

eBPF Security: The Uneven Landscape

Louie Wolf Gürkan Gür

Zurich University of Applied Sciences (ZHAW)
Institute of Computer Science INIT

Linux Plumbers Conference (LPC), 2026



LINUX
PLUMBERS
CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026

Who we are

Presenters

- ▶ Dr. Gürkan Gür
- ▶ Louie Wolf

Other Team Members

- ▶ Tobias Leu
- ▶ Gökcan Cantali

Acknowledgements

- ▶ Special Thanks to Daniel Borkmann

How did we get here

Timeline

- ▶ It all started with a simple seminar paper about the security of eBPF...

How did we get here

Timeline

- ▶ It all started with a simple seminar paper about the security of eBPF...
- ▶ We screened a large set of papers and summarized the results.

How did we get here

Timeline

- ▶ It all started with a simple seminar paper about the security of eBPF...
- ▶ We screened a large set of papers and summarized the results.
- ▶ There was actually a research gap, so I, as the MSE advisor, proposed to send it to a security workshop.

How did we get here

Timeline

- ▶ It all started with a simple seminar paper about the security of eBPF...
- ▶ We screened a large set of papers and summarized the results.
- ▶ There was actually a research gap, so I, as the MSE advisor, proposed to send it to a security workshop.
- ▶ We then revised it, screened 1000+ papers again and summarized the results.

How did we get here

Timeline

- ▶ It all started with a simple seminar paper about the security of eBPF...
- ▶ We screened a large set of papers and summarized the results.
- ▶ There was actually a research gap, so I, as the MSE advisor, proposed to send it to a security workshop.
- ▶ We then revised it, screened 1000+ papers again and summarized the results.
- ▶ The workshop got canceled :-)

How did we get here

Timeline

- ▶ It all started with a simple seminar paper about the security of eBPF...
- ▶ We screened a large set of papers and summarized the results.
- ▶ There was actually a research gap, so I, as the MSE advisor, proposed to send it to a security workshop.
- ▶ We then revised it, screened 1000+ papers again and summarized the results.
- ▶ The workshop got canceled :-)
- ▶ Louie and Tobias thought this was over.

How did we get here

Timeline

- ▶ It all started with a simple seminar paper about the security of eBPF...
- ▶ We screened a large set of papers and summarized the results.
- ▶ There was actually a research gap, so I, as the MSE advisor, proposed to send it to a security workshop.
- ▶ We then revised it, screened 1000+ papers again and summarized the results.
- ▶ The workshop got canceled :-)
- ▶ Louie and Tobias thought this was over.
- ▶ I was persistent ...

How did we get here

Timeline

- ▶ It all started with a simple seminar paper about the security of eBPF...
- ▶ We screened a large set of papers and summarized the results.
- ▶ There was actually a research gap, so I, as the MSE advisor, proposed to send it to a security workshop.
- ▶ We then revised it, screened 1000+ papers again and summarized the results.
- ▶ The workshop got canceled :-)
- ▶ Louie and Tobias thought this was over.
- ▶ I was persistent ...
- ▶ Send it to a conference, and also contacted Daniel Borkmann to get real hands-on insights.

How did we get here

Timeline

- ▶ It all started with a simple seminar paper about the security of eBPF...
- ▶ We screened a large set of papers and summarized the results.
- ▶ There was actually a research gap, so I, as the MSE advisor, proposed to send it to a security workshop.
- ▶ We then revised it, screened 1000+ papers again and summarized the results.
- ▶ The workshop got canceled :-)
- ▶ Louie and Tobias thought this was over.
- ▶ I was persistent ...
- ▶ Send it to a conference, and also contacted Daniel Borkmann to get real hands-on insights.
- ▶ We asked Daniel how it is going in the "trenches," and he proposed a lot of insights.

