
#include "utils/memutils.h"
//....
int main(int argc, char* argv[])
{
    Datum out;
    MemoryContextInit();
    while (___AFL_LOOP(4000))
    {
        char * in = slurp_file(argv[1]);
        out = DirectFunctionCall1Coll(tsvectorin,
                                     InvalidOid, CStringGetDatum(in));
        //... освобождаем память
    }
}
```

PostgresPro Что нашлось: \u00000 в jsonb

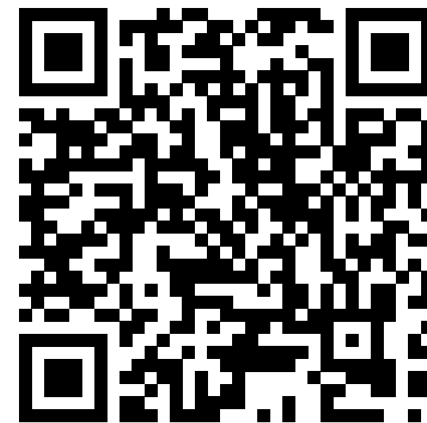


В PostgreSQL 14 и 15,
копирование блока памяти отрицательной длины
при указании нулевого символа через \u в типе jsonb

```
SELECT E'\n"\u00000"'::jsonb;
```

[https://www.postgresql.org/message-id/flat/](https://www.postgresql.org/message-id/flat/7332649.x5DLKWyVIX%40thinkpad-pgpro)

7332649.x5DLKWyVIX%40thinkpad-pgpro



PostgresPro Что нашлось: долгий `jsonpath`

Создание переменной типа `jsonpath` со специально подобранным значением занимает более минуты:

```
SELECT E'$ ? (@ like_regex\x0a"pat\x5c\x5cdbegex \x5c[^aguegex \x5c[^agu0424
ch[[[=V]=]=====arJ\x5c\x5cdbegex \x5c\x5cD0\x5c\x5c\x5ct\x5c\x5c\x07--1)*]dbegex \
\x5c[^agu0424 ch[[[=V]=]=====arW\x5c\x5cdbegex \x5c\x5cD0\x5c\x5c\x5ct\x5c\x5c\x07--
1)0424 ch[[[=V]=]===-\x5c\x5c\x5cU2222222bi$\xd1$a \xd1a[U2222222bi$ 1-\x5c\x5c\
\x5cU2222222bi$\xd1$ u0424 ch[[[=V]=]=====arW\x5c\x5cdbegex \x5c\x5cD0\x5c\x5c\x5ct\
\x5cg\x07--1)0424 ch[[[=V]=]===-\x5c\x5c\x5cU2222222bi$\xd1$a \xd1a[U2222222bi$ 1-\x5c\
\x5c\x5cU2222222bi$\xd1$ (@a 1-\x5c\x5c\x5cU2222222222bi222bi$\xd1a[$a 3\x03\xe8 1,
($b[*\x5c\x5c\x5cU2222222222bi22$Ma[$a *\x5c\x5c\x5cU2222222bi$ 1-\x5c\x5c\
