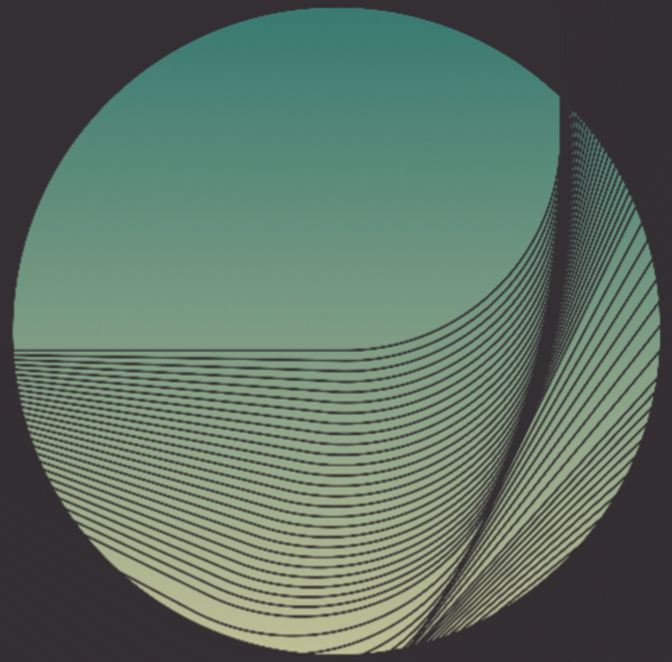
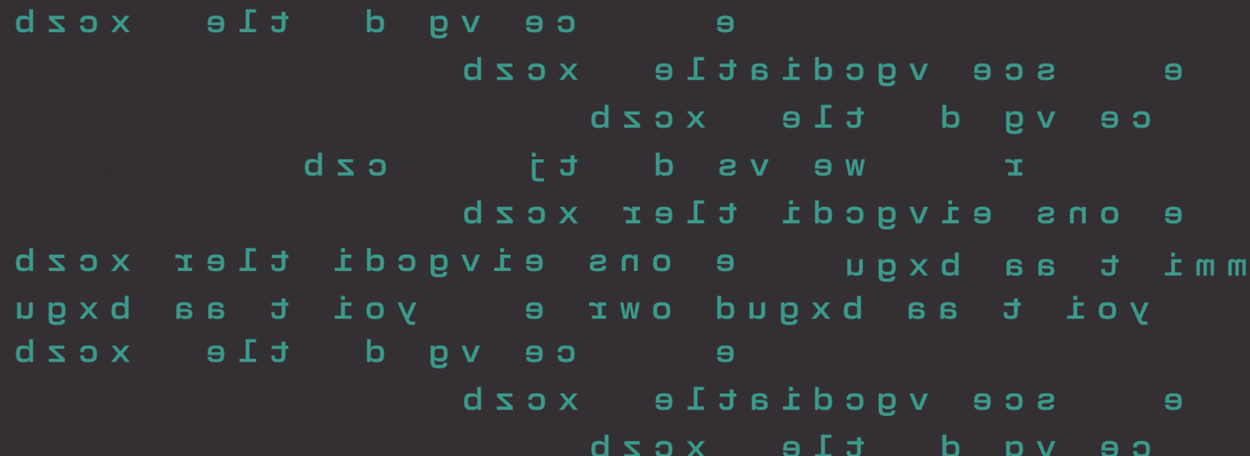


Orion – fuzzing workflow automation

Max Bazalii, Marius Fleischer



Who we are

Max Bazalii

Now

- NVIDIA DriveOS Offensive Security Team Lead
- Security research, AI automation and formal verification

Previously

- Apple OS security research – iOS/WatchOS jailbreaks
- Pegasus/Trident iOS spyware research

Marius Fleischer

Now

- NVIDIA DriveOS Offensive Security Team
- Applying AI to security challenges

Previously

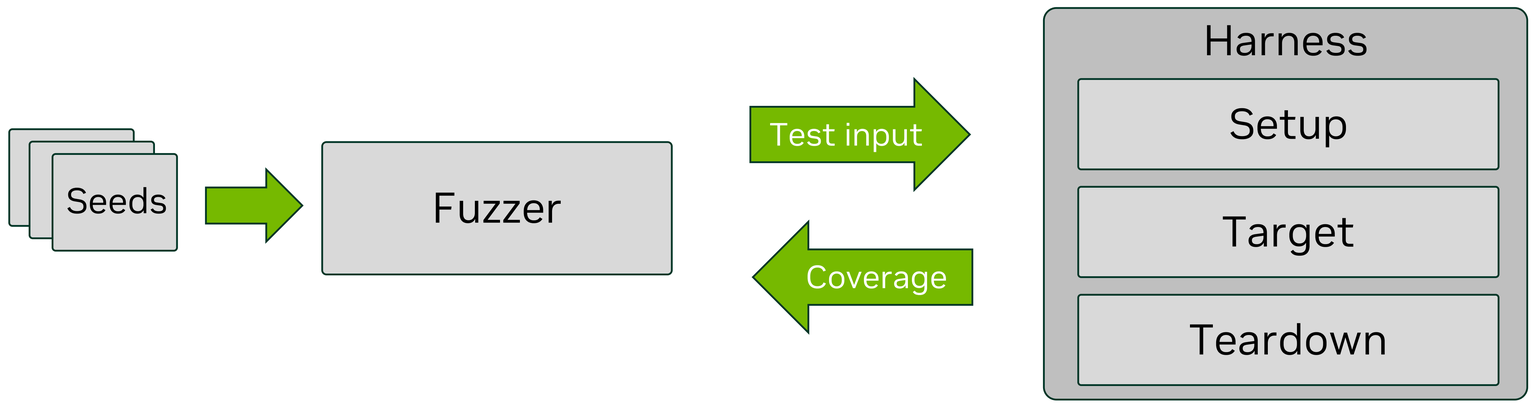
- UC Santa Barbara Security Lab
- Automated OS kernel vulnerability detection

