```
$ strace -f -p 4121 2>&1 | grep ...
```

Beyond strace



System call troubleshooting with Stratoshark,
from a packet analyst's point of view

Roland Knall | Open Source Summit Prague, 2026

```
talk.section == "problem"
```

You know this moment

```
$ strace -f -p 4121 2>&1 | grep -v EAGAIN
[pid  4127] openat(AT_FDCWD, "/etc/app/conf.d/db.yaml", O_RDONLY) = -1 ENOENT
[pid  4121] epoll_wait(7,  <unfinished ...>
[pid  4130] connect(12, {sa_family=AF_INET, sin_port=htons(5432), ...}, 16 <unfinished ...>
[pid  4127] write(2, "config: falling back to defaults"..., 33) = 33
[pid  4130] <... connect resumed>)  = -1 EINPROGRESS (Operation now in progress)
[pid  4121] <... epoll_wait resumed>[{events=EPOLLIN, data={u32=9, u64=9}}], 128, -1) = 1
... 48,000 more lines
```

strace recorded all of it correctly. Reading it is left to you.

```
talk.section == "problem" && grep.helps == false
```

Where grep stops helping

The problem spans processes

strace -f follows children of one tree. A proxy, its backend and a database client are three unrelated trees.

There is too much output

You pick -e trace= before you know what matters. Guess wrong and you reproduce the fault again.

System and network are separate

strace in one terminal, tcpdump in another, and timestamps you line up by eye.

ptrace slows the target

Worst-case microbenchmark: a geteuid() loop ran 102x slower under strace, 2.4x under bpftrace.

Overhead figures: Brendan Gregg, syscount.bt(8) man page, OVERHEAD section

strace and bpftrace are very good at capturing system calls.

They give you much less help analysing them.

```
talk.section == "background"
```

Wireshark in one minute

An open source protocol analyzer. Gerald Combs started it in 1998 as Ethereal; it has carried the name Wireshark since 2006. More than 2,000 people have contributed code.

It does not interpret a capture for you. It gives you a fast way to ask questions of one.

What analysts rely on

- Display filters on any dissected field
- Coloring rules that mark trouble
- Follow Stream for one conversation
- Time deltas and time references
- Statistics and I/O graphs
- Profiles for different jobs
- Capture files anyone can reopen

```
talk.section == "background" && org.type == "nonprofit"
```

The Wireshark Foundation

A US 501(c)(3) nonprofit, announced in March 2023. It hosts the project's infrastructure, runs the SharkFest conferences and funds analysis education. Members, sponsors and donations pay for it.

1998

Ethereal, first release

2006

Renamed Wireshark

2023

Foundation takes over the project

Jan 2025

Stratoshark announced

May 2025

Sysdig donates Stratoshark to the foundation

```
talk.section == "motivation"
```

Two ecosystems with the same shape

Role	Packets (pcap)	System calls (scap)
Capture library	libpcap	libscap, with libsinsp for context
File format	.pcap, .pcapng	.scap
Command line capture	tcpdump	sysdig
Detection	Snort, Suricata, Zeek	Falco
Interactive analysis	Wireshark	Stratoshark

The right-hand column had no entry in the last row until 2025. Packets are also harder to get in container platforms, where the syscall boundary is still available on every node.

```
talk.section == "background" && project == "falco"
```

Falco: runtime detection from syscalls




```
evt.type = execve and container.id != host and proc.name = bash
```

A rule condition, written in the same field names you will filter on later

Created at Sysdig in 2016, given to the CNCF in 2018, a graduated CNCF project since 2024.

```
talk.section == "background" && org == "falcosecurity"
```

The Falco project family

falco

The detection daemon and its default rule set

libs

libscap, libsinsp and the kernel drivers

plugins

Other event sources: CloudTrail, Kubernetes audit, GCP audit, Okta

falcosidekick

Forwards alerts to chat, SIEM and message queues

falcoctl

Installs and updates rules and plugins

falco-talon

Runs response actions in Kubernetes when a rule fires

Stratoshark uses the two parts shown in filter-bar green. It does not need a running Falco daemon.

```
talk.section == "tool"
```

Stratoshark

Wireshark's dissection engine and user interface, fed with system calls and logs.

It is built from the Wireshark source tree as a sibling application. The event list, detail tree, filter bar and dialogs are the ones you already know.

Status in October 2026

- Version 0.10.3
- Installers for Windows and macOS
- Linux: build from source or use a distribution package
- Reads .scap, CloudTrail, Kubernetes and GCP audit logs, Procmon files
- Ships strato, a tshark counterpart

```
talk.section == "tool" && capture.source
```

Four ways to get events in

Source	What it captures	Where it runs
`dumpcalls`	Local system calls	Stratoshark on Linux
`sshdig`	System calls on a remote host	Any desktop; sysdig on the target
`falcodump`	Logs through Falco plugins	Any desktop
`sysdig -w`	A .scap file to open later	The Linux target, no GUI needed

```
$ sudo sysdig -s 4096 -w incident.scap "proc.name=haproxy or proc.name=nginx"
```

Without -s, sysdig keeps only the first 80 bytes of each read and write buffer.

```
talk.section == "tool" && habit.transfers == true
```

Which Wireshark habits carry over

In Wireshark you would	In Stratoshark you
Filter on ip.addr or tcp.port	Filter on proc.name, evt.type, fd.name
Follow a TCP stream	Follow a file descriptor stream
Colour resets and retransmissions	Colour failed calls and process starts
Add a field as a column	Do the same with pid, container, latency
Set a time reference, read deltas	Do the same between two calls
Open an I/O graph	Graph calls per second by any filter
Send a pcapng to a colleague	Send a .scap

What does not carry over: knowing which of the roughly 350 Linux system calls matter. That knowledge is yours, and the tool assumes it.

```
talk.section == "tool" && ui.type == "graphical"
```

Why a graphical tool for syscalls

You filter after capture

Record wide once. Change the question ten times without reproducing the fault.