\x5cU2222222bi@\xd1a[U22]\x149\x5c($).i=====+? (@ like_0*1e-1\x5c\x5c\x5c\x5c\
\x07--1)*]dbegex \x5c[^agu0424 ch[[[=V]=]=====ar(@a 1-\x5c\x5c\
\x5cU2222222222bi222bi$\xd1a[$a 3\x03\xe8 1, ($b[*\x5c\x5c\x5cUMa[$a *\x5c\x5c\
\x5cU2222$2222222222bi22222bi$ 1-\x5c\x5c\x5cU2222222bi@\xd1a[U22]\x149\
\x5c($).i=====arJ\x5c\x5cdbegex \x5c\x5cD0\x5c\x5c\x5c\x5c\x07--1)*]dbegex \
\x5c[^agu0424 ch[[[=V]=]=====arW\x5c\x5cdbegex \x5c\x5cD0^^^^^^)*^..b\xff\
\xff*|.^^^^($)\xef\xffnePn" flag "is"\x5c\x5cct\x5c\x5c\x07--1)*]" flag
"i")'::jsonpath;
```

В работе у группы производительности



PostgresPro Баг в jsonpath/regex

[illegible]

PostgresPro Бағ в jsonpath/regex

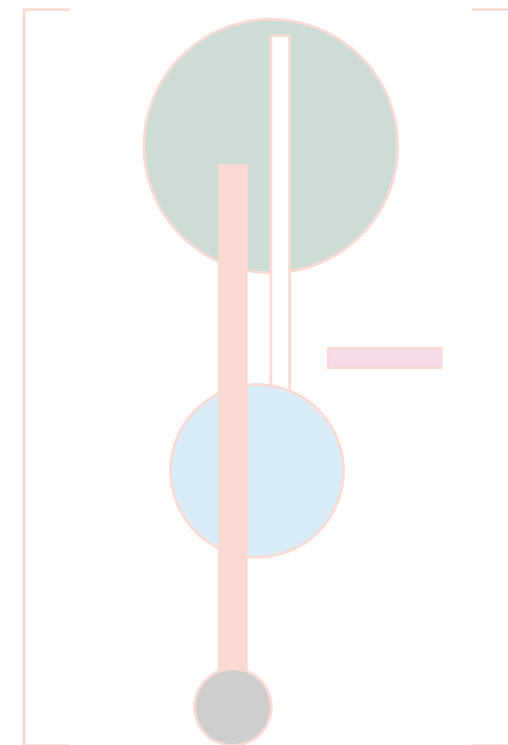
Assert ("lp->nouts == 0 && rp->nins == 0")

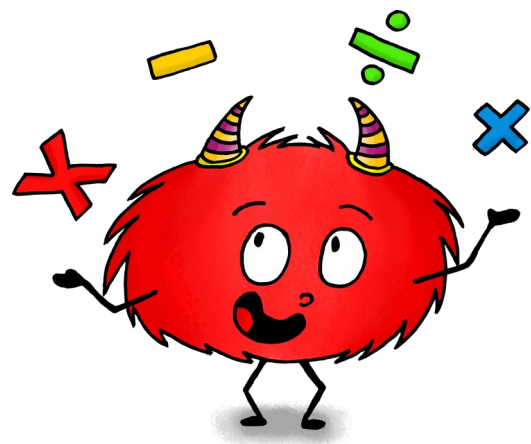
PostgresPro Баг в jsonpath/regex

```
SELECT ' ' ~ ' (? : [ ^ \d \D ] ) { 0 } ' ;
```

[BUG #18708](#)

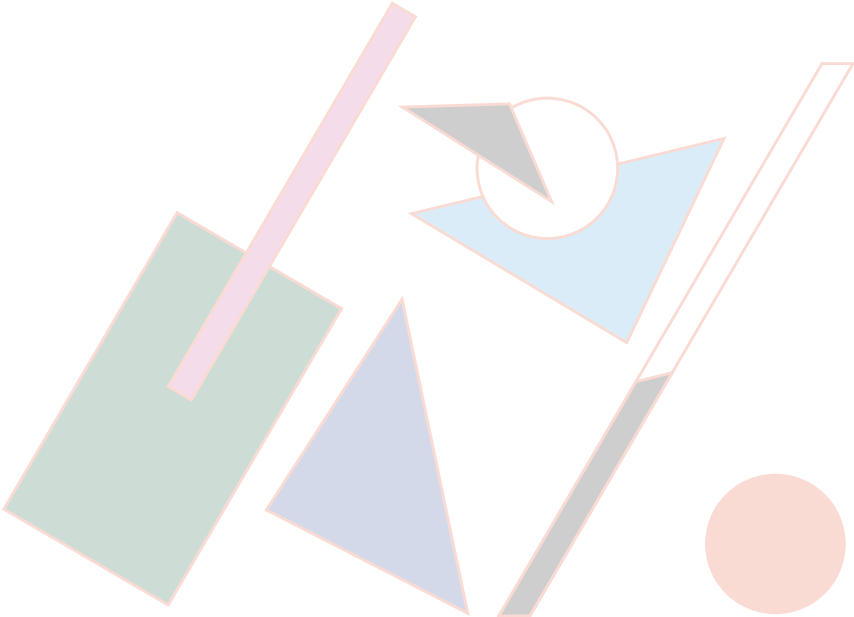
Fixed in [b69bdce](#)





⋈⋈⋈ ⋈⋈ ⋈⋈ ⋈⋈ ⋈ = 34·60+25-2065.

Фаззинг операций



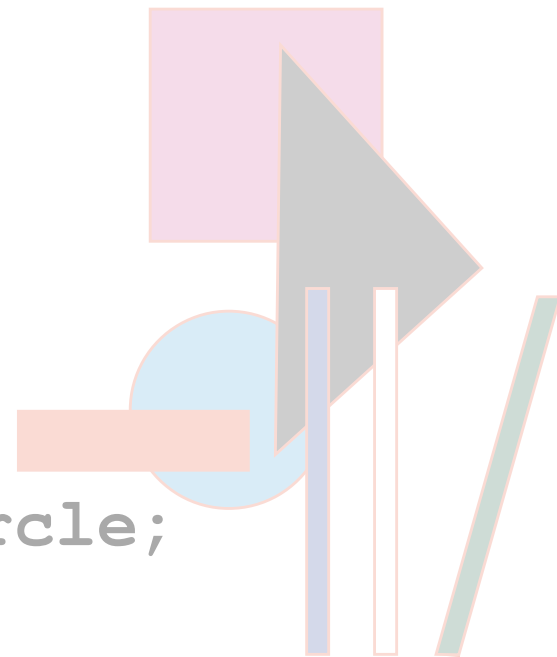
PostgresPro Фаззинг операций

Ор-функция – это рд-функция, реализующая определенную операцию, совершаемую над операндами заданного типа.

- Простая;
- Входит в поверхность атаки;
- Стандартизованный интерфейс;
- Отчуждаемая от остального Постгреса.

```
SELECT '<(0,0),1>'::circle <->  
                                '<(5,0),1>'::circle;
```

Внутри вызывается `circle_distance()`



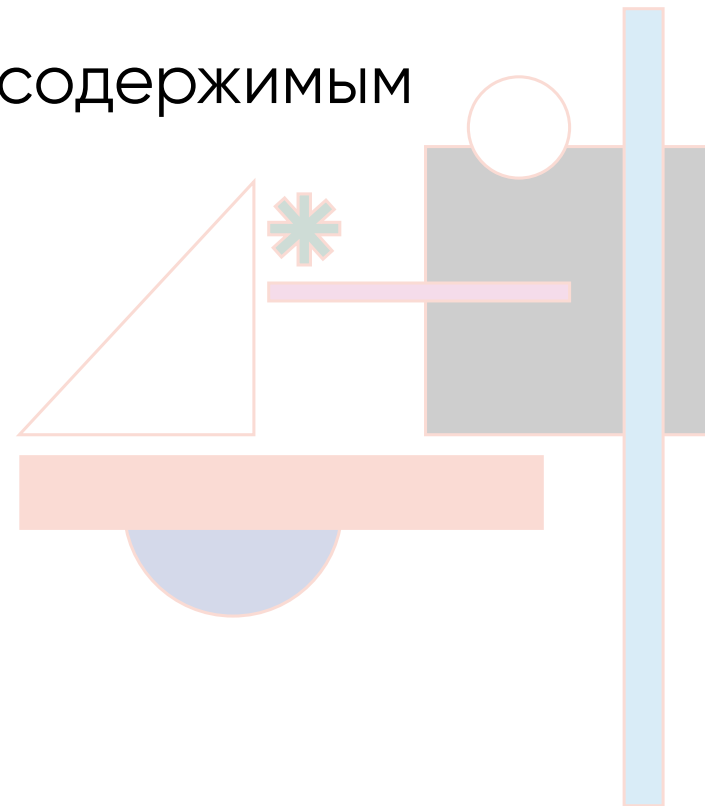
PosgresPro Фаззинг операций

Главная проблема – фаззить операцию, а не операнды.