How smart fuzzing works

Track → **Mutate** → **Execute** → **Learn**

Save inputs that unlock new paths

Repeat until crash. Add **sanitizers** to catch the good stuff

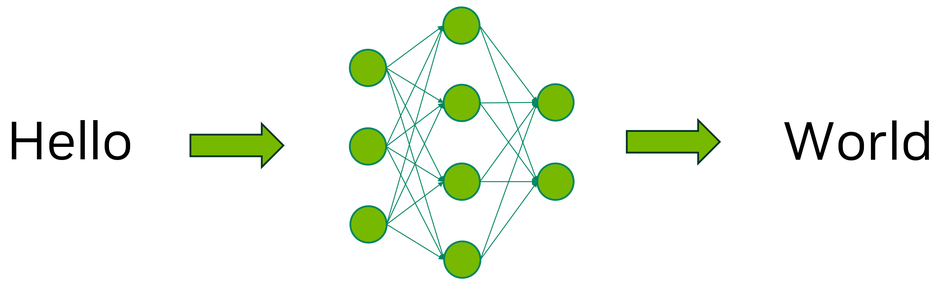


LLMs in one slide

Pretrained on massive corpora (foundation models)

Predict next token in a sequence

Can reason, analyze, generate code, and more

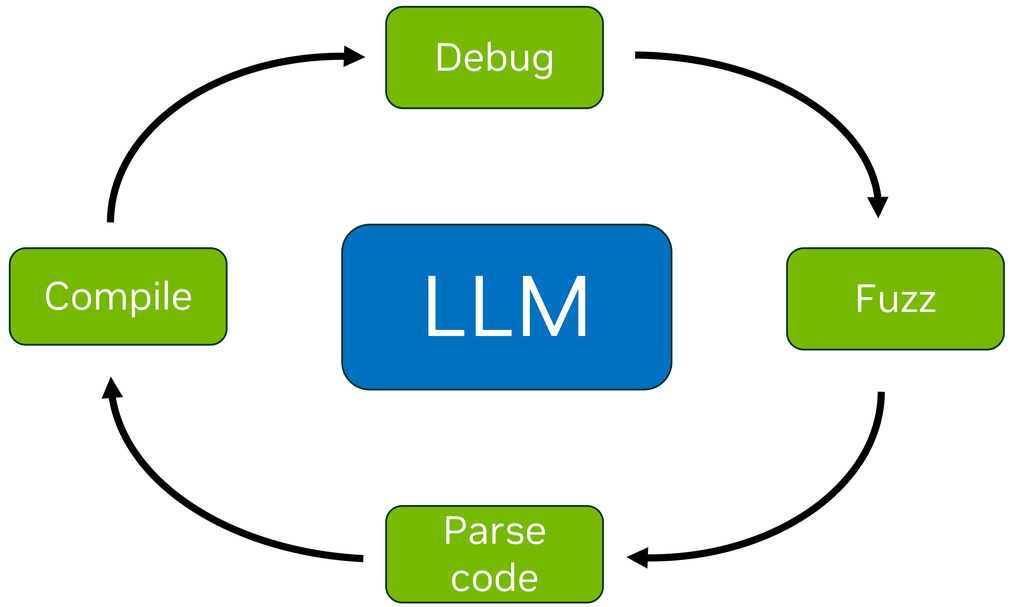


LLMs + Tools + ReAct Loop = Agents

Act on code via tools (e.g. GDB, fuzzers)