Outline: eBPF in Two Acts

- ▶ **ACT I** — Capturing the State
- ▶ **ACT II** — Life in the Trenches

ACT I: Capturing the State

Motivation: Why eBPF Security?

eBPF: Powerful but Risky

- ▶ eBPF enables programs to run in the **Linux kernel context**
- ▶ Any vulnerability can cause **system crash**, **memory corruption**, or **information leakage**
- ▶ Use cases: kernel observability, networking, security enforcement, ...
- ▶ Widely adopted in Cilium, Tetragon, Falco, and more

The Gap

Prior work focuses on **individual components** (e.g., verifier correctness, JIT memory safety) rather than the **full eBPF lifecycle**.

eBPF Program Lifecycle

Two Phases

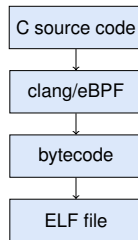
Development:

- ▶ C source code → Clang/eBPF target
- ▶ Architecture-independent bytecode → ELF container

Runtime:

- ▶ Bytecode → **Verifier** (static safety analysis)
- ▶ Verified bytecode → **JIT Compiler** (native machine code)
- ▶ Execution via **Runtime** (hooks, maps, helpers/kfuncs)

Development



Runtime

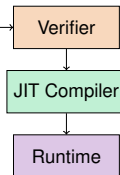


Figure: eBPF program workflow.

Structured Literature Review

1. Searched ACM DL, IEEE Xplore, Springer-Link, arXiv
2. Query: ("eBPF") AND ("security" OR "vulnerability")
3. 1,690 records → 1,065 after dedup
4. Three-stage screening: 144 (abstract) → 58 (section) → 29 (full-text)
5. Backward snowballing → **30 final publications**

CVE Analysis

- ▶ Searched CVE.org for keyword `bpf`
- ▶ **224 CVEs** identified and mapped to lifecycle components
- ▶ The Linux kernel became its own CVE Numbering Authority (CNA) → Cross-referenced with security-relevant kernel commits (Jan–Aug 2026)

CVE Distribution Across eBPF Components

224 CVEs by Component

Component	CVEs	%
Verifier	104	45.0%
Maps	50	21.6%
JIT Comp.	39	16.9%
Helpers	35	15.2%
Interpreter	3	1.3%

Notable Asymmetry

The verifier accounts for **nearly half** of all CVEs, exceeding the share of any other two components combined.

Maps: The Overlooked Component

Maps are the **second-largest** source of disclosed vulnerabilities but receive comparatively little dedicated academic treatment.

ACT II: WHAT IS GOING ON IN THE TRENCHES?

