2025 IEEE/ACM International Conference on Automation of Software Test (AST) | 979-8-3315-0179-2/25/$31.00 ©2025 Authors | DOI: 10.1109/AST66626.2025.00021

Bringing Light into the Darkness: Leveraging Hidden Markov Models for Blackbox Fuzzing

1st Anne Borcherding
Fraunhofer IOSB
Karlsruhe, Germany
anne.borcherding@iosb.fraunhofer.de
0000-0002-8144-2382

2nd Mark Giraud
Fraunhofer IOSB
Karlsruhe, Germany
mark.giraud@iosb.fraunhofer.de
0000-0002-2972-2758

3rd Johannes Häring
Karlsruhe Institute of Technology
Karlsruhe, Germany
johannes.haering@student.kit.edu
0000-0002-0681-4959

***Abstract*—Securing the network interfaces of industrial control systems is essential for protecting critical infrastructure like water treatment plants and nuclear centrifuges from potential attacks. A key strategy to mitigate risks of successful attacks involves identifying and closing vulnerabilities exploitable through network interfaces using testing techniques such as fuzzing. While established techniques exist for graybox fuzzing, which assume access to system binaries, industrial components often require blackbox testing due to the use of third-party components and regulatory constraints. We propose `Palpebratum`, an approach that leverages Hidden Markov Models to approximate missing information in blackbox test scenarios. We evaluate `Palpebratum`'s performance in terms of code coverage, comparing it with two baseline blackbox fuzzers and the graybox fuzzer `AFLnwe`. Our results demonstrate that `Palpebratum` significantly outperforms one blackbox fuzzer, achieving an average of 4,379.33 basic blocks compared to 4,307.60 (p-value < 0.001). For the second blackbox fuzzer, `Palpebratum` achieves comparable coverage but with only half the number of test cases, demonstrating effectiveness despite the Hidden Markov Model's overhead. These findings suggest that `Palpebratum` enhances blackbox test case generation and emphasizes the importance of an efficient implementation to offset the added overhead.**

***Index Terms*—operational technology, fuzzing, blackbox, graybox, coverage-guided, hidden markov models.**

I. Introduction

Attacks such as `Stuxnet` and `Triton` demonstrate that critical infrastructure, particularly the Operational Technology (OT) components within it, is the target of attackers [1, 2]. For such attacks to succeed, the OT components—and especially their network stacks—must contain vulnerabilities. Research by Anton et al. [3] and vulnerability disclosures like `Ripple20` [4, 5] indicate widespread vulnerabilities among OT components.

One approach to reduce the likelihood of a successful attack is to minimize vulnerabilities in an OT component. This necessitates integrating security considerations throughout all phases of the development lifecycle [6]. In particular, system-level testing is essential for identifying and addressing any existing vulnerabilities.

The testing approach we consider for this work is *fuzzing*. Fuzzing involves generating test cases using randomness and sending them to the System under Test (SuT). In parallel, the SuT's behavior in response to these test cases is monitored to detect anomalies or crashes. Fuzzing approaches can be classified in several ways, with one prominent distinction being the level of information available to the fuzzer [7]. We categorize fuzzers as follows: (1) *blackbox* fuzzers, which lack access to internal information of the SuT; (2) *graybox* fuzzers, which have access to binaries and coverage information; and (3) *whitebox* fuzzers, which have full access to the source code. In graybox fuzzing, several sophisticated approaches have been proposed. These include mutational coverage-guided fuzzing, implemented by fuzzers like `AFL` [8] and its successors `AFL++` [9], `AFLNet` [10], `AFLnwe` [11], and `LibAFL` [12]. This approach uses information on the SuT's behavior, represented by the code coverage achieved by a given test case, to assess the *interestingness* of a test case. Test cases deemed interesting are used for further testing, while others are discarded, allowing for more efficient testing. It is important to note that employing this graybox approach requires access to coverage information during testing.

However, when testing OT components, blackbox approaches, though often essential, lack access to coverage information. Blackbox testing is necessary because OT components frequently consist of third-party elements [13], such as network stacks, for which manufacturers may not have access to the source code. Consequently, manufacturers must rely on blackbox testing to evaluate the OT component at the system level. Furthermore, relevant standards mandate blackbox testing in industrial environments [6].

