PAGENT: Automated PoC Generation for FFmpeg Vulnerabilities

Program Analysis Guided LLM Agent → Automatic Proof-of-Concept Generation

Authors: Tim Fan, Katelyn Dang, Aiden Hunter, Sachit Madaan | **Mentors:** Prof. Tevfik Bultan, Grad Mentor Achintya Desai

The Problem

- Security reports often identify a likely bug, but not a working **proof-of-concept (PoC)**
- Reproducing the crash manually is slow and labor-intensive
- Without a PoC, developers have a harder time validating and fixing vulnerabilities

Why It Matters

- Slows down vulnerability validation
- Increases developer effort
- Delays patching and security response

Why It's Difficult

- Large C/C++ codebase
- Complex, structured input formats
- Vulnerable function may be deep in the call chain
- One wrong input field can prevent the crash

Why FFmpeg

- Widely used multimedia framework
- C/C++ codebase
- Complex structured inputs
- Frequent target of vulnerability discovery

Beyond Fuzzing

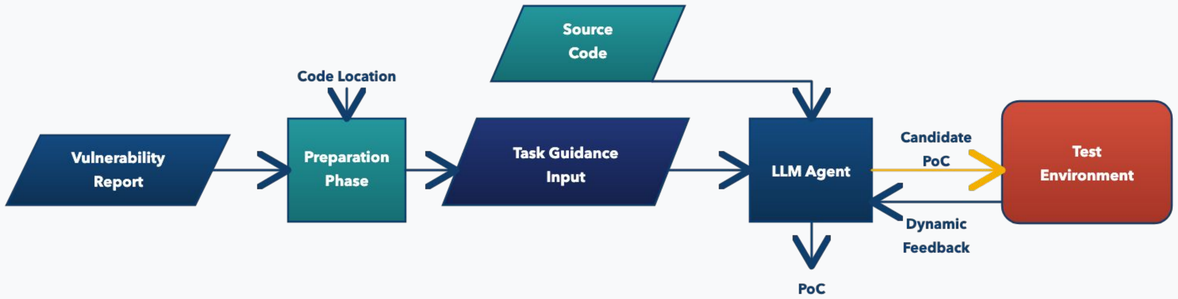
- Fuzzing is powerful, but inefficient for targeted vulnerability reproduction
- Explores many irrelevant inputs
- PAGENT combines static analysis and dynamic feedback to guide the agent through code paths

Static Analysis Example

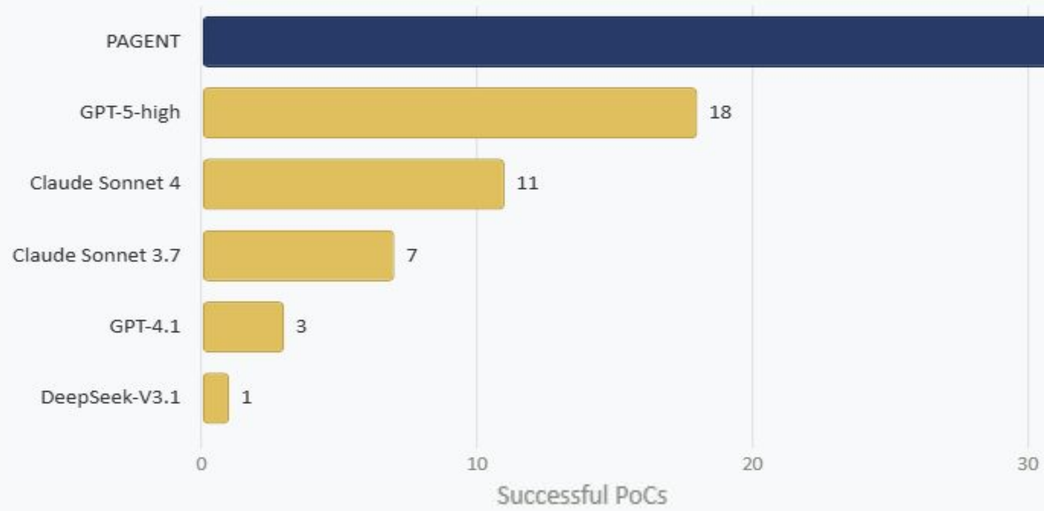
```
"potential_target_1": {
  "Vulnerability Type": "Integer-Underflow-Vulnerability",
  "Vulnerable Function": "NEG_USR32",
  "Entrypoint": "LLVMFuzzerTestOneInput",
  "Taint Path": ['LLVMFuzzerTestOneInput', 'av_parser_parse2',
    'h264parse', 'parse_nal_units', 'get_bits', 'NEG_USR32'],
  "Vulnerable Program Location": "142"
```

PAGENT: Program-Analysis Guided PoC Generator

PAGENT combines static analysis, dynamic execution feedback, and iterative LLM-guided refinement. The agent receives a vulnerability report + build, then iteratively mutates inputs and uses coverage feedback to improve subsequent attempts until a crash occurs.



PAGENT vs. Unguided LLM Baselines



Inside the Agent Loop

Results

33 Successful Proof-of-Concept Exploits Generated
for FFmpeg via PAGENT

Common Vulnerability Types

- Out-of-bounds Access
- Use-After-Free
- Use of Uninitialized Value

```
==12345==ERROR: AddressSanitizer:
heap-buffer-overflow on address 0x611...
READ of size 4 at 0x6110000... thread T0
  #0 in mov_read_packet  ffmpeg/libavformat/mov.c:8421
  #1 in av_read_frame
ffmpeg/libavformat/utils.c:1763
SUMMARY: AddressSanitizer: heap-buffer-overflow
```

Example: sanitizer confirms the generated input triggers the reported crash.

Engineering Contributions

- **33/69** successful PoCs (**47.8% success rate**)
- **18** post-patch PoCs
- **1** real bug/PoC still persists in FFmpeg's current version, causing DoS
- Outperformed all unguided baselines evaluated:
  - **83.3%** better than best performing unguided LLM: GPT-5-high

Average PoC Generation Time

- Successful PoCs: 39.4 minutes
- Failed PoCs: 123.18 minutes
- Combined Average: 84.5 minutes

Compared to a typical 24-hour fuzzing window, PAGENT reached successful PoCs 36x faster on avg.

Key Takeaways

Guidance Beats Guessing
GUIDED vs. UNGUIDED
Static analysis and coverage feedback make LLM PoC generation much more effective than unguided prompting.

Patch Validation
POST-PATCH CHECKING
Supports testing successful PoCs on patched builds to check if the vulnerability still exists after patching.

Days → Hours
FASTER VULNERABILITY RESPONSE
PAGENT cuts PoC development time from days of analyst effort to hours of automated agent runtime.

Future Work

Future work focuses on making PAGENT fully agentic and broadly applicable so that there is less manual setup with better results. An important direction to take is automating rule generation and environment setup for any software, allowing PAGENT to go beyond its current workflow.

Acknowledgements: Prof. Ziad Matni · Prof. Tevfik Bultan · Achintya Desai · Parker Carlson

References: PAGENT · ARVO dataset · FFmpeg fuzzing harness · OpenHands agent framework