APE: Adversarial Protocol Extraction Via an Iterative Multi-Agent LLM System

Gavin Crigger
College of Arts and Sciences
University of Virginia

Abstract

Black-box fuzzing campaigns often struggle with code/state coverage due to lack of information limiting effective input generation. Internet protocol fuzzing suffers excessively from this challenge due to its inherent statefulness, thus an iterative protocol extraction and state inference tool is needed to improve existing and future fuzzing methods. APE (Adversarial Protocol Extractor) serves as a tailored application to the space with underwhelming results but promising room for improvements. It is a multi-agent system with a grammar agent, probe agent, critique agent, and finite state machine builder. Generally poor results were found through evaluating APE via BLEU/ROUGE scores and word error rates upon Live555, Kamailio, and PureFTPd protocol implementations. Key findings were a gap in diverse probing, ineffective protocol identification, and early plateaus in grammar consensus. Future work should target explorations into ways the system may be improved as well as more effective evaluations (ChatAFL campaigns with APE grammars). Artifacts can be found at https://github.com/gavinvc/llm-grammar-extraction.

1 Introduction

For many systems, internet protocol implementations represent the most exposed points of entry that adversaries may exploit. Proper implementations of such protocols are essential to providing services outside of the system's network, yet any live errors can lead to intrusions, data leakage, or resource abuse/exhaustion. These security-critical components thus necessitate exhaustive security evaluation to ensure system integrity. Administrators should have tools to point a fuzzing campaign at an exposed endpoint with no prior information on the system, as this is the threat model of many adversaries attempting to intrude on networks. While prior research has touched on many aspects of such a tool, there does not exist a comprehensive toolkit for investigating how an attacker (or security tester) can point resources at an endpoint and discover exploitable bugs.

This is where APE comes in. Under the scenario of a black-box security analysis campaign, APE is given an endpoint and sends diagnostic requests to iteratively build a grammar and state machine based on both holistic and local extraction techniques. While it is generally ineffective in its current state, APE is a modular system with extensive room for improvement in future work.

2 Background

Generating an appropriate and diverse seed corpus for a fuzzing campaign is often the first hurdle to achieving effective results. This is trivial when one has the time, resources, and domain expertise to dedicate to manual input generation, yet such a set of circumstances is not realistic. Black-box fuzzing, where security testers lack information about the underlying system that they are testing, introduces the constraint of generating inputs that result in adequate code and state coverage while being in the dark with respect to implementation specifics. Research methods dedicated towards answering the question of how to effectively generate input structures for binaries or other unknown system specifications have thus been an important focus of the software security testing community.

Input generation for protocol fuzzing introduces the additional challenge of statefulness [2]. Fuzzing stateful systems is difficult because of the chain of inputs that they require in order to achieve specific state transitions - add this to a black-box fuzzing scenario and you have a challenging yet realistic set of fuzzing goals. There are several fuzzing works that touch on parts of this problem, yet none that provide a comprehensive approach to extracting a protocol's grammar and state machine given a black-box environment with no prior traffic data.

The researchers behind *LLM-Boofuzz* identify the shortcomings of *ChatAFL* [2], which fails to adapt to unknown protocol scenarios, and *ArtifactMiner* [3], which depends heavily on initial sample quality. They state that current research neglects protocol extraction through parsing real traffic