Understand structure across files/functions

Automate debugging, mutation, compilation

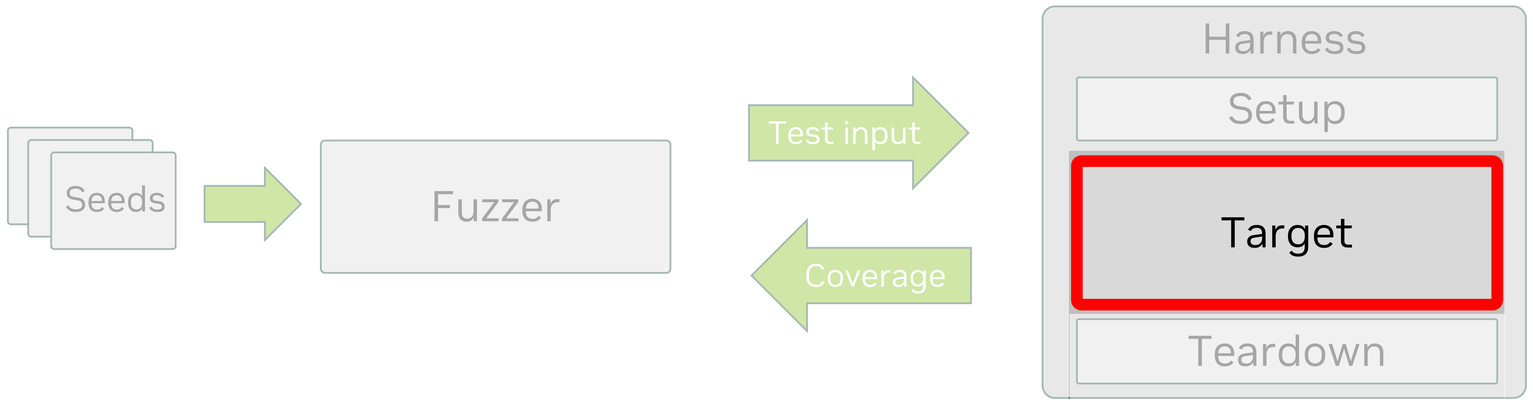


Challenge – fuzz target identification

Find key functions with attack surface

Hard in massive codebases with poor docs

Bad target = zero bugs

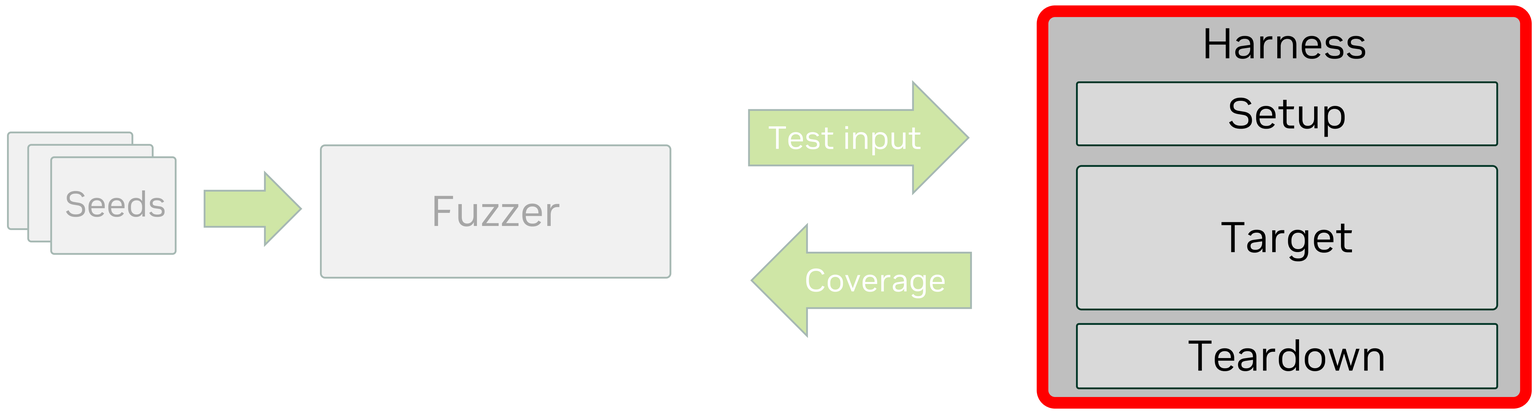


Challenge – fuzz harness creation

Trigger target **reliably** without crashing.