To bridge the gap between the blackbox testing required for OT components and the techniques used in graybox testing, we propose a method to approximate the missing information necessary for graybox mutational coverage-guided fuzzing. Our approach enables the application of established graybox techniques within a blackbox testing environment.

Approach: To provide a blackbox surrogate for observing the SuT's behavior in graybox fuzzing, we train a Hidden Markov Model (HMM) on the network traffic of the network protocol used by the SuT. This HMM captures the SuT's behavior by modeling it as a probabilistic state machine with state-specific probabilistic emissions. During testing, we query the HMM to determine the most likely state sequence leading to the observed network traffic. The resulting transition coverage within the HMM's state machine serves as a surrogate for the coverage information available in graybox testing.

2833-9061/25/$31.00 ©2025 IEEE
DOI 10.1109/AST66626.2025.00021

Our approach, Palpebratum, is named after the bioluminescent fish species *Photoblepharon Palpebratum*, which illuminates its surroundings in the deep ocean using light-producing organs below its eyes [14]. In a similar fashion, our approach seeks to illuminate blackbox fuzzing by incorporating elements of graybox fuzzing.

Research Questions: Our work is driven by the following research questions.

RQ1 How does the behavior approximation of the HMMs compare to that of the graybox fuzzer AFLnwe?

RQ2 Does the HMM-based behavior approximation improve blackbox fuzzing in terms of code coverage?

RQ3 How does the network data preprocessing impact the performance of both the HMMs and the fuzzers?

Results: Our experiments addressing RQ1 demonstrate that the HMMs identify fewer test cases as interesting compared to AFLnwe, with notable differences in the specific test cases flagged by each method. This suggests that the HMMs model the SuT's behavior differently from AFLnwe. This is expected, given that the HMMs rely on network traffic, whereas AFLnwe has access to graybox coverage information.

To address RQ2, we evaluate two instances of Palpebratum, each employing a different HMM and network packet preprocessing method, against two blackbox baseline fuzzers and AFLnwe. Our experiments reveal that one of the instances, $\texttt{Palpebratum}_{\text{AE}}$, significantly outperforms one blackbox baseline in terms of coverage achieved over 24 hours, while achieving coverage comparable to the second baseline. This is achieved despite the overhead introduced by the HMMs, which halves the number of test cases sent within 24-hours. These findings indicate that although the HMMs provide valuable information, the associated overhead limits their effectiveness in our experimental setup. This suggests an opportunity to further enhance Palpebratum's final coverage by optimizing the HMM's overhead.

To address RQ3, we compare the performance of the two instances of Palpebratum. Our experiments reveal a significant performance difference between the instances, indicating that preprocessing has a substantial impact on the downstream fuzzer's performance.

Contributions: In summary, our work makes the following contributions:

1) Present a method for preprocessing network traffic to be used by HMMs.
2) Assess the performance of HMMs trained on the preprocessed network traffic for their effectiveness with respect to interestingness assessment.
3) Implement blackbox fuzzers that apply graybox techniques by using an HMM as a surrogate for missing information.
4) Evaluate the performance of these fuzzers against two baseline fuzzers using the File Transfer Protocol (FTP) implementation ProFTP as SuT.

Structure: In Section II, we present the foundations related to mutational coverage-based fuzzing and HMMs, which form the basis of our approach, Palpebratum. Section III details our approach, while Section IV outlines our methodology. Sections V to VII present and discuss the experiments we conducted regarding network data preprocessing, the performance of HMMs, and the performance of HMM-based fuzzers. A consolidation of these results and discussions, including insights with respect to our research questions, is provided in Section VIII. Section IX discusses work related to Palpebratum, and Section X suggests possible future research directions. Finally, Section XI concludes this work.