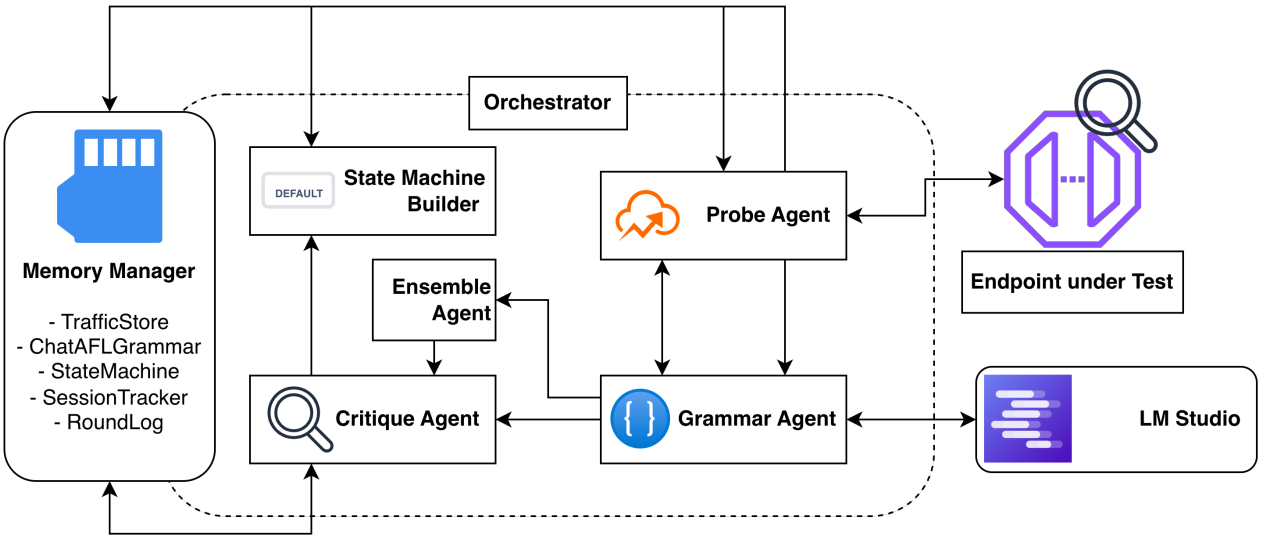

Figure 1: High-level system architecture of APE; agents interact under an orchestrator service alongside a memory manager, and specific agents interact with external endpoints/services.

with an LLM, then present a methodology that cleans traffic packets into LLM-interpretable text that is then used to query an LLM and extract a protocol while adapting to statefulness [4]. Zhang et. al present a different approach to using LLMs in input generation, employing an LLM to perform a "holistic search" while industry-standard fuzzing methods operate on the local level in G^2Fuzz [6]. Additionally, useful metrics for evaluating the success of protocol extraction are put forth in Maklad et. al's RAG-based LLM inference of protocol finite state machines, providing insight into state evaluation alongside protocol extraction [1].

3 Methods

I approach the problem of adversarial protocol extraction by developing a multi-agent LLM system designed to interact with an open endpoint to learn as much about the protocol implementation possible. In line with ChatAFL's approach, the system consults with an LLM API that will thus forward be referred to as the PLLM (partner LLM). The PLLM used for these methods is `qwen/qwen2.5-coder-7b-instruct-mlx` run via LM studio to mimic an OpenAI API endpoint that can be queried at no-cost.

3.1 Agents

There are three core agents to this multi-agent system focused on their own respective functions: probing, grammar generation, and artifact critique.

3.1.1 Probe Agent

Traffic generation and collection is driven primarily by the probe agent, which consults with the grammar agent, memory manager, and PLLM to generate context-driven message sequences to send to the endpoint-under-test. A general set of seed message sequences serve as the initial probes to the endpoint, and the resulting message-response pairs are used to generate future probes. After this, the agent runs a number of grammar guided, PLLM suggested, and state machine guided probes against the endpoint. All of the generated traffic is parsed and sent to the memory manager, which parses and records the traffic into iterable objects.

3.1.2 Grammar Agent

The grammar agent uses traffic exchanges generated by the probe agent as context when querying the PLLM, instructing it to construct a ChatAFL-style grammar that the agent then parses. If the post-query parsing fails, the agent retries the query with more explicit instructions. Because this is an iterative process, the grammar agent may already have pieced together a grammar by the time it queries the PLLM - in this case, the agent stitches together any newly discovered grammar components. The grammar agent also handles PLLM-assisted inference of the protocol's state machine, which is then used by the state-machine-builder agent to construct a fully structured state machine.

3.1.3 Critique Agent

After a grammar is generated via the grammar agent, a critique agent assesses and issues fixes to the ChatAFL-style grammar. There are six key goals in this stage: formatting, traffic grounding, cross-protocol grounding, removing header contamination, assessing RFC/known protocol matches, and assessing whether the campaign has plateaued. Based on each of these critiques, the agent issues patches as either deletions, rewrites, or additions across several iterations. The final, patched-up grammar is sent back to the memory manager to persist across rounds. If the critique agent determines that a plateau has been reached, it will output the current artifacts and prompt the user to decide on how the campaign should proceed. At this point, the user may instruct APE to stop the campaign early, take a different approach to the grammar, or ignore the plateau and continue as it has.

3.1.4 State Machine Builder

A state machine builder agent handles construction of a finite state machine throughout the APE run by considering seen-states and transitions in the system's memory. While not exactly an LLM agent, this is a key piece of the architecture that gives the user an idea of how the overall probing and grammar progress fits in with their own understanding of the endpoint. Alongside a `visualize_fsm.py` script, the state machine "agent" effectively illustrates the extent of APE-derived knowledge throughout the extraction run. The exact memory context used is RoundContext objects which contains core events relevant to the round.