Recent Developments

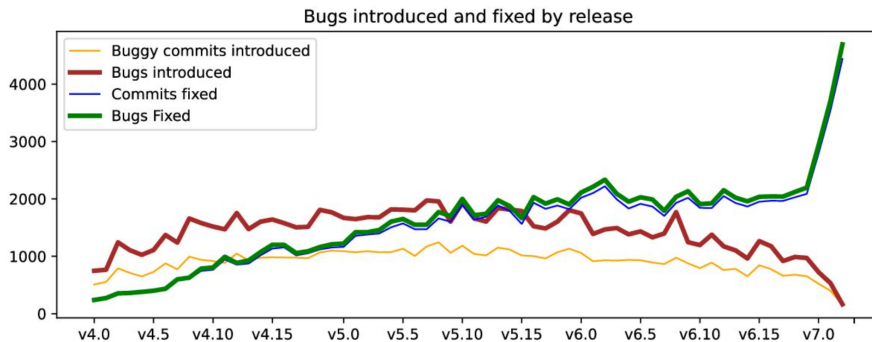


Figure: Plot from Jonathan Corbet [1] @ Kernel Recipes 2026

Bug Fix Analysis: Fixes Trailer

commit afe897e89918434be8600250dbdda977aa99f53d

Author: Yonghong Song <yonghong.song@linux.dev>

Date: Sat Sep 12 12:52:01 2026 -0700

bpf: Read a kfunc's __sz argument only when it is in a register

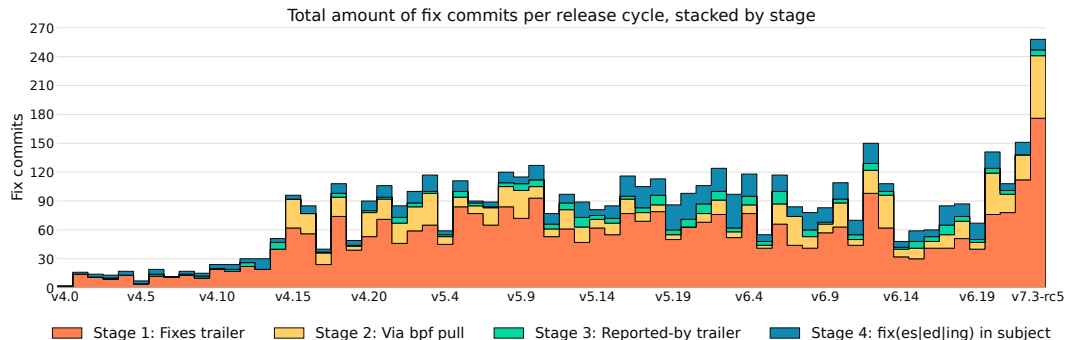
In bpf_kfunc_stack_access_bytes(), the size of a ptr + __sz pair is read from const_reg_vals[] at index 'BPF_REG_1 + arg + 1' [...]

Fixes: e0b7b91c72db ("bpf: Support stack arguments for kfunc calls")

Signed-off-by: Yonghong Song <yonghong.song@linux.dev>

Link: <https://lore.kernel.org/r/202609121952...>

Recent Developments in eBPF



Observation

The number of fix commits has also reached an all-time high in BPF.

BPF Mailing List Activity (2021–2026)

Year	E-mails	E-mails per day (avg.)
2021	19580	53.6
2022	24833	68.0
2023	28278	77.5
2024	28721	78.7
2025	29850	81.8
2026*	35583	97.5

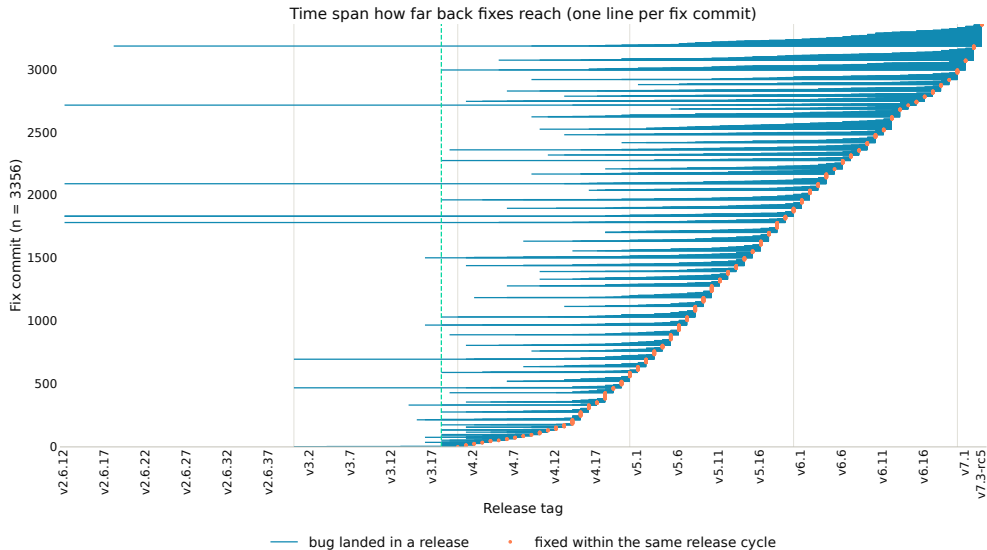
Table: Yearly e-mail volume on the bpf mailing list.

Observation

Average daily e-mail volume on the BPF lore rose from 54 to 98 between 2021 and 2026.

