SIO2Jail

Wojciech Dubiel
Tadeusz Dudkiewicz
Przemysław Jakub Kozłowski
Maciej Wachulec

Motywacja

Konkursy algorytmiczne wymagają specyficznego środowiska do uruchamiania zgłaszanych programów

- Pomiar czasu / złożoności obliczeniowej
  - Najlepiej deterministyczny
- Zapewnienie powtarzalnego wykonania
- Sandbox zabezpieczający przed
  - naruszeniem zasad – np. tworzeniem wielu wątków, obejściem limitu pamięci
  - ingerencją w system sprawdzający – np. czytanie pliku zawierającego poprawne rozwiązanie, modyfikacja wyniku pomiaru czasu
  - uzyskaniem kontroli nad serwerem sprawdzającym i wykorzystaniem do dalszych ataków

Motywacja

Obecne rozwiązanie – `oitimetool`:

- skupia się głównie na mierzeniu czasu, sandboxowanie traktuje drugorzędnie
- przepisuje kod maszynowy w momencie pierwszego wykonania
- używa do tego biblioteki PIN
  - własnościowa biblioteka Intela do instrumentacji kodu
  - nie działa z nowymi kernelami
  - z założenia nie służy do uruchamiania niezaufanego kodu
- program zawodnika jest w tej samej przestrzeni adresowej co oitimetool
  - trudno odróżnić błąd programu zawodnika od błędu sprawdzaczki
  - prowadzi to do podatności pozwalających na ucieczkę
- sandboxowanie wyłącznie przez blokowanie syscalli
  - mocno związane z implementacją biblioteki standardowej
  - ma zhardcodowaną politykę
  - trudno dodać wsparcie dla nowych języków

Dostępne technologie

- perf
- seccomp-bpf
- ptrace
- namespaces
  - mount namespaces
  - PID namespaces
  - UTS namespace
  - IPC namespace
  - user namespace
- rlimit
- cgroup

Dostępne technologie – perf

Nowoczesne procesory mają liczniki sprzętowe, zliczające:

- cykle w których procesor nie spał
- wykonane instrukcje
- odwołania do pamięci
- trafienia i nietrafienia w cache
- itd.

Dostępne technologie – perf

Nowoczesne procesory mają liczniki sprzętowe, zliczające:

- cykle w których procesor nie spał
- wykonane instrukcje
- odwołania do pamięci
- trafienia i nietrafienia w cache
- itd.

Perf – API w Linuxie do czytania tych liczników

- odczyt liczby zdarzeń dotyczących wybranego procesu
- ...lub grupy procesów
- z podziałem na przestrzeń jądra i użytkownika

Dostępne technologie – seccomp-bpf

Seccomp – mechanizm filtrowania wywołań systemowych w Linuxie

- przygotowujemy program filtrujący zapisany w BPF
- przekazujemy go do jądra
- jądro sprawdza filtr przy każdym wywołaniu systemowym
  - bardzo wcześnie, zanim przejdzie do kodu odpowiedzialnego za konkretny numer wywołania – ograniczamy powierzchnię ataku
  - bardzo szybko – wszystko dzieje się w jądrze, nie trzeba przełączać kontekstu

Dostępne technologie – ptrace

Ptrace – standardowy interfejs POSIXowy do debugowania procesów. Pozwala m.in na:

- zatrzymanie procesu
  - przed wejściem w wywołanie systemowe
  - po wyjściu z wywołania systemowego
  - po wystąpieniu sygnału TRAP
  - po zwróceniu odpowiedniej wartości przez filtr seccomp
- odczyt rejestrów i przestrzeni adresowej zatrzymanego procesu
- zapis rejestrów i przestrzeni adresowej zatrzymanego procesu

Dostępne technologie – namespaces

Namespaces – mechanizm w Linuxie pozwalający kontrolować w jaki sposób różne grupy procesów widzą niektóre zasoby jądra, takie jak:

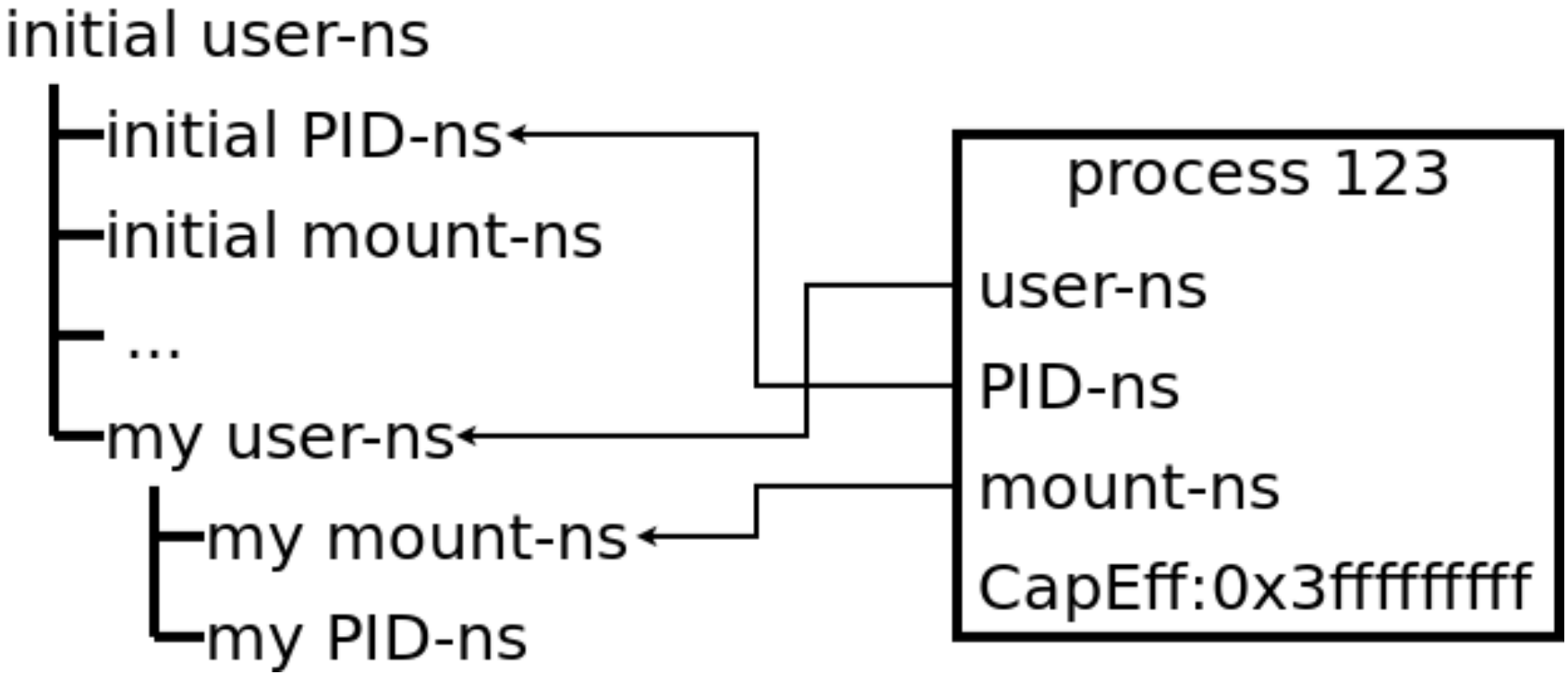
- wirtualny system plików – mount namespaces
- uruchomione procesy – PID namespaces
- interfejsy sieciowe – network namespaces
- obiekty IPC (kolejki komunikatów itd.) – IPC namespaces
- nazwa hosta – UTS namespaces
- identyfikatory użytkowników – user namespaces

Przynależność do namespace-u jest dziedziczona z procesu-rodzica na jego procesy potomne. Z założenia proces nie powinien być w stanie uzyskać dostępu do obiektów spoza swojego namespace-u.

User namespaces, capabilities, i czym jest root

- `uid == 0` nie decyduje czy proces jest uprzywilejowany
- capabilities – flagi konrolujące dostęp danego procesu do różnych grup uprzywilejowanych operacji
  (np. `CAP_CHOWN` decyduje czy możemy zmieniać właściciela pliku na dowolnego użytkownia)
- w każdym user namespace-sie ten sam proces może mieć inne capabilities
- proces tworzący user namespace dostaje wewnątrz niego wszystkie capabilities
- capabilities wewnątrz user namespace-u dotyczą tylko zasobów utworzonych wewnątrz

User namespaces, capabilities, i czym jest root



Mimo, że proces 123 ma `CAP_KILL` w namespace-sie "my user-ns", i jego PID namespacem jest "initial PID-ns" to nie może zabijać procesów innych użytkowników w initial PID-nsie.
Może to robić jedynie w "my PID-ns". Ma też `CAP_SYS_ADMIN` w "my user-ns", więc może montować systemy plików "my mount-ns", ale nie w "initial mount-ns".