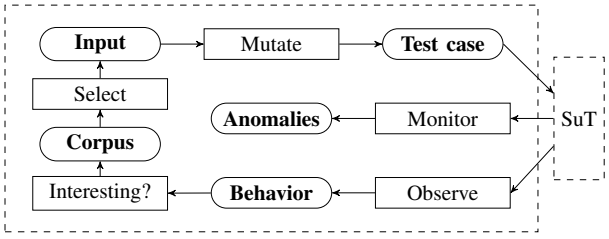

Fig. 1. Overview of the coverage-guided mutational graybox fuzzing process. The components of the fuzzer are enclosed within the dashed lines. Each fuzzing cycle begins with selecting an input from the corpus, which is initialized with a set of seed inputs and evolves throughout the fuzzing process.

II. Foundations

This section presents the foundations of mutational coverage-based fuzzing and HMMs, which form the basis of Palpebratum.

A. Mutational Coverage-Guided Fuzzing

Fuzzing is a long-standing testing technique that remains highly effective in revealing new bugs and vulnerabilities [15]. It is particularly effective in finding bugs within network stacks, which are especially pertinent to OT component security [16].

The core approach of fuzzing involves providing inputs, known as *test cases*, to the SuT. These test cases may be malformed, either semantically or syntactically. The behavior of the SuT in response to these test cases is monitored to identify anomalies and crashes [7]. Based on this general approach, several subdomains of fuzzing have emerged [17]. For this work, we focus on blackbox fuzzing and mutational coverage-guided graybox fuzzing.

Mutational graybox fuzzers typically require a set of valid input files for the SuT, referred to as *seeds*. Depending on the SuT, this could include a set of PDF files or network packets. A single run of a fuzzer is called a fuzzing *campaign*, typically bounded by a predefined time budget [18]. While some approaches stop after detecting the first crash of the SuT [19], Liyanage et al. suggest an adaptive approach [20]. During the fuzzing campaign, the fuzzer maintains a set of inputs known as the *corpus*. This corpus is initialized with seeds and evolves over the course of the campaign.

One fuzzing campaign consists of several fuzzing *cycles*, each comprising the steps visualized in Fig. 1. At the start of a fuzzing cycle, the fuzzer selects an input from the corpus

based on a certain selection strategy. This input is then mutated using a predetermined or dynamic set of mutations to generate a new test case t. The test case t is subsequently sent to the SuT. The behavior of the SuT is analyzed in two ways. First, the fuzzer *monitors* the SuT's behavior to detect anomalies or crashes. For example, address sanitizers can be used to monitor memory accesses by the SuT and alert the fuzzer if out-of-bounds accesses occur [21]. Test cases leading to anomalies and crashes are stored to be reported to the tester. Second, the fuzzer *observes* additional behavior of the SuT, such as the code coverage achieved by t, which can be measured when performing graybox fuzzing. If t uncovers previously unexplored parts of the SuT, t is deemed *interesting* since it provides access to reach new areas of the SuT's code. Consequently, t is added to the corpus and can be selected for further mutations in subsequent fuzzing cycles.

In blackbox fuzzing, no information about the internals of the SuT is accessible, and the SuT can only be interacted with via external communication interfaces [7]. For OT component fuzzing, this typically includes external network interfaces. In blackbox testing, the fuzzer's ability to monitor and observe the SuT is limited. As a result, most blackbox fuzzers do not incorporate observations of the SuT's behavior in the test case generation [19] and only detect crashes that are externally visible [22].

B. Hidden Markov Models

HMMs are used to represent dynamic processes over time where the current state is not directly observable [23]. Instead, only the emitted observation symbols are observable, resulting in an *observation sequence*. Thus, HMMs account for two key aspects of a process's uncertainty: (1) the probability of the next state S_{t+1} given the current state S_t, denoted as $P(S_{t+1} \mid S_t)$, and (2) the probability of an observation O_t given a state S_t, denoted as $P(O_t \mid S_t)$.