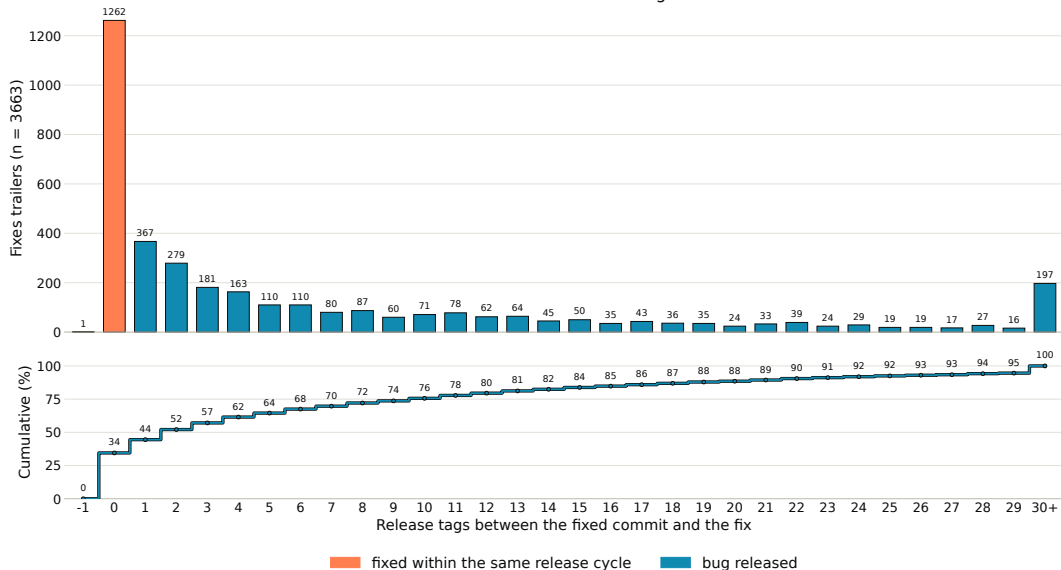
*cut-off date at end of September

Timespan Between Bug and Fix



Timespan Between Bug and Fix

Number of releases between a bug and its fix



What do these authors have in common?

Chengfeng Ye, Eric Dumazet, Jiayuan Chen, Cong Wang, Zijian Zhang, Michal Luczaj, Geliang Tang, Jason Xing, Jakub Sitnicki, Xu Kuohai, Xin Liu, Liu Jian, Pengcheng Yang, Wang Yufen, John Fastabend, Lorenz Bauer, Andrey Ignatov, Sabrina Dubroca, Lingpeng Chen

Fun Fact

On a dark autumn night, in the quiet hours before 03am...

```
commit 604326b4...
```

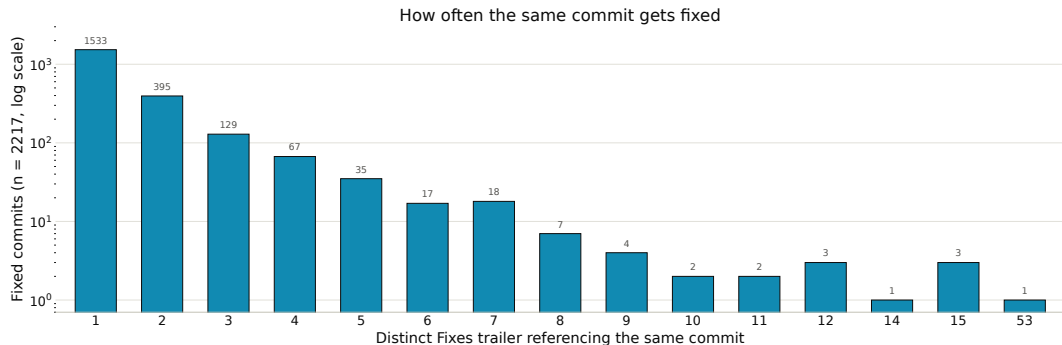
```
Author: <censored>
```

```
Date: Sat Oct 13 02:45:58 2018
```