Dostępne technologie – rlimit

rlimit (a.k.a. ulimit) – POSIXowy mechanizm określania maksymalnego rozmiaru zasobów zużywanych przez proces. W Linuxie pozwala ograniczyć m.in.:

- rozmiar przestrzeni adresowej
- rozmiar stosu
- czas działania programu
- liczbę otwartych plików

Dostępne technologie – cgroup

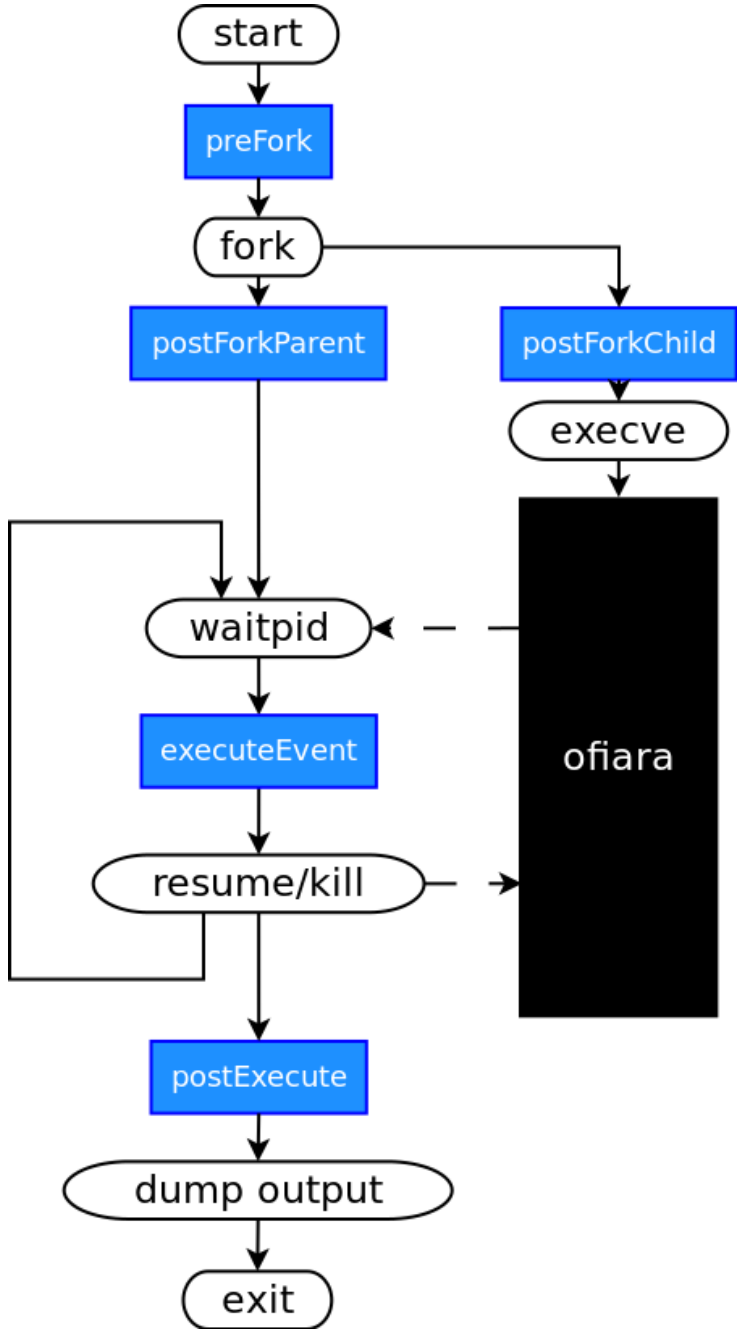
Architektura – cykl życia

Cykl dzieli się na

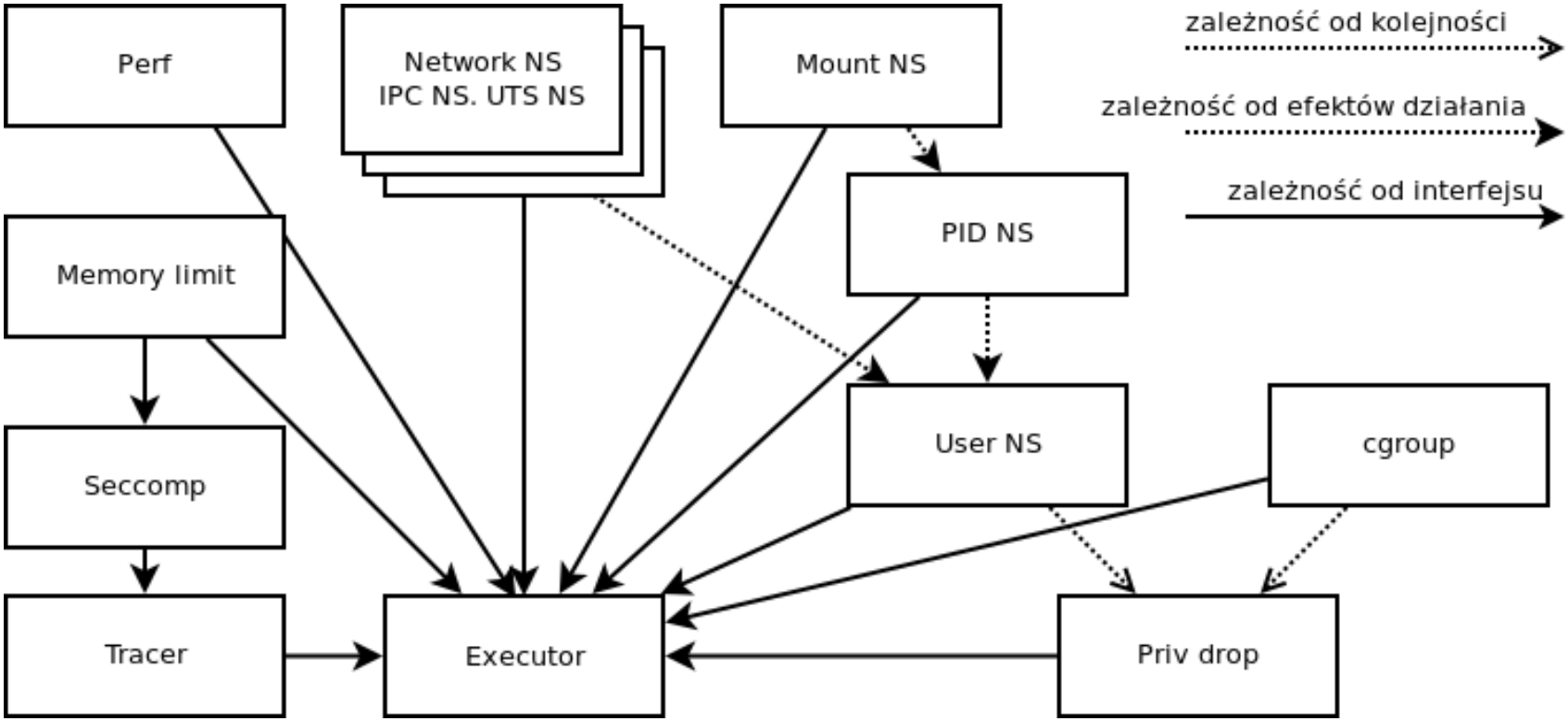
- etapy wykonujące się w pętli głównej (białe owale)
- etapy wywołujące callbacki modułów (niebieskie prostokąty)

Co się dzieje na poszczególnych etapach?