Context is already resolved

Each event carries the process, its parent, the container and what the descriptor points to.

Overview and detail share a screen

A graph shows the stall. A click on it lands on the call that caused it.

The evidence travels

A capture file attached to a ticket lets the next person check your reasoning with their own filters.

Scripts and pipelines still have a place: strato and sysdig read the same files from a shell.

```
demo.number == 1   # about 6 minutes
```

A service that will not stay up

A supervisor starts four workers. One of them exits within a second and is restarted in a loop. The log says only "worker exited".

```
$ sudo sysdig -s 2048 -w demo1.scap
```

What to watch for

1. The same three panes, with events in place of packets
2. Right-click a field, apply it as a filter
3. Failed calls only, across every process
4. Follow the worker's stderr descriptor
5. The same question asked of strace -f output

```
talk.section == "demos" && (syscall || tcp)
```

The same connection, seen from two sides

A syscall capture tells you

- Which process and container owns the socket
- When the application called connect, read and write
- How long each call blocked
- What else that process did in between

Address, port and timestamp join the two. An application using TLS hands ciphertext to write(), so neither capture shows plaintext without extra work.

A packet capture tells you

- What left the host and what arrived
- Retransmissions, window sizes, round-trip time
- Resets and who sent them
- Behaviour of devices in the path

```
demo.number == 2   # about 7 minutes
```

The proxy answers 502

HAProxy sits in front of several backend containers. Some requests come back as 502 Bad Gateway. The proxy log names no cause.

Capture file: 502Error.scap, a public sample linked from the Wireshark wiki.

What to watch for

1. Find the 502 in a write buffer
2. Follow that descriptor to the request
3. Find the proxy's connection to the backend
4. Switch to the backend process and container
5. Read the time gap between the two

```
talk.section == "verdict" && winner == "stratoshark"
```

Where Stratoshark is the better choice

You do not know the culprit	"Something keeps rewriting this file." Capture the host, filter on the file name, read the process column.
Several services are involved	One timeline covers proxy, application and database client, with container names attached.
The fault is rare	Record until it happens, then analyse offline as often as you need.
Someone else must verify	The vendor, the developer or the auditor gets the capture file and can test their own theory.
The target is busy	An eBPF driver does not stop the process on every call the way ptrace does.

```
talk.section == "verdict" && winner != "stratoshark"
```

Where the older tools are still right

strace

- One process, on the box, right now
- Present on nearly every system, no root needed for your own processes
- Deeper argument decoding: structs, flags, ioctl requests
- Fault injection, stack traces with -k, summaries with -c

bpftrace

- Anything that is not a syscall: kernel functions, tracepoints, user-space probes
- Aggregation in the kernel: counts and latency histograms without storing events
- Questions nobody built a tool for
- Low cost on very busy hosts

```
talk.section == "verdict" && _ws.expert.severity == "warn"
```

Limitations today

It is version 0.10

A full user guide does not exist yet. Expect rough edges and regressions between releases.

Linux desktop support is thin

No upstream Linux packages. You build it, or take an older distribution package.

Syscall capture is Linux only

On Windows and macOS you analyse files, capture logs, or capture remotely over SSH.

Capturing needs root

And sysdig or a matching driver on the target, which some hardened hosts will not allow.

Captures grow fast and can drop

A whole host produces far more events than one interface produces packets. Full buffers lose events.

A capture holds secrets

Buffers contain passwords, tokens and file contents from every process on the host.

```
talk.section == "verdict" && tool.choice
```

Which tool for which question

The question	Start with
Why does this one command fail?	strace
Which process touches this file or port?	sysdig, then Stratoshark
What happened across these three services?	Stratoshark
How is latency distributed over an hour?	bpftrace
What is the kernel doing below the syscall?	bpftrace, perf
What happened on the wire?	Wireshark
How do I hand this over for review?	A .scap and Stratoshark

Most real incidents use two of these rows.

```
talk.section == "next"
```

Try it this week

1. Install Stratoshark from stratoshark.org on your desktop
2. Install sysdig on a Linux test machine
3. Record ten seconds while you run something you know well
4. Open the file and find one thing you did not expect

```
$ sudo timeout 10 sysdig -s 1024 -w first.scap
$ scp server:first.scap . && stratoshark first.scap
```

No Linux machine at hand: the Wireshark wiki page for Stratoshark links sample captures, among them the demo 2 file.

WIRESHARK FOUNDATION

Keep strace. Add a second tool for the cases it was never built for.

stratoshark.org
wiki.wireshark.org/Stratoshark
falco.org
wiresharkfoundation.org

Roland Knall
Bsky: @knall.cloud