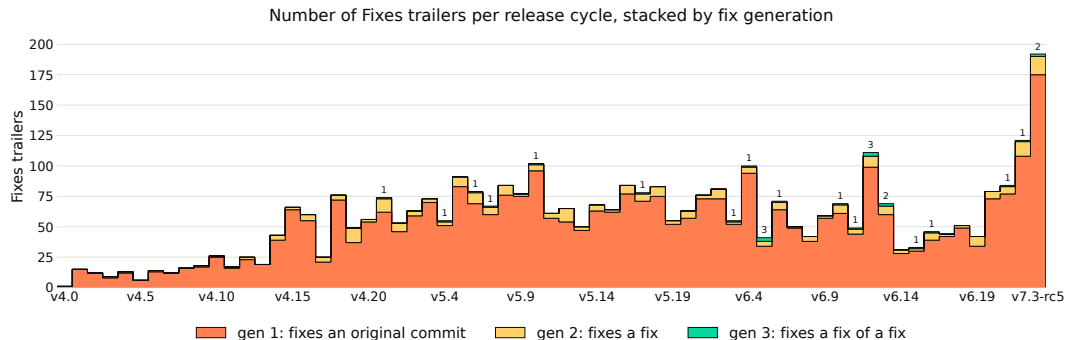
```
bpf, sockmap: convert to generic sk_msg interface
```

Add a generic `sk_msg` layer, and convert current sockmap and later kTLS over to make use of it. While `sk_buff` handles network packet representation from netdevice up to socket, `sk_msg` handles data representation [...]

Commits Getting Fixed Multiple Times



Generations of Fixes



To Summarize

1. **Sources** such as research papers and CVEs are helpful but not sufficient to provide a comprehensive picture of eBPF's security landscape. . .

To Summarize

1. **Sources** such as research papers and CVEs are helpful but not sufficient to provide a comprehensive picture of eBPF's security landscape. . .
2. **Bug fixes** have reached a record level, which creates a lot of work someone has to do. . .

To Summarize

1. **Sources** such as research papers and CVEs are helpful but not sufficient to provide a comprehensive picture of eBPF's security landscape. . .
2. **Bug fixes** have reached a record level, which creates a lot of work someone has to do. . .
3. Sent **E-mails** have also reached a record level, almost doubling the amount of e-mails per day someone has to read. . .

To Summarize

1. **Sources** such as research papers and CVEs are helpful but not sufficient to provide a comprehensive picture of eBPF's security landscape. . .
2. **Bug fixes** have reached a record level, which creates a lot of work someone has to do. . .
3. Sent **E-mails** have also reached a record level, almost doubling the amount of e-mails per day someone has to read. . .
4. **Bugs** sometimes stay in the code for a long time and are not always easy to fix. . .

How to fix this?

Could LLMs be used to analyze the live state of eBPF security AND help with fixing them at scale?

How to fix this?

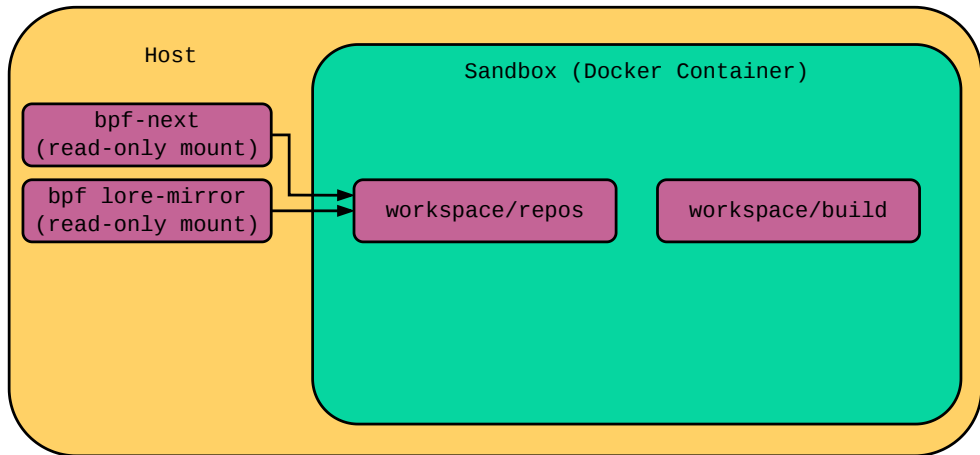
Could LLMs be used to analyze the live state of eBPF security AND help with fixing them at scale?