- preFork, postForkParent, postForkChild: przygotowanie środowiska
- waitpid, executeEvent, resume/kill: reakcja na zdarzenia z procesu-dziecka
- postExecute: zebranie wyników, przygotowanie danych wyjściowych



Architektura – moduły



Demo

```
$ ./sio2jail -b boxes/minimal:/:ro test/1-sec-prog
0
__RESULT__ 0 1000 0 896 0
ok
$ ./sio2jail -b boxes/minimal:/:ro test/1-sec-evil
__RESULT__ 137 1000 0 900 0
intercepted forbidden syscall openat(295)
$
```

Testy porównawcze

Wykonaliśmy testy porównujące wyniki sio2jaila z z oitimetoolem

- 500 000 losowo wybranych par (zgłoszenie, test) z II i III etapu 24. i 25. Olimpiady Informatycznej
- kompilacja gcc-4.9.2
- uruchomienie za pomocą oitimetool i sio2jail i zapisanie
  - statusu zgłoszenia (OK, błąd wykonania, przekroczenie limitu, itd.)
  - zmierzonego czasu (liczba instrukcji)
  - zmierzonej pamięci
  - czasu trwania sprawdzania

Testy porównawcze - liczba instrukcji

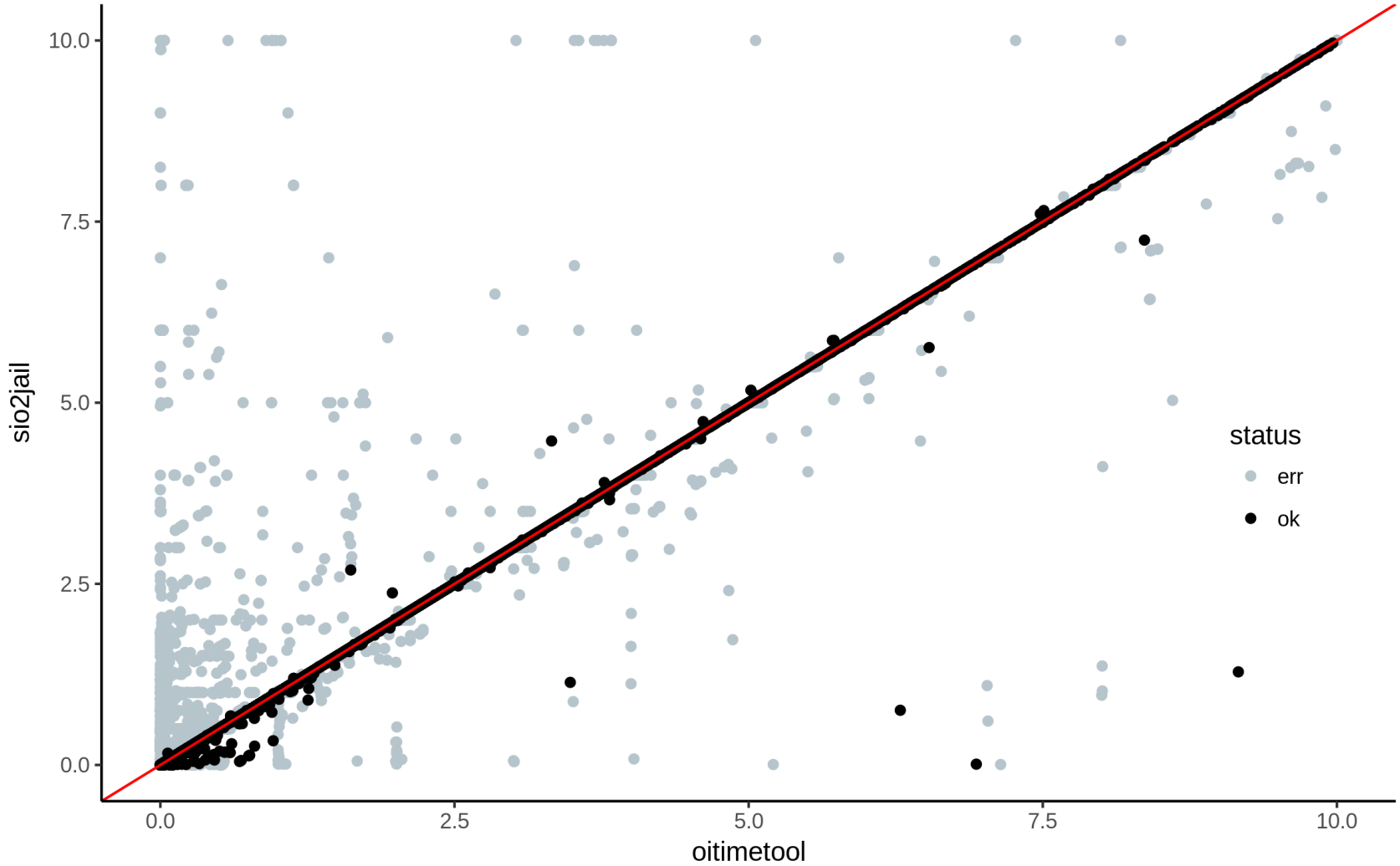


Figure 1: Czas zmierzony sio2jail w zależności od czasu zmierzonego oitimetool

Testy porównawcze - liczba instrukcji

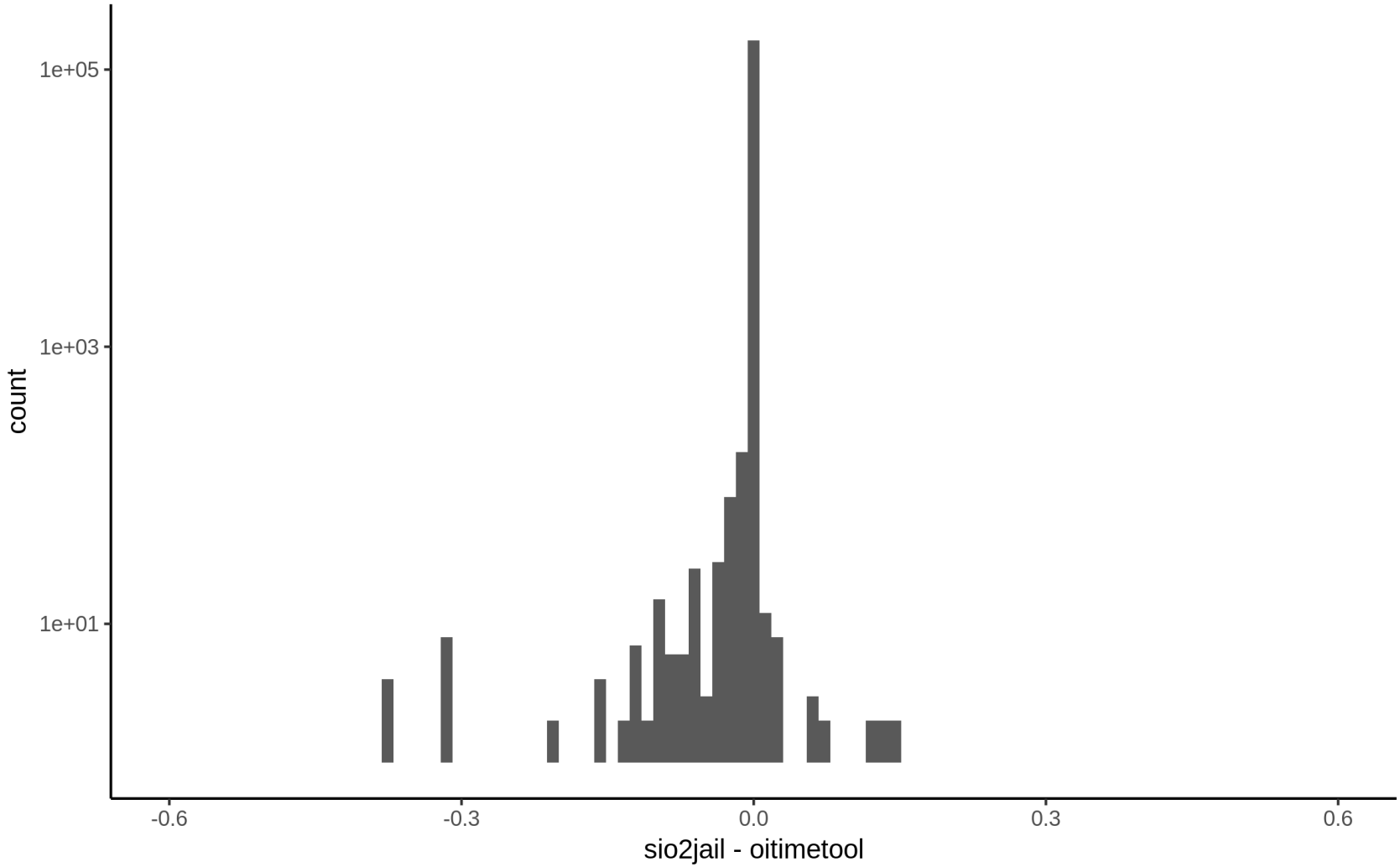


Figure 2: Histogram różnicy zmierzonych czasów (*sio2jail* – *oitimetool*)