```
SELECT '<(0, z), 1>'::circle <-> '<(5, 0), 1>'::circle;
```

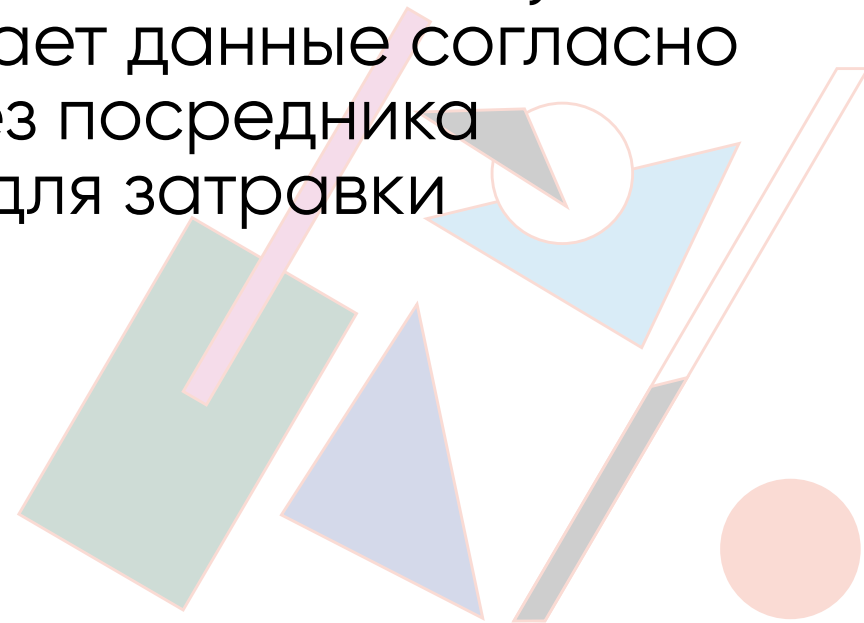
Ломает input-функцию, а до ор- дело не доходит.

Нужны синтаксически корректные аргументы, с содержимым находящимся под управлением фаззера.



Это работа для **Structure-Aware Fuzzing'a!**

SAF — это когда есть какие-то требования к синтаксису или структуре входных данных и фаззер выдает данные согласно этим требованиям. Сам, или через посредника использующие данные фаззера для затравки

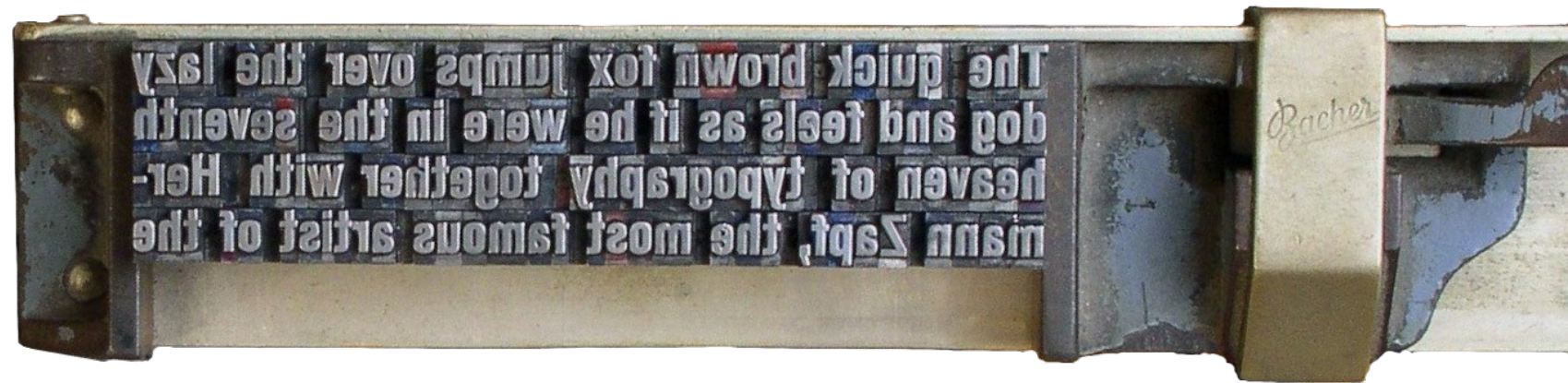
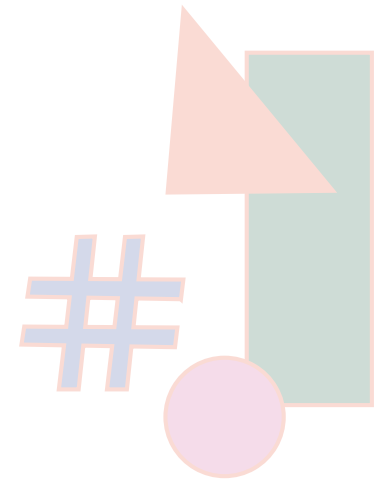


PostgresPro Structure Aware Fuzzing (SAF)

Библиотека **LibBlobStamper**

Фреймворк для создания штампов

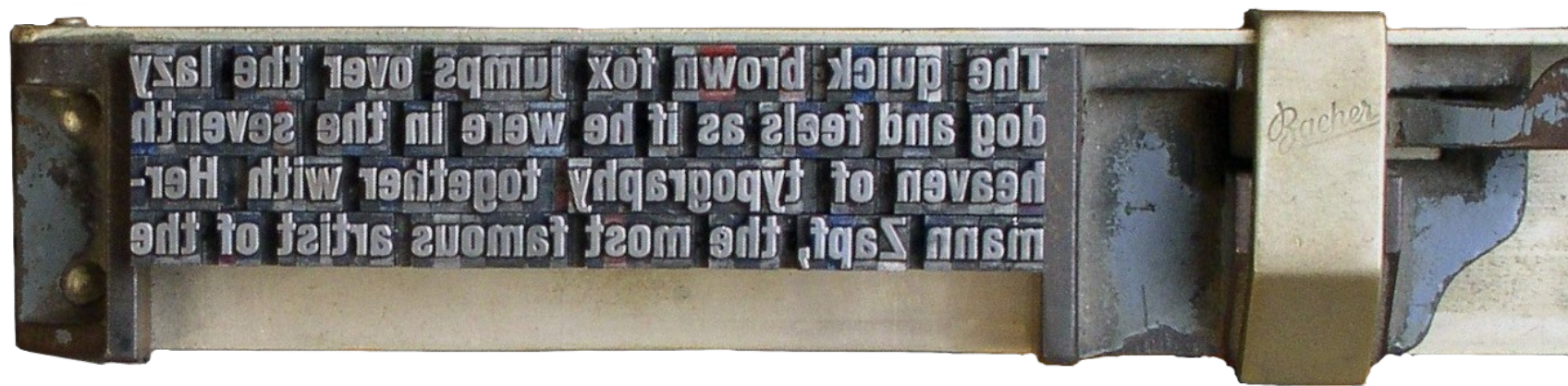
<https://github.com/postgrespro/libblobstamper>



PosgresPro Structure Aware Fuzzing (SAF)

Штамп потребляет «фаззинину» (**Pulp Data**) и на ее основании создает структурированные данные.

Фаззер, как раньше, управляет входными данными не понимая их смысла. Но исследуемой функции подается синтаксически корректный ввод.



В PostgreSQL начиная с 14, попытка вычислить вхождение одного полигона в другой (@>), если оба полигона содержат узлы в нуле или бесконечности, приводит к вычислениям сложности $O(n!)$

```