(just one more LLM bro, and it will fix everything!!!1!!!1!)

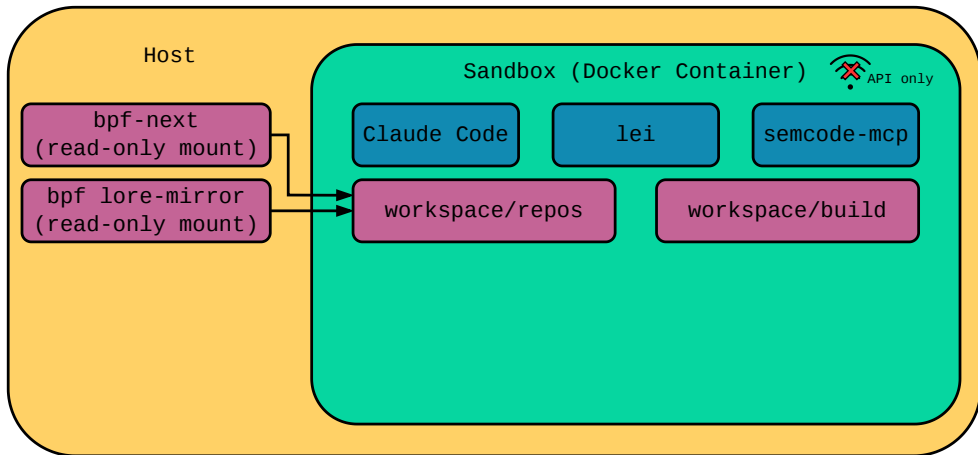
Agent-based Pipeline

1. Let the LLM **wait** for a bug report to come in and act as a "first aid responder".
2. Let it **reproduce** the bug to get an initial idea of whether the bug is real or could be a false positive.
3. Let it generate a **detailed description**, including the error chain and context, explaining why the bug occurred, as well as additional context.
4. Let it generate an initial **suggestion** on how to fix the bug.
5. **And most importantly** make sure to have low false positive rate!