Testy porównawcze - zużycie pamięci

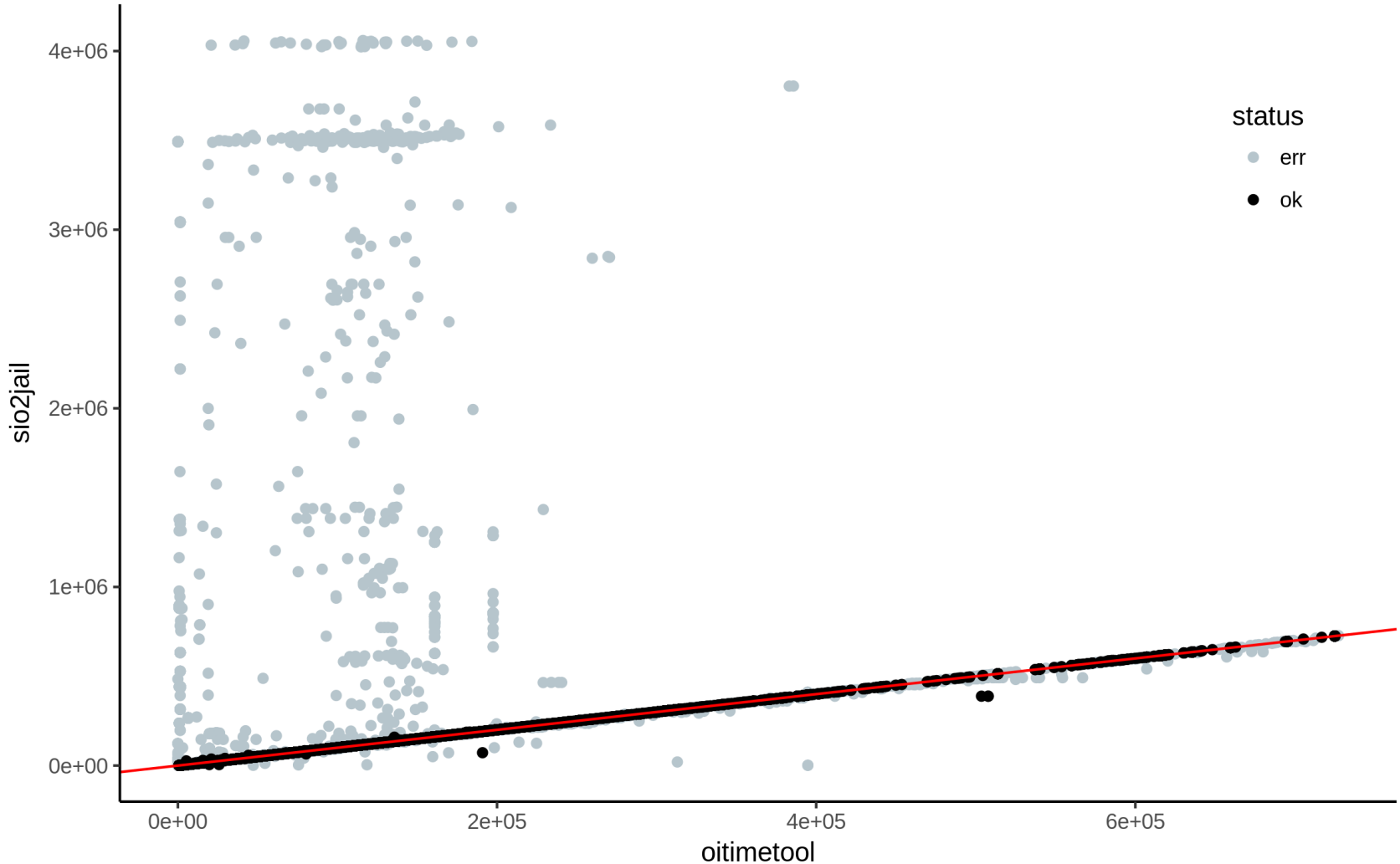


Figure 3: Zużycie pamięci zmierzone sio2jail w zależności od zmierzonego oitimetool