Small harness bugs kill **stability.** Proper setup requires code digging

Bad harness = zero bugs, no matter how smart the fuzzer is

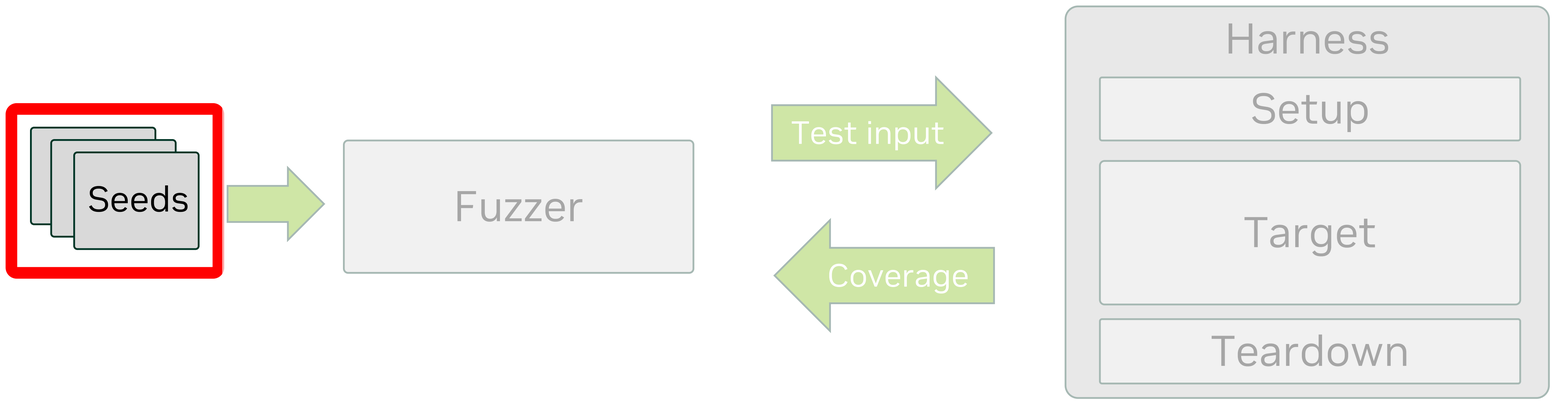


Challenge – initial input seeds

Seeds must reach the target. No crashes or early exits

Real targets rarely provide examples

Good seeds = early bugs. Bad seeds = wasted fuzzing cycles



Challenge – bug isolation and reproduction

Fuzzers find **crashes**, are they real **bugs**?

Repro needs **exact input + runtime context**

No repro = no triage, no fix

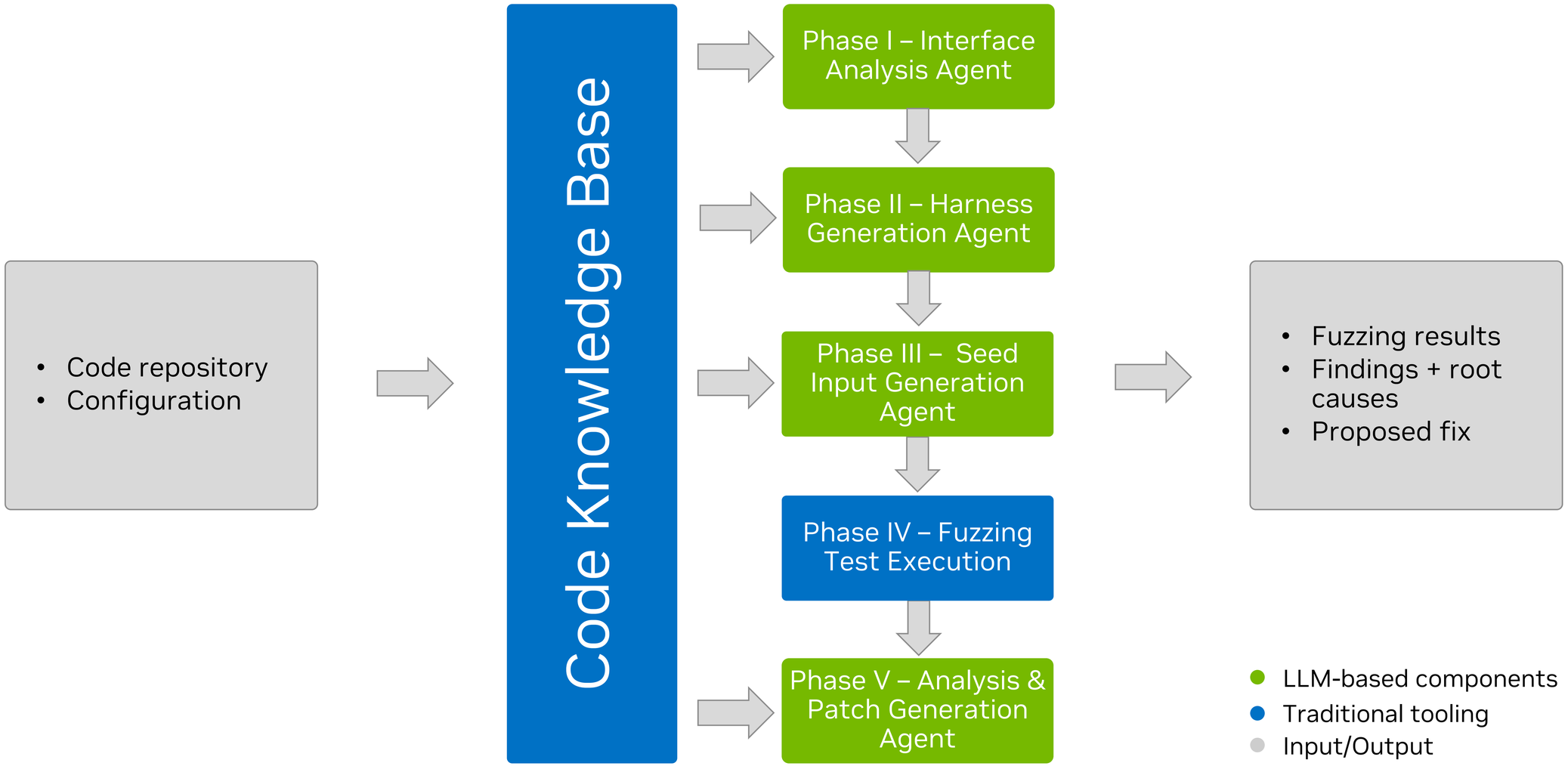
Challenge – crash triaging and remediation