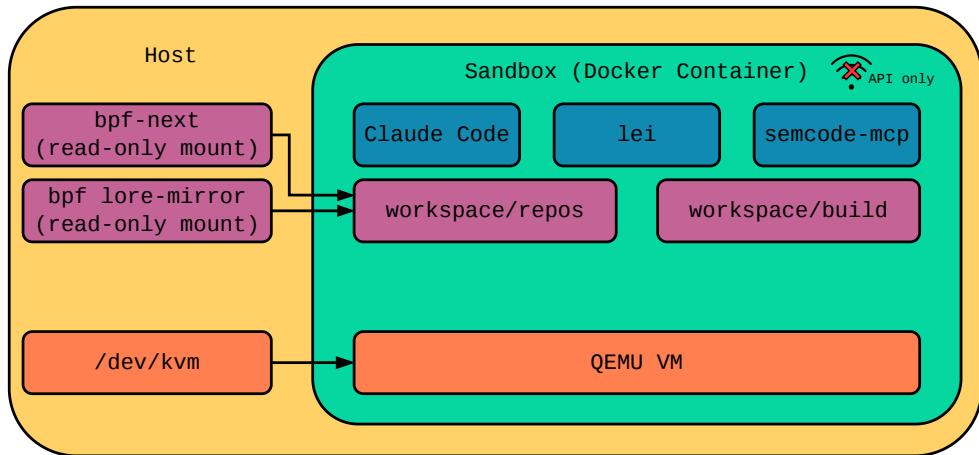
Setup for the MailFix Workflow



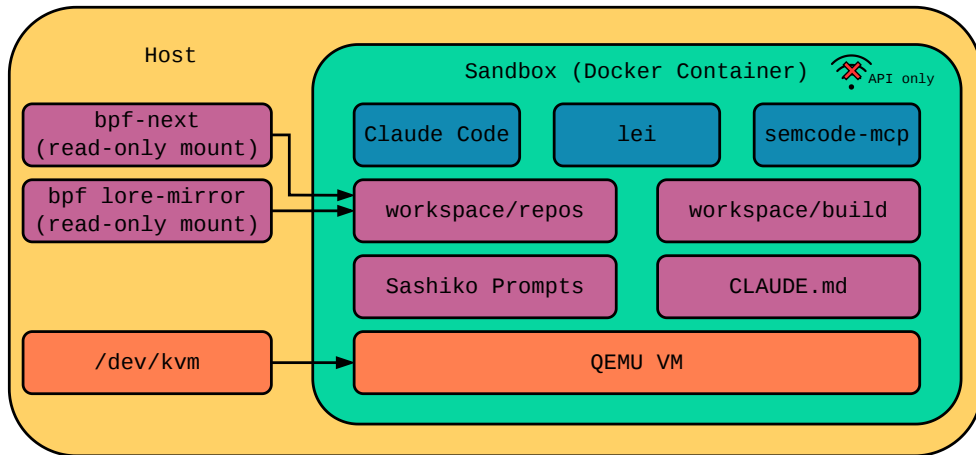
Setup for the MailFix Workflow



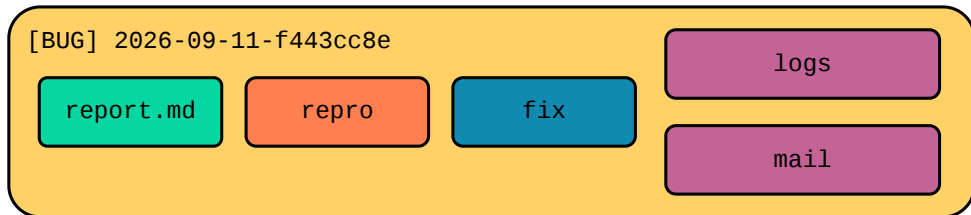
Setup for the MailFix Workflow



Setup for the MailFix Workflow



Bug Report Structure in More Detail



1. **The Bug**: what goes wrong, under which conditions, why it matters (crash, leak, hang, etc.)
2. **Reproduction**: What was built and run, what happened, log references
3. **How it happens**: call chain from the entry point to the faulting spot, with function names, files, and the state that goes wrong
4. **Reported vs. observed**: difference between the own reproduction and what was reported
5. **Proposed fix**: a patch diff: what it changes and evidence that it works
6. **Not verified**: What could not be run, confirmed, or compared

What could this be useful for?

1. Get **live data** by classifying each bug report, security relevant or not, which components are affected?
2. **Feed back** the results of the analysis to review tools like Sashiko by improving the prompts based on common mistakes we see in the data.
3. Add **context** for the people trying to fix the bug with the help of LLMs.

Wrap-up

- ▶ **Our Pipeline is work in progress:** collecting the security live data, verifying and improving the quality of results, etc.
- ▶ **Explore other LLMs:** we used Claude Code to test it, but there are other harnesses, open weight models and so on that could be tried as well.
- ▶ **Feedback is more than welcome:** if you have any feedback, ideas, please feel free to write to us -> wolo@zhaw.ch || gueu@zhaw.ch

References



Corbet, Jonathan The Kernel Report *Kernel Recipes*, 2026.
<https://kernel-recipes.org/en/2026/schedule/the-kernel-report/>



Anthropic Claude Code, 2026. <https://claude.com/product/claude-code>



Wong, Eric lei, 2026. <https://public-inbox.org/lei.html>



Meta Experimental semcode, 2026. <https://github.com/facebookexperimental/semcode>



Gushchin, Roman sashiko, 2026. <https://github.com/sashiko-dev/sashiko>



KH, Greg Security in the LLM age *Kernel Recipes*, 2026.
<https://kernel-recipes.org/en/2026/security-in-the-llm-age/>