Testy porównawcze - zużycie pamięci

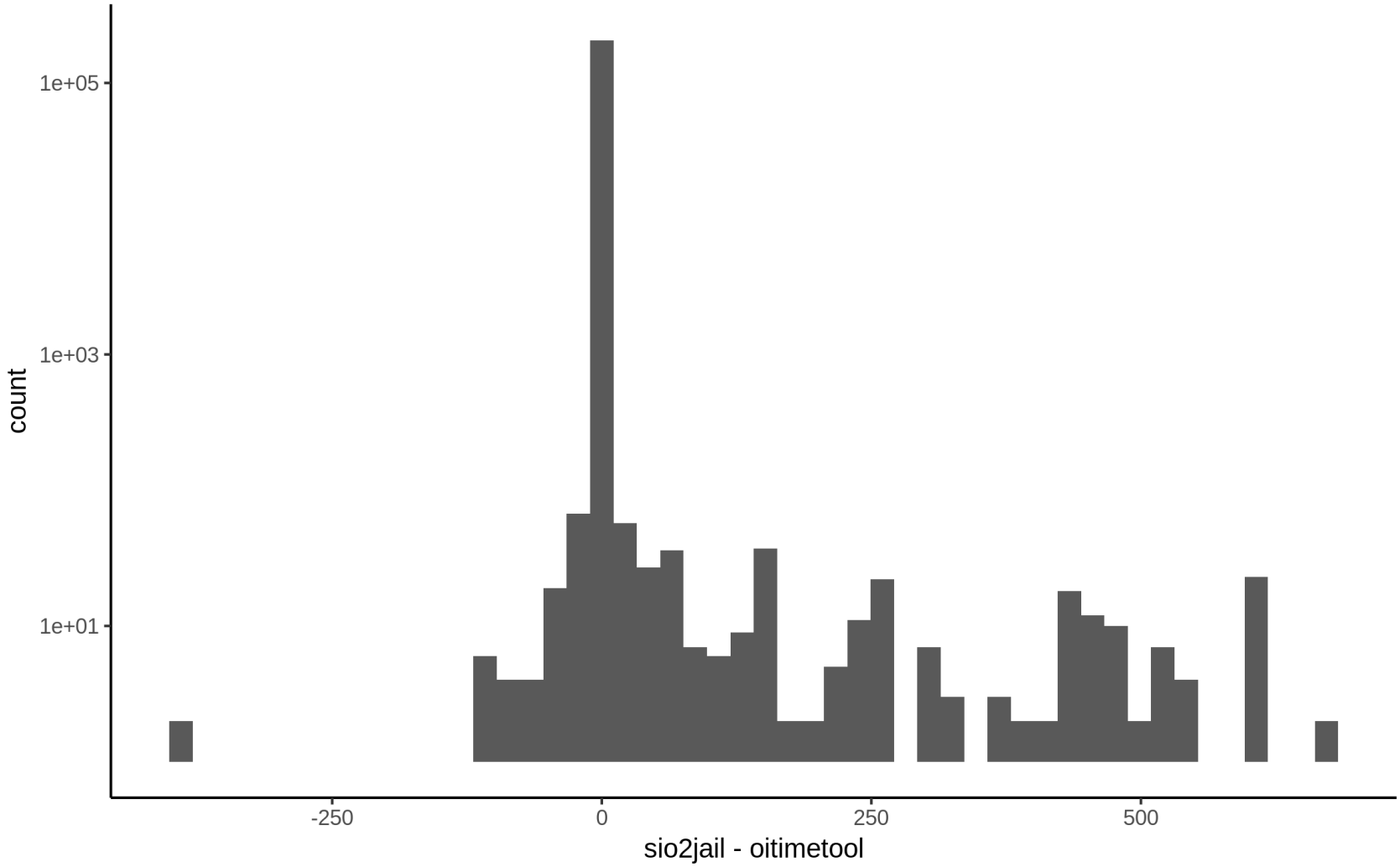


Figure 4: Histogram różnicy zmierzonych zużyć pamięci (*sio2jail* − *oitimetool*)