select '((-inf, 0), (0, inf), (-inf, 0), (0, inf), (0, 0), (0, 0), (0, 0),  
(0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0),  
(0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0),  
(0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0),  
(0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0),  
(0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0),  
(0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0),  
(0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0),  
(-inf, 0))'::polygon @> '((0, 0), (0, 0),  
(0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0),  
(0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0),  
(0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0),  
(0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0),  
(0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0),  
(-inf, 0))'::polygon;
```

Запрос выполняется больше суток

Что нашлось: `poly_contain`, из 0 в бесконечность

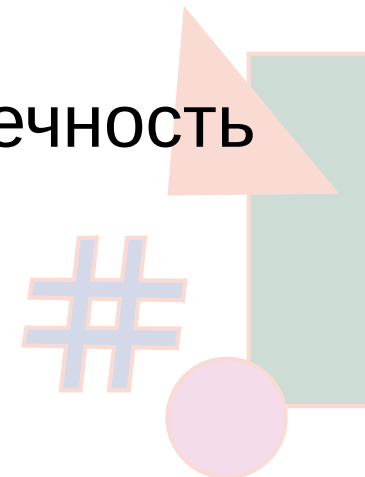


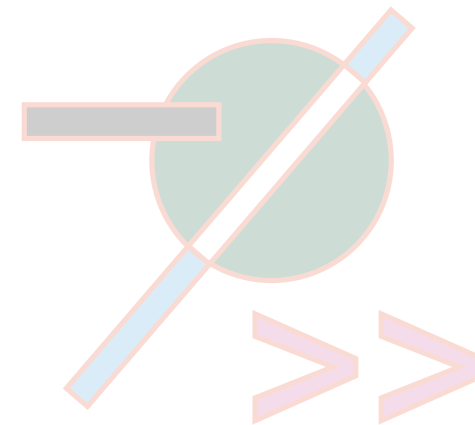
Баг зарепорчен в upstream:

[BUG #18214](#)

Похожая проблема в PostgreSQL 11:

[BUG #17962](#)





Фаззинг сетевой подсистемы libpq



PosgresPro Фаззинг libpq

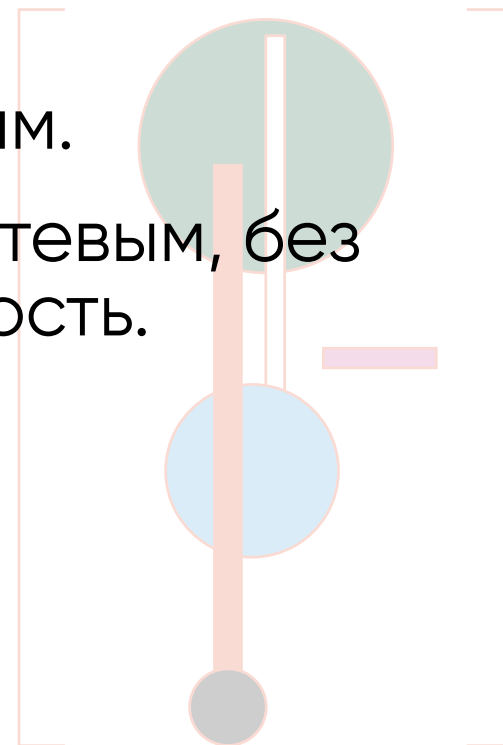
Сетевое соединение входит в поверхность атаки by default.