3.2 Memory Manager

Each APE campaign has a memory manager dedicated to long-term storage of grammar, state machine, traffic, and round information. Traffic records are stored following LLM-Boofuzz's methodology, retaining all structured text components and the first 50 raw bytes from the traffic as records in a TrafficStore object [4]. The manager also stores the entire grammar history, giving the user not only the final grammar after the campaign but also the round-by-round playback on how the system got to that point. Another key piece of the memory manager is the session trajectory store, which retains logs on where each probe is at in its sequence of messages and corresponding states on the endpoint under test. This informs the state machine generation, which reflects on session trajectories before storing the current set of concrete, visited states and their transitions based on the trajectories.

3.3 Orchestrator

The orchestrator serves as the skeleton behind the multi-agent system, containing one of each agent and an overarching memory manager that links their functionalities together. Runtime configurations are handled by the orchestrator, which oversees the round-to-round management of the protocol extractor.

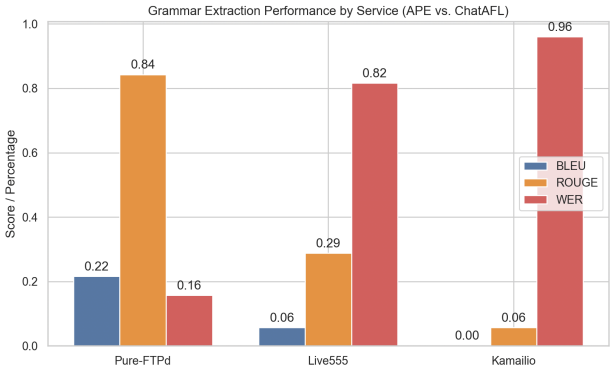

Figure 2: Bar chart of overall result averages

4 Evaluation

I evaluate the grammar produced by APE against ChatAFL-produced grammars. I do so through three core text-based scores: a Bilingual Evaluation Understudy (BLEU) score, Recall-Oriented Understudy for Gisting Evaluation (ROUGE) score, and Word Error Rate (WER). This effectively acts as precision, recall, and error respectively, giving a holistic overview on the similarity between APE grammar and tried-and-true ChatAFL grammars. ChatAFL grammar results were almost exactly the same as expert-derived protocol grammars, and they thus serve as an appropriate target for APE's grammar. In particular, the BLEU score represents precision through assessing n-gram appearance in reference test while the ROUGE score tackles recall through n-gram overlap between output and reference text [5].

5 Results

Results were underwhelming across the three benchmarks, with PureFTPd (FTP) having the best scores followed by Live555 (RTSP) and Kamailio (SIP).

5.1 PureFTPd

The PureFTPd runs were the most effective, requiring no hints and achieving the best results. With an average BLEU score of 0.22, ROUGE score of 0.84, and WER of 0.16, the APE-generated grammar was not entirely precise but did demonstrate effective recall and low word error. I postulate that this general success was due to my system being built with examples of FTP grammars, thus equipping it for more success in a PureFTPd setting.

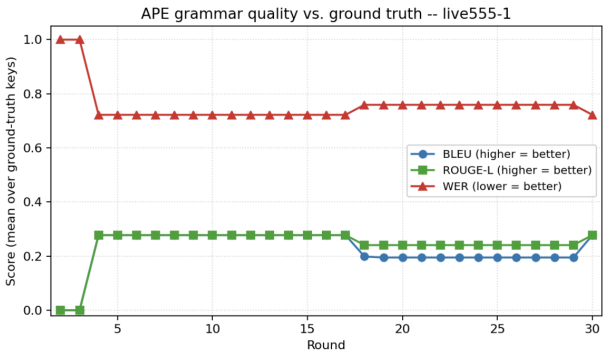

Figure 3: Live555 APE extraction run over 30 rounds, with BLEU and ROUGE scores as well as WER.

5.2 Live555

The shift to Live555 from PureFTPd demonstrates a significant degradation in performance. With a BLEU score of 0.06, ROUGE score of 0.29, and WER of 0.82, the grammar is simply nowhere near shaping up to the ChatAFL-generated grammar. Additionally, Live555 begins the trend of non-FTP grammars requiring hints passed to APE at runtime. Although there is a fairly clear set of instructions and protocol shown to the user upon initial curl to the Live555 endpoint, APE fails to understand what protocol implementation it is looking at and continues to label it as "unknown" unless being explicitly told. This shows how grammar inference is failing at some point in the architecture.