Crash ≠ fix. Find root cause, not just symptoms

Triage = **context, deduping, blame mapping**

Good triage = faster patches, fewer regressions

Orion – high level design

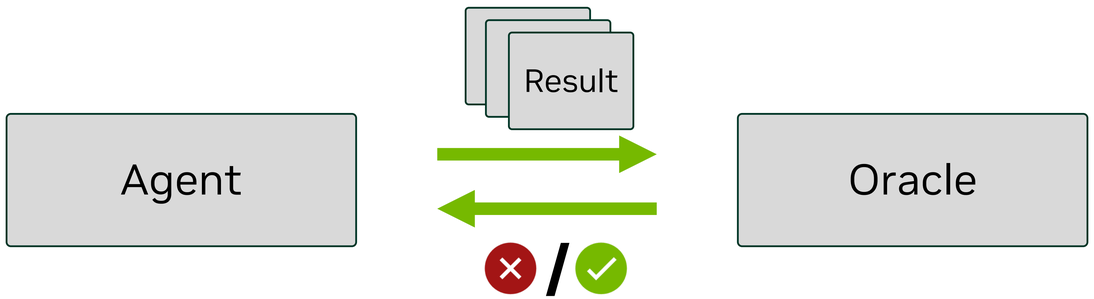


Feedback loops and oracles

Agents use tools to test themselves

Failure triggers **retries** - no human needed

Feedback = safe fail, fast recovery



Infrastructure for LLM-based code analysis

Context = key. Agents need code structure, not just prompts

We built parsing + graphing + sandboxed oracles

Result - **agents reason** like engineers, test, and fix

Benchmarking

Methodology

- Manual annotation over multiple targets
- Multiple evaluation metrics

Output verification via:

- Direct comparison
- Pair-wise relative ranking
- LLM-as-a-Judge

Targets

- **NVIDIA GPIO** driver (QNX)
- **CLIB** – C package manager
- **H3** – geospatial indexing lib

Orion – fuzzing target identification agent

What are high-risk targets?

- Perform calls associated with bugs
- Perform dangerous operations
- Widely used across the codebase
- High cyclomatic complexity
- Large size in lines of code

Orion – fuzzing target identification agent

What are high-risk targets?

- Perform calls associated with bugs
- Perform dangerous operations
- Widely used across the codebase
- High cyclomatic complexity
- Large size in lines of code

Agent output

Top 10 targets

41.0 - **clib_package_install** –
{'total': 41.0,
'sink_funcs': 9.2, 'danger_exprs': 8.8,
'loc': 9.0, 'internal_calls': 4.0,
'cyclomatic_comp': 10.0}

38.0 - **clib_package_new** –
{'total': 38.0,
'sink_funcs': 7.8, 'danger_exprs': 9.0,
'loc': 8.0, 'internal_calls': 9.2,
'cyclomatic_comp': 4.0}

30.4 - **clib_package_install_executable** -
{'total': 30.4,
'sink_funcs': 9.6, 'danger_exprs': 9.8,
'loc': 8.0, 'internal_calls': 1.0,
'cyclomatic_comp': 2.0}

Orion – dependency analysis agent

What do we need from analysis agent?

- Understand semantic dependencies
- Analyze interaction patterns with system state
- Identify function/module relationships
- Determine setup and teardown steps

Orion – dependency analysis agent

What do we need from analysis agent?

- Understand semantic dependencies
- Analyze interaction patterns with system state
- Identify function/module relationships
- Determine setup and teardown steps

Agent output

clib_package_install -

{'dependencies': [
{'description': ':Ensures the creation of directories for dependencies, fetches dependency files via network, and sets the 'PREFIX' environment variable'},
'functions': ['clib_package_install_dependencies']}
]}

clib_package_new -

{'cleanup': [
{''description': 'allocated struct clib_package_t needs to be deallocated'},
'functions': ['clib_package_free']}
]}

Orion – harness generation agent

What is a good fuzzing harness?

- Must be compatible with the fuzzer
- Provide the correct input to the target
- Invoke dependencies for the target
- Perform appropriate cleanup
- Verify the return values

Orion – harness generation agent

What is a good fuzzing harness?

- Must be compatible with the fuzzer
- Provide the correct input to the target
- Invoke dependencies for the target
- Perform appropriate cleanup
- Verify the return values

Agent output

```
#include <stddef.h>
#include <stdlib.h>
#include <string.h>
#include "../../src/common/clib-package.h"

int LLVMFuzzerTestOneInput(const uint8_t *data, size_t size) {
    if (size < 4) return 0; // Ensure we have enough data

    // Ensure null-termination and allocate space for the slug
    char slug[256]; memset(slug, 0, sizeof(slug));
    memcpy(slug, data, size > 255 ? 255 : size);

    // Create clib_package_t object
    clib_package_t *pkg = clib_package_new_from_slug(slug, 1);
    if (!pkg) return 0;

    // Ensure null-termination for the dir
    char dir[256]; memset(dir, 0, sizeof(dir));
    memcpy(dir, data, size > 255 ? 255 : size);

    // Fuzz the target function
    int result = clib_package_install(pkg, dir, 1);

    // Cleanup
    clib_package_free(pkg);
    return 0;
}
```

Orion – harness verification agent

Why do we need harness verification?