В реальности:

- Завернуто в SSL
- Спрятано за фильтрами и firewall'ами

Но все равно обязано быть максимально безопасным.

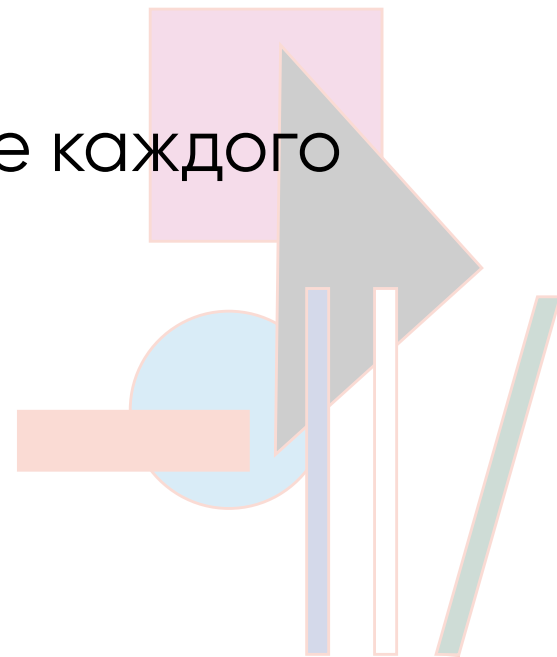
В данном исследовании мы работаем с «чистым» сетевым, без вышележащих уровней обеспечивающих безопасность.



PostgresPro Фаззинг libpq

Что нужно для фаззинга сетевого соединения:

- Подменить сетевой обмен на выхлоп фаззера
- Запустить PostgreSQL с сетевой подсистемой в один процесс
 - Избавиться от backend'ов
 - Не запускать worker'ы
- Восстанавливать контекст (содержимое БД) после каждого цикла фаззинга





Фаззинг сетевой подсистемы через подмену POSIX-вызовов



PosgresPro Фаззинг сетевого соединения

Суть проблемы:

Фаззер не умеет направлять данные в сокет, только в файл.

Сопутствующие сложности:

Посредники вроде netcat нас не устраивают.

(Фаззеру нужна тесная интеграция с исполняемой программой)

Решение:

Хардкодим вместо получения данных из сокета получение данных из фаззера через файл.



PostgresPro Фаззинг сетевого соединения

Сопутствующие сложности:

Придется отрывать всю установку соединения.

Миссия не выполнима:

- 5 версий PostgreSQL => 5 разных сложных патчей
- Новый год => новая версия => новый патч

Мы не хотим тратить всё время на портирование фаззинга на новые версии.

Можно как-то менее инвазивно?





PostgresPro Фаззинг сетевого соединения

А может проще создать поддельную сетевую подсистему?

Пожалуй проще. Для PostgreSQL она всего лишь набор POSIX-ВЫЗОВОВ.

Плюсы:

Код подставной сетевой подсистемы изолирован, PostgreSQL продолжает работать с ней как обычно, с минимальными правками кода.

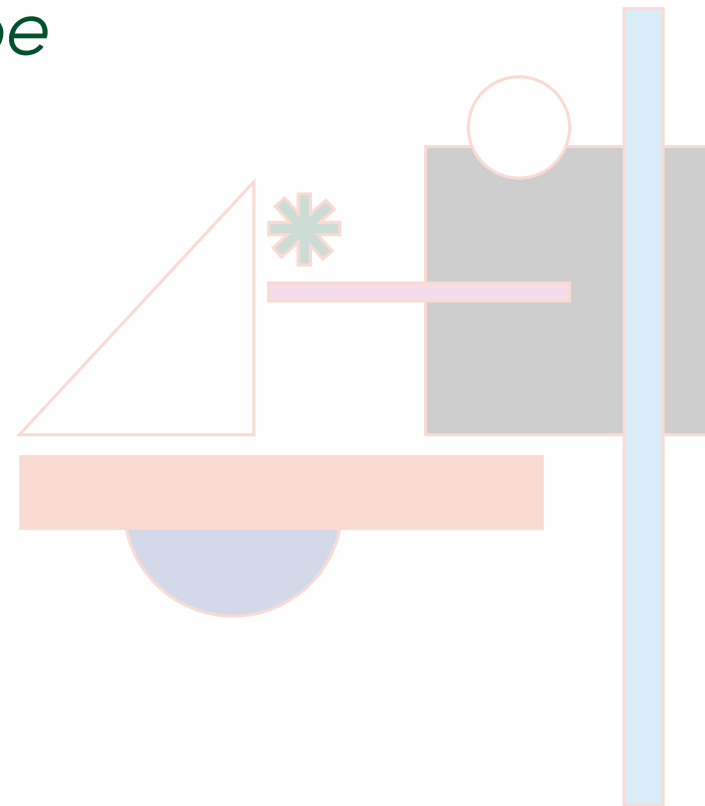