In an HMM, the *hidden*, non-observable states are modeled as a Markov chain, which adheres to the Markov property. According to this property, the next state S_{t+1} depends only on the current state S_t and not on any preceding states:

$$P(S_{t+1} \mid S_t, S_{t-1}, \dots) = P(S_{t+1} \mid S_t). \tag{1}$$

1) Tasks: Given an HMM, we are usually interested in the following three tasks [23, 24]: Evaluation, decoding, and learning. For each task, we assume a given observation sequence O and an HMM λ. The Evaluation task consists of computing the probability of the observation sequence: $P(O \mid \lambda)$. This task is addressed using the Forward algorithm. For Decoding, we need to compute the most probable state sequence that leads to O. This problem can be solved using the Viterbi algorithm. The Learning task adjusts the HMM parameters to maximize the probability of O: $P(O \mid \lambda)$. This corresponds to training the HMM and is typically approached with the Baum-Welch algorithm.

2) Multivariate HMMs: In domains such as speech recognition, univariate HMMs are used, which emit scalar values. In contrast, *multivariate* HMMs are designed to handle multidimensional observations [25]. This allows them to model more complex and richer patterns in the data.

3) Second-order HMMs: Typically, an HMM is based on a first-order Markov chain, where the state transition probability depends only on the current state and not on previous states (as described in Eq. (1)). Second-order Markov chains extend this concept by making the state transition probability dependent on both the current state and the preceding state [26]. This extension enables the model to capture more complex relationships between the states. However, a second-order Markov chain M_2 with N states can also be modeled by a first-order Markov chain M_1 with N^2 states [27]. In this representation, each state of M_1 corresponds to a pair of states from M_2.

III. Approach

The goal of our research is to analyze how blackbox information can be used to guide a fuzzer. To this end, we propose `Palpebratum`, which uses an HMM to process the network traffic generated by the fuzzer and the SuT, providing an approximation of a test case's interestingness as it is used in graybox fuzzing. Prior to the fuzzing test, we train the HMM on user-generated network traffic, including the communication protocols considered during the fuzzing test. During fuzzing, the HMM is utilized as shown in Fig. 2 and described in the following paragraph.

As the fuzzer generates and sends a test case to the SuT, the request and subsequent responses of the SuT are recorded. Each of the observed network packets is then preprocessed, and the sequence of these preprocessed network packets is presented to the HMM. Using the Viterbi algorithm, the most likely state sequence that led to the observed network packets is calculated (see Section II-B). If this state sequence contains state transitions in the HMM's state machine that have not been covered by previous test cases, the corresponding test case is considered interesting. This definition closely relates to the definition of interestingness in graybox fuzzing, which is usually based on whether a test case hits new basic blocks in the program [18]. As in graybox fuzzing, the HMM-based interestingness approximation is then used to guide the fuzzer (see Section II-A). Note that no re-training of the HMM is conducted during fuzzing to reduce the overhead.

IV. Methodology

Implementing and evaluating `Palpebratum`requires several essential building blocks. We provide the code necessary to recreate our experiments as well as our data on GitHub[1].

First, the network traffic the HMMs are trained and decoded on needs to be preprocessed to be usable by the HMMs. To this end, we implement a preprocessing pipeline based on two publications from the literature [28, 29]. This pipeline includes methods for dimensionality reduction, and the performance

[1]https://github.com/anneborcherding/palpebratum

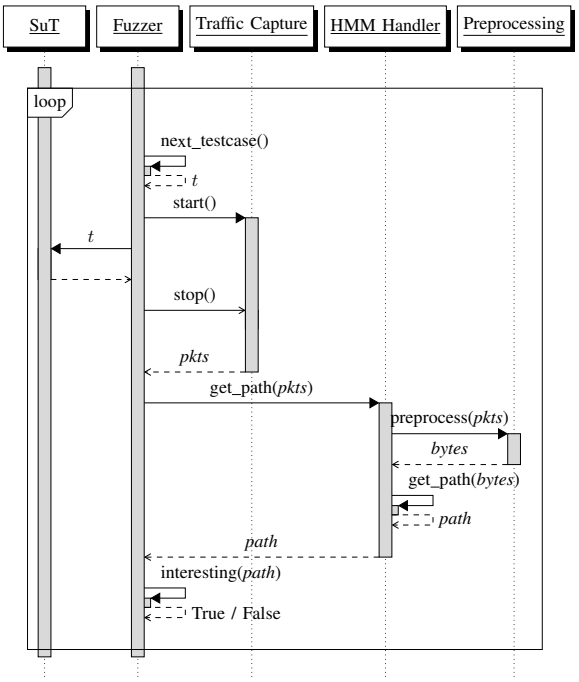

Fig. 2. Sequence diagramm of the novel HMM-based blackbox fuzzing. We capture the network traffic while a test case is sent to and processed by the SuT. This network traffic is then preprocessed and the HMM is queried for the most likely state sequence for this data. This state sequence is then used to assess the interestingness of a test case.