- Generated harnesses often contain errors
- Feedback loops allow iterative agent improvement.
- Compiler oracle checks syntax & build success.
- Compiler errors provide hints for agent correction.

Orion – harness verification agent

Why do we need harness verification?

- Generated harnesses often contain errors
- Feedback loops allow iterative agent improvement.
- Compiler oracle checks syntax & build success.
- Compiler errors provide hints for agent correction.

Agent output

Attempt 1
Compilation failed:

```
test/fuzzing/llm_harness.c
:36:18: error: read-only
variable is not assignable

36 | dir[max_len] = '\0';
| ~~~~~~~~~~~~ ^
1 error generated.
```

Attempt 2
Compilation successful.

Orion – input seed generation agent

What is a good input seed?

- Works correctly with the fuzzer/harness
- Includes values for conditional branches
- Incorporates constants (strings, arrays)
- Is valid input accepted by the target
- Provides good initial code coverage

Orion – input seed generation agent

What is a good input seed?

- Works correctly with the fuzzer/harness
- Includes values for conditional branches
- Incorporates constants (strings, arrays)
- Is valid input accepted by the target
- Provides good initial code coverage

Agent output

```
[{'value': 'install_dir', 'offset': 0},
{'value': 'pkgname', 'offset': 16},
...
{'value': 8, 'offset': 64}]
```

```
int clib_package_install(clib_package_t *pkg, const char
*dir, int verbose) {
...
    if (0 == opts.force && pkg && pkg->name) {
...
        if (pkg->src->len > 0) {
...

}
```

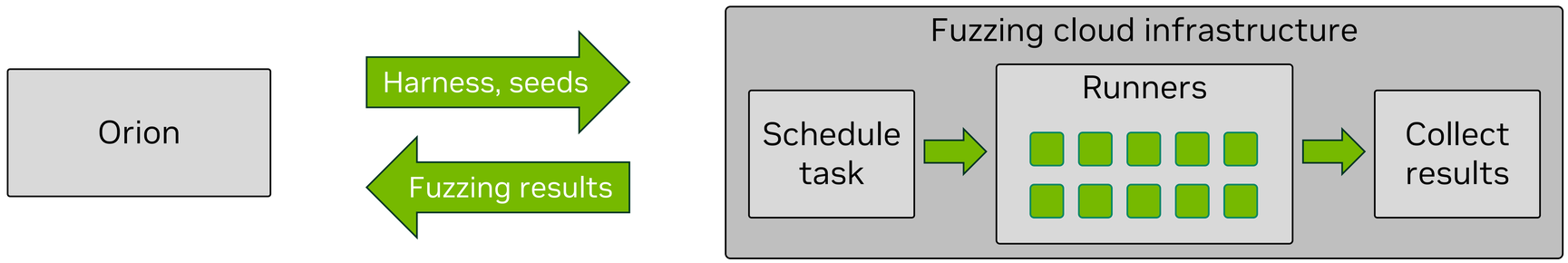
```
69 6e 73 74  61 6c 6c 5f   |install_|
64 69 72 00 00 00 00 00   |dir.....|
70 6b 67 6e 61 6d 65 00   |pkgname.|
00 00 00 00 00 00 00 00   |........|
...
08 00 00 00 00 00 00       |.......|
```

Orion – integration with fuzzing infra

Launch: Agents deploy harnesses/seeds to infra

Collect: Infra returns crashes + coverage

Adapt: Agents refine, retry, or triage automatically



Orion – crash analysis agent

What it does

Analyzes crash data to understand the root cause and generate minimal reproducers.

How it works

Parses stack traces, sanitizer logs, and source code for triage.

Output

Produces validated minimal test cases (via compiler/sanitizer) for debugging and patch testing.

Orion – patch generation agent

What makes a good generated patch?

- Compiles cleanly with no syntax errors
- Fixes the root cause, not just symptoms
- Avoids regressions or side effects
- Matches code style and integrates cleanly

Orion – patch generation agent

What makes a good generated patch?

- Compiles cleanly with no syntax errors
- Fixes the root cause, not just symptoms
- Avoids regressions or side effects
- Matches code style and integrates cleanly

Agent output

```
diff --git a/clib_package.c
b/clib_package.c
--- a/clib_package.c
+++ b/clib_package.c
@@ -1,7 +1,7 @@
int clib_package_install(clib_package_t
*pkg, const char *dir, int verbose) {
- int max = opts.con;
+ int max = (pkg->src->len < opts.con) ?
pkg->src->len : opts.con;
fetch_package_file_thread_data_t
**fetchs = malloc(sizeof(*fetchs) * max);
@@ -23,7 +23,7 @@
fetchs[i] = fetch;
pending++;
- if (i < (max - 1)) {
+ if ((i + 1) < max) {
i++;
} else {
```

Results – Interface Analysis

Benchmark results for GPT 4.1

95%

Overall

Performance across all metrics

77%

Dangerous Expressions

Pointer arithmetic, Pointer dereference, bit operations, etc.