(Сказал я в 2023м году и тут же `select()` заменили на `epoll_wait()`)



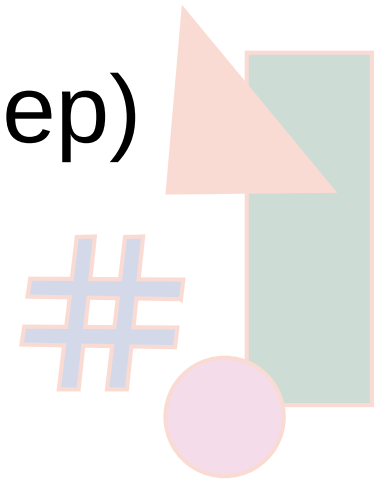


Пример подмены POSIX-вызова

общая схема на примере
`getchar()`

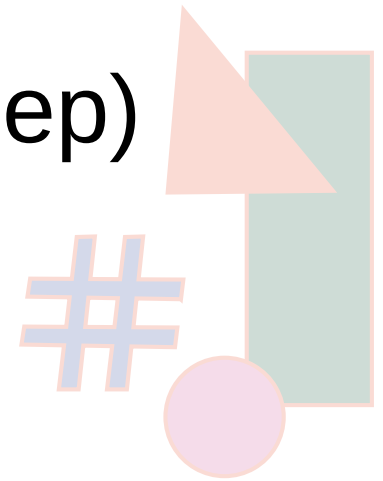


PosgresPro Подмена системного вызова (пример)



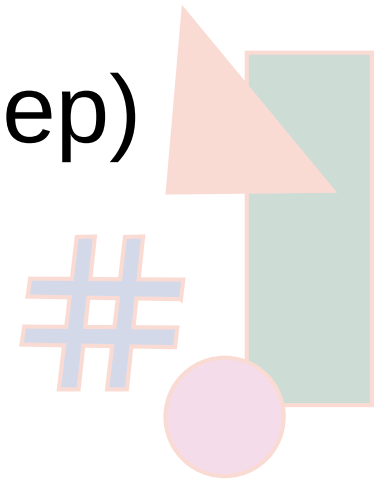
```
int (*original_getchar)();
original_getchar = dlsym(RTLD_NEXT, "getchar");
char *fake_input = "fake_input_buf";
int getchar()
{
    if(!getenv("FAKE_IT")) return original_getchar();
    if (fake_input[0] == '\\0') return EOF;
    fake_input++;
    return(fake_input[-1]);
}
```

PosgresPro Подмена системного вызова (пример)



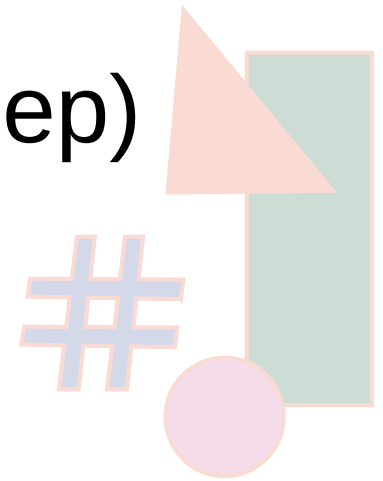
```
int (*original_getchar)();  
original_getchar = dlsym(RTLD_NEXT, "getchar");  
char *fake_input = "fake_input_buf";  
int getchar()  
{  
    if(!getenv("FAKE_IT")) return original_getchar();  
    if (fake_input[0] == '\\0') return EOF;  
    fake_input++;  
    return(fake_input[-1]);  
}
```

PosgresPro Подмена системного вызова (пример)



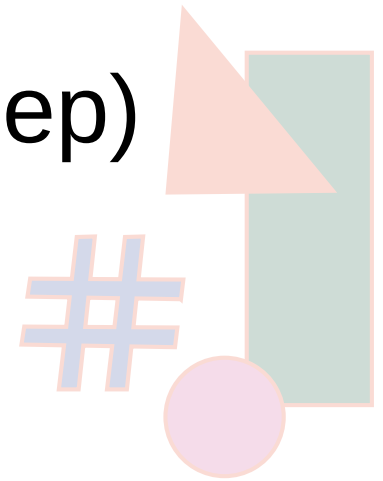
```
int (*original_getchar)();  
original_getchar = dlsym(RTLD_NEXT, "getchar");  
char *fake_input = "fake_input_buf";  
int getchar()  
{  
    if(!getenv("FAKE_IT")) return original_getchar();  
    if (fake_input[0] == '\\0') return EOF;  
    fake_input++;  
    return(fake_input[-1]);  
}
```

PosgresPro Подмена системного вызова (пример)



```
int (*original_getchar)();  
original_getchar = dlsym(RTLD_NEXT, "getchar");  
char *fake_input = "fake_input_buf";  
  
int getchar()  
{  
    if(!getenv("FAKE_IT")) return original_getchar();  