of three options for this dimensionality reduction is assessed by measuring their respective reconstruction error on network traffic generated during fuzzing. We provide a summary of these experiments and the results in Section V.

Second, we analyze how HMMs trained on the preprocessed network traffic perform with respect to the interestingness of fuzzing test cases (RQ1). To this end, we train HMMs with different numbers of nodes on user-generated FTP network traffic. We then run the state-of-the-art graybox fuzzer `AFLnwe` [11] for eight hours against the FTP server `ProFTP` [30], which is commonly used for fuzzing evaluations. During this fuzzing campaign, we record the test cases that are deemed interesting by `AFLnwe` as well as the complete network trace. We use the recorded network traffic to replay it to the HMMs and evaluate whether the test cases are deemed interesting by the HMMs. Based on these measurements, we analyze the similarity between the interestingness assessments of the graybox fuzzer `AFLnwe` and the HMMs based on blackbox information (see Section VI).

Third, we implement a blackbox fuzzer that utilizes an HMM for interestingness assessment based on the fuzzing library `LibAFL` [12]. We evaluate the performance of two instances of this fuzzer against the following three baseline fuzzers in terms of code coverage (RQ2 and RQ3).

1) `RANDOM`, which receives random feedback on whether a given test case is interesting or not. Each test case has a probability of 0.5 to be considered interesting.
2) `BLACKBOX`, that receives no information on the interestingness of a test case and considers all test cases as interesting.
3) `AFLnwe`, an existing state-of-the-art graybox fuzzer that utilizes graybox coverage information to decide on the interestingness of a test case.

`RANDOM` and `BLACKBOX` are based on the same implementation as the HMM-based fuzzers, but receive different feedback on the interestingness of the test cases. The approach implemented with `BLACKBOX` is comparable to the current state-of-the-art in blackbox network fuzzing. Based on these baselines, we evaluate how the HMM-based interestingness assessment performs compared to (1) a random assessment, (2) no assessment, and (3) an assessment based on additional graybox information. We measure the performance in terms of final coverage the evaluated fuzzers achieve on the SuT. Our fuzzing experiments adhere to the guidelines of Klees et al. [19].

V. Network Data Preprocessing

The preprocessing of network packets, and especially the hyperparameter choices for this preprocessing, potentially impacts the performance of the downstream model, the HMMs. Thus, we propose, implement, and evaluate a preprocessing pipeline based on two approaches from the literature: `CAPC`, presented by Chiu et al. [28], and `Deep Packet`, presented by Lotfollahi et al. [29]. We conducted experiments to identify satisfying hyperparameters that are presented in detail in [31, Chapter 6]. In this section, we only report the results that we utilize for `Palpebratum`, omitting details on the background, design, and experiments for brevity.

The preprocessing pipeline consists of the following steps. First, the raw packet bytes are padded or cut to a fixed length of d_i bytes, and each byte is normalized to $[0, 1]$. Then, we conduct a dimensionality reduction step, reducing the length of each data point from d_i to d_o. This representation of the network packet is then forwarded to the downstream application. For dimensionality reduction, we consider three different approaches for our experiments: a Principal Component Analysis (PCA), an autoencoder (`AE`), and the convolutional autoencoder `CAPC` presented by Chiu et al. [28].

We analyze the performance of the dimensionality reduction approaches, as well as the impact of hyperparameter choices, based on six network traffic datasets that focus on fuzzing and industrial use cases. The performance is measured by the reconstruction error for test and validation, using a 10-fold cross validation during training.