94%

Sink Functions

Functions often associated with vulnerabilities

* More details: https://arxiv.org/abs/2509.15195

Results – Harness Generation

Benchmark results for GPT 4.1

85%

Overall

Success rate across all validation checks.

88%

Compiler verified

Harnesses compile and pass model-based verification.

84%

Required function calls

Includes all necessary function calls and dependencies.

* More details: https://arxiv.org/abs/2509.15195

Results – Time savings

Results for GPT 4.1

92x

Interface Analysis

Automatically identify key interfaces and entry points

204x

Harness Generation

Create working fuzzing harnesses with correct setup automatically

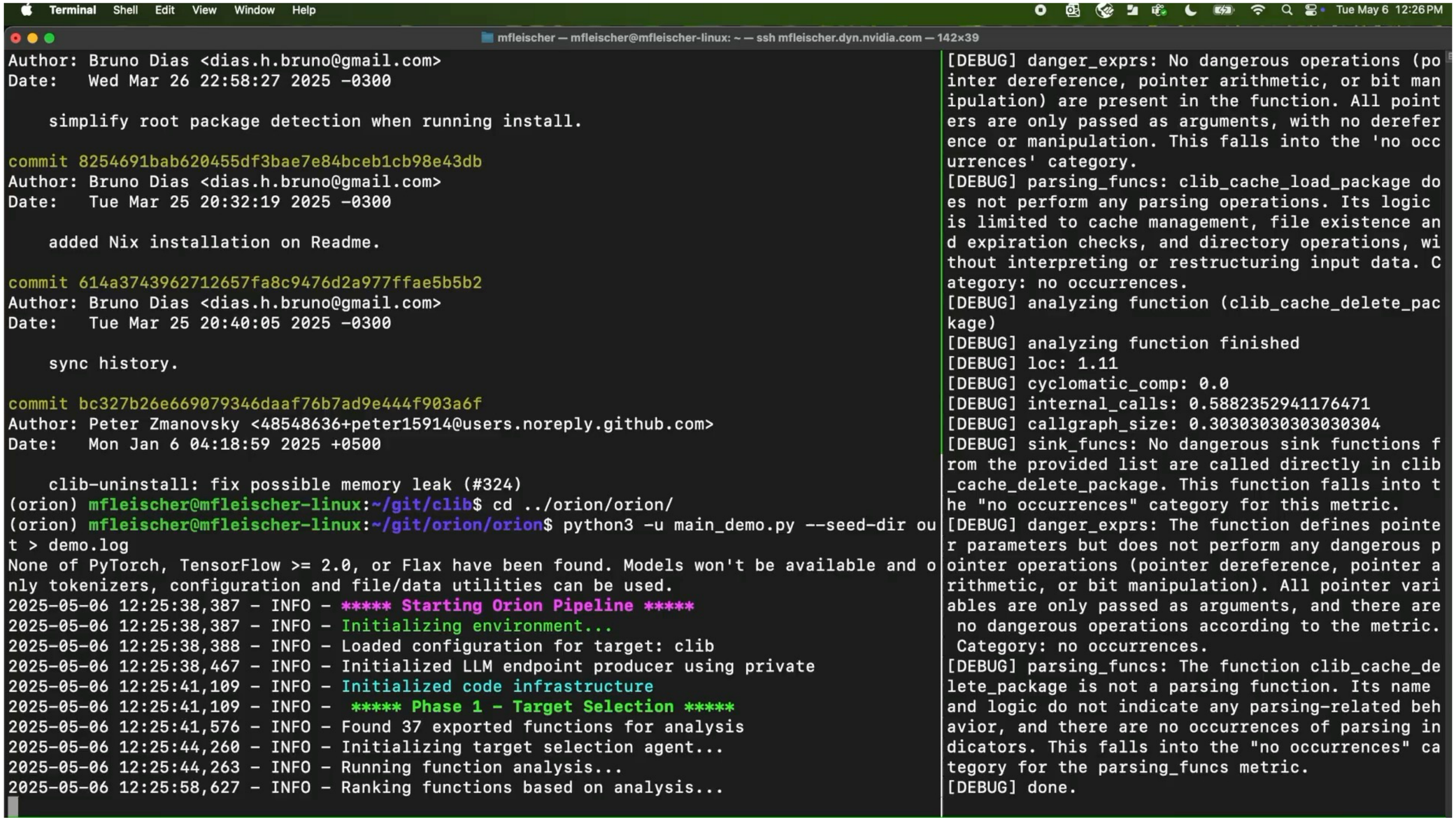
46x

Patching

Suggest and test fixes based on crashes automatically

* More details: https://arxiv.org/abs/2509.15195

Demo


Terminal Shell Edit View Window Help
Tue May 6 12:26 PM
mfleischer — mfleischer@mfleischer-linux: ~ — ssh mfleischer.dyn.nvidia.com — 142×39
Author: Bruno Dias <dias.h.bruno@gmail.com>
Date: Wed Mar 26 22:58:27 2025 -0300
simplify root package detection when running install.
commit 8254691bab620455df3bae7e84bceb1cb98e43db
Author: Bruno Dias <dias.h.bruno@gmail.com>
Date: Tue Mar 25 20:32:19 2025 -0300
added Nix installation on Readme.
commit 614a3743962712657fa8c9476d2a977ffae5b5b2
Author: Bruno Dias <dias.h.bruno@gmail.com>
Date: Tue Mar 25 20:40:05 2025 -0300
sync history.
commit bc327b26e669079346daaf76b7ad9e444f903a6f
Author: Peter Zmanovsky <48548636+peter15914@users.noreply.github.com>
Date: Mon Jan 6 04:18:59 2025 +0500
clib-uninstall: fix possible memory leak (#324)
(orion) mfleischer@mfleischer-linux:~/git/clib$ cd ../orion/orion/
(orion) mfleischer@mfleischer-linux:~/git/orion/orion$ python3 -u main_demo.py --seed-dir out > demo.log
None of PyTorch, TensorFlow >= 2.0, or Flax have been found. Models won't be available and only tokenizers, configuration and file/data utilities can be used.
2025-05-06 12:25:38,387 - INFO - ***** Starting Orion Pipeline *****
2025-05-06 12:25:38,387 - INFO - Initializing environment...
2025-05-06 12:25:38,388 - INFO - Loaded configuration for target: clib
2025-05-06 12:25:38,467 - INFO - Initialized LLM endpoint producer using private
2025-05-06 12:25:41,109 - INFO - Initialized code infrastructure
2025-05-06 12:25:41,109 - INFO - ***** Phase 1 - Target Selection *****
2025-05-06 12:25:41,576 - INFO - Found 37 exported functions for analysis
2025-05-06 12:25:44,260 - INFO - Initializing target selection agent...
2025-05-06 12:25:44,263 - INFO - Running function analysis...
2025-05-06 12:25:58,627 - INFO - Ranking functions based on analysis...
[DEBUG] danger_exprs: No dangerous operations (pointer dereference, pointer arithmetic, or bit manipulation) are present in the function. All pointers are only passed as arguments, with no dereference or manipulation. This falls into the 'no occurrences' category.