    if (fake_input[0] == '\\0') return EOF;  
    fake_input++;  
    return(fake_input[-1]);  
}
```

PosgresPro Подмена системного вызова (пример)



```
int (*original_getchar)();  
original_getchar = dlsym(RTLD_NEXT, "getchar");  
char *fake_input = "fake_input_buf";  
int getchar()  
{  
    if(!getenv("FAKE_IT")) return original_getchar();  
    if (fake_input[0] == '\0') return EOF;  
    fake_input++;  
    return(fake_input[-1]);  
}
```

PosgresPro Подмена системного вызова (пример)

#

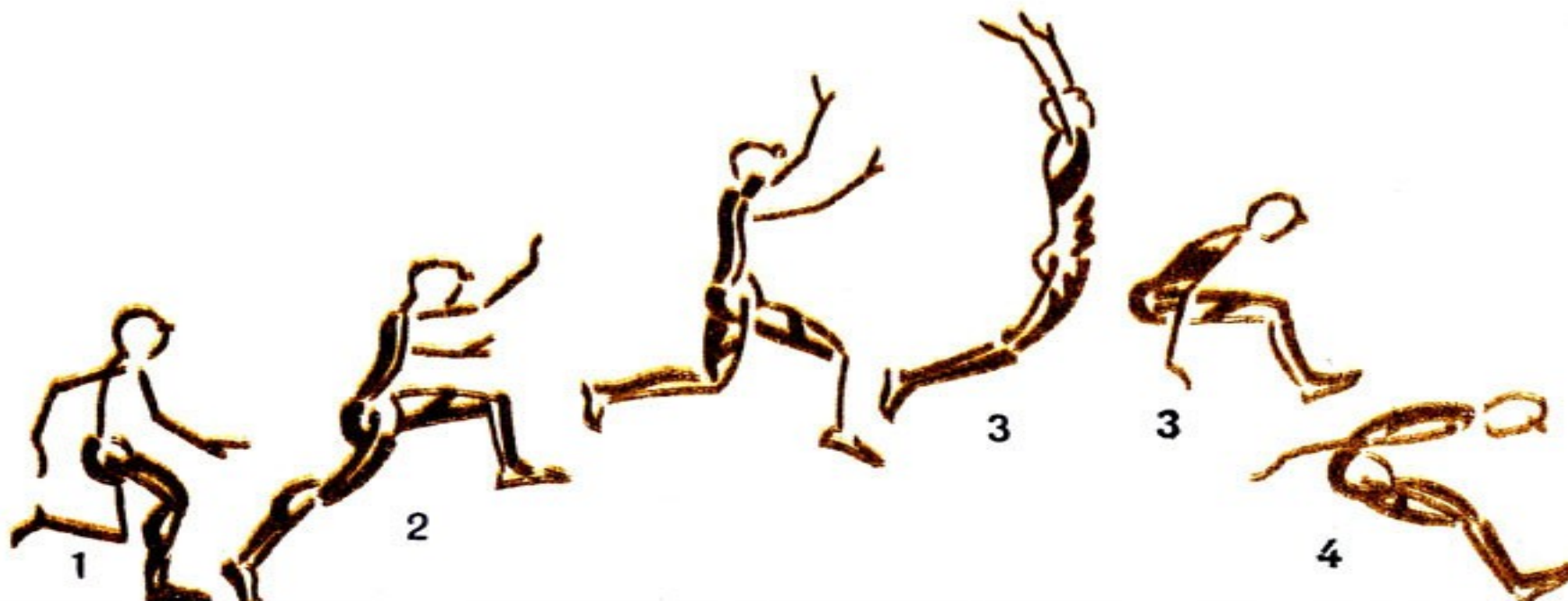
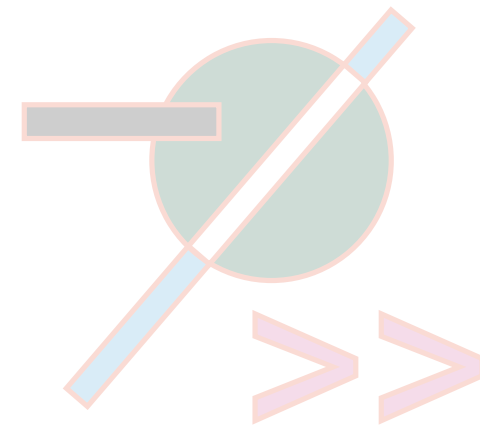
```
int (*original_getchar)();  
original_getchar = dlsym(RTLD_NEXT, "getchar");  
char *fake_input = "fake_input_buf";  
int getchar()  
{  
    if(!getenv("FAKE_IT")) return original_getchar();  
    if (fake_input[0] == '\\0') return EOF;  
    fake_input++;  
    return(fake_input[-1]);  
}
```

PosgresPro Подмена системного вызова (пример)

#

```
int (*original_getchar)();  
original_getchar = dlsym(RTLD_NEXT, "getchar");  
char *fake_input = "fake_input_buf";  
int getchar()  
{  
    if(!getenv("FAKE_IT")) return original_getchar();  
    if (fake_input[0] == '\\0') return EOF;  
    fake_input++;  
    return(fake_input[-1]);  
}
```

Этапы установки соединения и их подмена



Шаги установки сетевого соединения

- `socket()`
- `bind()`
- `listen()`
- `select()` / `epoll_wait()`
- `accept()`
- `recv()` / `send()`

`fork()`

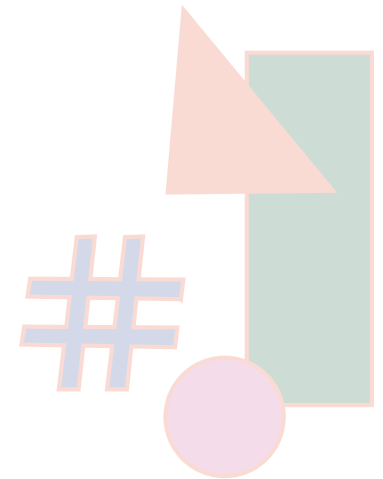
Подменяем все кроме `socket()` и `listen()` на свой код. Можем жить полностью автономно, можем вызывать оригинальные вызовы, но модифицировать поведение.