5.3 Kamailio

Kamailio had an average BLEU score of 0.00, ROUGE score of 0.06, and WER of *0.96* - a complete failure to extract a grammar similar to that of ChatAFL. It fails to identify what protocol Kamailio implements and provides no valuable grammar in the process of extraction. This may serve as a useful stretch target as a result, as APE demonstrates that it is not in a position to tackle complex implementations like Kamailio.

6 Discussion

While these results are nowhere near the ground-truth results of ChatAFL-generated grammar, I contend that this does not serve as an indicator that low-resource, open-source, black-box grammar extraction is ineffective as a whole. There is a significant space for extension of this problem that I intend to explore in the future. The fact that small improvements were invoked through additional context (like hints given to APE at runtime or more specific prompting throughout development) indicate that there is a wide space for improvement. One key improvement space is protocol inference, as PureFTPd

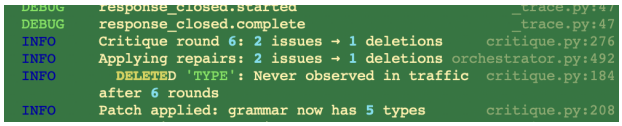

Figure 4: Screenshot of critique agent resulting in an update to the running grammar, deleting a potentially-hallucinated grammar that did not appear in traffic

demonstrated far more effective results while also showing that APE recognized the protocol - a combination that could be extended to Kamailio and Live555. Finally, Figure 3 shows how the grammar extraction pretty quickly reaches a plateau where metrics are hardly improving (or changing at all) - as a result, I am led to believe that there may be improvement in more diverse probing and Escaping plateaus.

7 Future Work

There is *significant* room for future work as I ran out of time to evaluate the extent of customization applicable to this architecture. Such customizations include prompt engineering to get more effective results per LLM call, different LM Studio open-source LLMs, and actually diving into the potential effects that an orchestrator service could have. Additionally, I ran out of time to fork, instrument, and run APE grammars in a ChatAFL campaign - this would give me a grounded set of metrics that I could use to compare with ChatAFL results. Thus, future work should (and will, in my future free time) assess the improvement areas my architecture proposes as well as more robust benchmarks.

8 Conclusion

Although APE was unsuccessful at deriving ChatAFL-grade grammars, I put forth a multi-agent system for probing and extracting grammar information from black-box endpoints. There is a vast area of improvement here, from different LM Studio models to prompt engineering. I hope to spend my summer investigating ways to make APE a successful system and derive low-cost fuzzing success as a result.

References

[1] MAKLAD, Y., WAEL, F., ELSERSY, W., AND HAMDI, A. Retrieval Augmented Generation Based LLM Evaluation for Protocol State Machine Inference with Chain-of-Thought Reasoning. In *Lecture Notes in Networks and Systems* (2025), vol. 1416, pp. 313–323.

[2] MENG, R., MIRCHEV, M., BÖHME, M., AND ROYCHOUDHURY, A. Large Language Model Guided Protocol Fuzzing. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)* (2024).

[3] SHARMA, P., AND YEGNESWARAN, V. PROSPER: Extracting Protocol Specifications Using Large Language Models. In *Proceedings of the 22nd ACM Workshop on Hot Topics in Networks (HotNets '23)* (2023), pp. 41–47.

[4] WANG, T., LI, Y., PAN, Z., CHEN, Q., LI, Z., ZHANG, Y., SHEN, Y., HU, M., AND CHEN, Q. LLM-Boofuzz: Generation-Based Black-Box Fuzzing for Network Protocols via LLMs. *Electronics 14*, 23 (2025), 4550.

[5] YANG, A., LIU, K., LIU, J., LYU, Y., AND LI, S. Adaptations of rouge and bleu to better evaluate machine reading comprehension task. pp. 98–104.

[6] ZHANG, K., LI, Z., WU, D., WANG, S., AND XIA, X. Low-Cost and Comprehensive Non-textual Input Fuzzing with LLM-Synthesized Input Generators. In *34th USENIX Security Symposium (USENIX Security 2025)* (2025), pp. 6999–7018.

Notes

This research is performed as part of *CS 6501 - Software Security Testing* at the University of Virginia. It has not gone through any review processes and is limited to course resources.