[DEBUG] parsing_funcs: clib_cache_load_package does not perform any parsing operations. Its logic is limited to cache management, file existence and expiration checks, and directory operations, without interpreting or restructuring input data. Category: no occurrences.
[DEBUG] analyzing function (clib_cache_delete_package)
[DEBUG] analyzing function finished
[DEBUG] loc: 1.11
[DEBUG] cyclomatic_comp: 0.0
[DEBUG] internal_calls: 0.5882352941176471
[DEBUG] callgraph_size: 0.30303030303030304
[DEBUG] sink_funcs: No dangerous sink functions from the provided list are called directly in clib_cache_delete_package. This function falls into the "no occurrences" category for this metric.
[DEBUG] danger_exprs: The function defines pointer parameters but does not perform any dangerous pointer operations (pointer dereference, pointer arithmetic, or bit manipulation). All pointer variables are only passed as arguments, and there are no dangerous operations according to the metric. Category: no occurrences.
[DEBUG] parsing_funcs: The function clib_cache_delete_package is not a parsing function. Its name and logic do not indicate any parsing-related behavior, and there are no occurrences of parsing indicators. This falls into the "no occurrences" category for the parsing_funcs metric.
[DEBUG] done.

Orion findings

Open-source findings

- Found multiple new bugs in CLIB, missed by OSS-Fuzz
- No details yet, bugs are not patched
- Maintainers have been notified

Internal impact

- Deployed across NVIDIA DRIVE stack
- Found ~100 new bugs: memory, logic, assertions
- Automated crash analysis helps a lot

Pitfalls of agentic workflows

- **LLMs don’t know your code:** reliable context infra is critical; prompting alone fails
- **LLMs can be unreliable:** they hallucinate, struggle with math, and mismanage state. Validation oracles are must
- **Agents ≠ silver bullet:** simple scripts often outperform overengineered agents

LLMs vs Tools

- **Tools > prompts** - some tasks are better outside the model like call graphs, line counts, type checks.
- **Tools make great oracles** – compilers, fuzzers, sanitizers give ground truth.
- **Trusted tools boost LLMs** – validate, retry, and correct outputs.

Benchmarking Lessons Learned

- **Classic benchmarking still matters** – think edge cases, coverage, etc.
- **LLM judges need ground truth** – oracles like compilers or test suites help.
- **Keep the judging task simple** – simpler tasks = more reliable evaluation.

Infra Lessons Learned

- **Bad context = bad output.** Retrieval accuracy makes or breaks agents.
- **Tooling gaps are real** – expect glue code or full custom wrappers.
- **Persistent memory = smarter agents** – reuse past results to refine output.

Open-Source Plans

- **Open sourcing in progress** – internal dependencies make decoupling hard.
- **Goal: release reusable modules** – reproducible and useful beyond Orion.
- **No hosted version** – bring your own fuzzing + LLM stack.

Q&A

Contacts

- **Max Bazalii** – mbazalii@nvidia.com
- **Marius Fleischer** – mfleischer@nvidia.com

Interested in more details?

Orion on arXiv