Режимы работы ложной сети

- Захват
- Воспроизведение
- Подмена

В режиме «Захват» собираем образцы, в режиме «Воспроизведение» подкладываемых выхлоп фаззера вместо нужного пакета. «Подмена» хороша для ручных исследований.



PostgresPro Что нашлось

Theodor Larionov

В PostgreSQL 13, переполнение при создании отрицательного месячного интервала

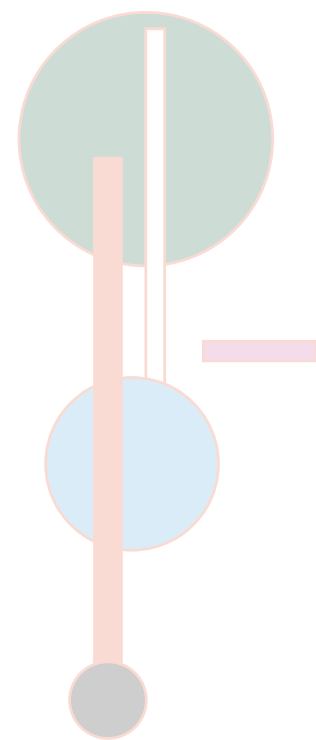
```
SELECT * from TO_CHAR(interval '-1Mon', 'rm');
```

[BUG #16953](#)





**Запуск в один
процесс с
сохранением
сетевой
подсистемы**



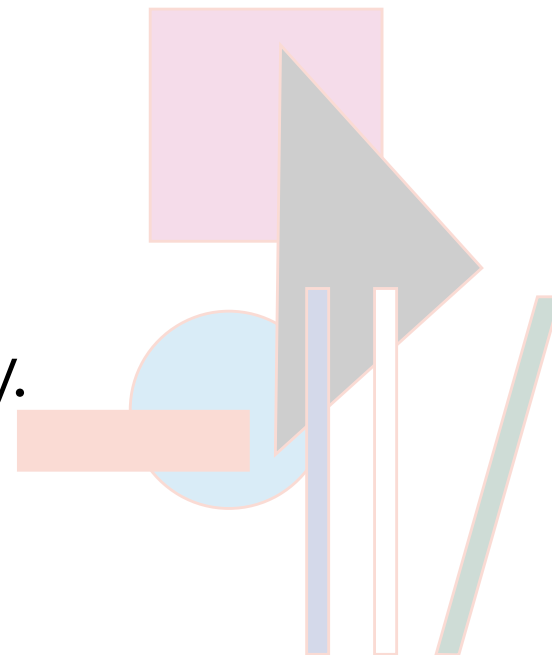
PostgresPro Запуск в один процесс

Суть проблемы:

- Фаззер не работает с многопроцессными системами;
- Single Mode не подходит, нужна рабочая сетевая подсистема.

Нормальный режим:

- Запускается PostMaster;
- Приходит входящий запрос;
- PostMaster от-fork'ивает от себя Backend;
- Backend отвечает на запрос и завершает работу.



PostgresPro Запуск в один процесс

Решение:

- А зачем нам Backend? Пусть PostMaster сам отвечает на запрос.
- Так он же после этого завершит работу весь целиком!
- Так и цикл фаззинга у нас на этом закончится.



PostgresPro Запуск в один процесс

```
--- a/src/backend/postmaster/postmaster.c
+++ b/src/backend/postmaster/postmaster.c
@@ -4295,7 +4295,8 @@ BackendStartup(Port *port)
     #ifdef EXEC_BACKEND
         pid = backend_forkexec(port);
     #else
         /* !EXEC_BACKEND */
-        pid = fork_process();
+        pid = 0;
         if (pid == 0)
             /* child */
         {
             free(bn);
         }
     }
 }
```

И отключить в паре мест проверку живости PostMaster'a



Background worker'ы тоже не нужны



PostgresPro Отключаем Background Worker'ы

```
--- a/src/backend/postmaster/postmaster.c
+++ b/src/backend/postmaster/postmaster.c
@@ -6304,6 +6306,14 @@ StartChildProcess(AuxProcType type)
{
    pid_t      pid;

+    if (type == BgWriterProcess ||
+        type == CheckpointerProcess ||
+        type == WalWriterProcess)
+        return -1;
#ifdef EXEC_BACKEND
    char        *av[10];
```



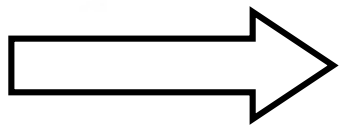
PostgresPro Отключаем Background Worker'ы

И больше не посылаем им сигналы!

```
--- a/src/backend/postmaster/postmaster.c
+++ b/src/backend/postmaster/postmaster.c
@@ -4589,6 +4589,8 @@ PostmasterStateMachine(void)
     static void
     signal_child(pid_t pid, int signal)
     {
+         if (pid == -1)
+             return; /* Do nothing for disabled workers! */
         if (kill(pid, signal) < 0)
             elog(DEBUG3, "kill(%ld,%d) failed: %m", (long) pid, si
#ifdef HAVE_SETSID
```

Иначе оно при остановке заберет с собой всю систему!

Восстановление ДИСКОВОГО КОНТЕКСТА





Используем возможности btrfs

PostgresPro КОНТАКТЫ

Николай Шаплов

Postgres Professional

E-mail: n.shaplov@postgrespro.ru

Matrix: [@dhyan:nataraj.su](https://matrix.to/#/@dhyan:nataraj.su)