Our experiments show that `CAPC` generalizes better in an in-domain evaluation, meaning training and testing on the same dataset. In contrast, `AE` leads to a better out-of-domain generalization. For `Palpebratum`, we train the preprocessing module on user-generated network traffic but use it on fuzzing data, requiring out-of-domain generalization. Thus, we expect `AE` to be better suited for the task given by `Palpebratum`.

Moreover, within our experiments, the PCA results in higher reconstruction errors than the other two approaches.

For `Palpebratum`, we choose the hyperparameters that lead to the smallest reconstruction errors in our experiments. Namely, we fix the seeds used for the dimensionality reduction approaches, as well as the general seed for the randomness, and set $d_i = 304$ and $d_o = 24$.

VI. HMM Behavior Approximation

To address RQ1, we evaluate the performance of the HMM-based interestingness assessment in comparison to the graybox assessment conducted by `AFLnwe`. Specifically, we investigate how the size of the HMMs and dimensionality reduction approach used for network data preprocessing (see Section V) impact the results. To this end, we consider HMMs with varying numbers of nodes and preprocess the network traffic using two approaches: `AE` and `CAPC`.

A. Hidden Markov Models

Based on the following reasoning, we choose first-order HMMs with 7, 18, 27, 38, and 51 nodes, each including one start and one end node. While choosing a different approach to train and use the resulting models, Gascon et al. train a second-order Markov chain on FTP data, resulting in a model with 6 nodes [32]. Thus, we decided to include an HMM with $6 \cdot 6 + 2 = 38$ nodes, transforming the second order model to a first order one (see Section II-B). Moreover, to analyze the impact of adding or deleting one node from the Markov chain, we include HMMs with $7 \cdot 7 + 2 = 51$ and $5 \cdot 5 + 2 = 27$ nodes respectively. Furthermore, we include HMMs with 18 and 7 nodes, as these models showed to be most suited to represent user-generated FTP network traffic. For this analysis, we trained HMMs with a varying number of nodes on the user-generated FTP traffic for several different FTP server implementations. Then, we assessed the performance of the HMMs using the Bayesian Information Criterion [33].

B. Experimental Setting

We conduct the following steps for our experiments.

1) Train the HMMs on a network traffic capture that includes 137 sequences of user-generated FTP traffic.
2) Execute `AFLnwe` against `ProFTP` for eight hours, recording all network traffic generated by the fuzzer and the SuT. This results in 9 GB of network traffic, and 672,214 test cases generated by `AFLnwe`.
3) Gather the test cases that are considered interesting by `AFLnwe` from the final corpus.
4) Replay the recorded network traffic to the HMMs and identify which test cases are deemed interesting by the different HMMs.
5) Compare the interestingness assessment of `AFLnwe` and the HMMs.

We compare the interestingness assessment of `AFLnwe` and the HMMs by analyzing how many test cases were deemed interesting by both approaches, and at which point of time during the fuzzing campaign the interesting test cases are found. With this, we aim to discover new insights on how the HMMs and `AFLnwe` represent and assess the behavior of the SuT during testing.

TABLE I
COMPARISON OF INTERESTINGNESS DECISIONS TAKEN BY HMMS WITH A VARYING NUMBER OF NODES (N) AND AFLNWE, SHOWING THE TRUE POSITIVES (TP), FALSE POSITIVES (FP), AND THE PRECISION (PR).

N	PCA			AE			CAPC		
	TP	FP	Pr	TP	FP	Pr	TP	FP	Pr
7	1	9	0.10	1	15	0.06	2	9	0.18
18	2	6	0.25	2	40	0.05	2	39	0.05
27	2	6	0.25	2	65	0.03	1	7	0.12
38	1	8	0.11	4	45	0.08	3	56	0.05
51	2	20	0.09	5	116	0.04	5	109	0.04

C. Results

We analyze the test cases deemed interesting by `AFLnwe` and the HMMs and present the corresponding metrics in Table I. For each HMM, we report the number of test cases that were considered interesting by both the HMM and `AFLnwe` (True Positives (TP)), the number of test cases that were considered interesting only by the HMM (False Positives (FP)), and the precision (Pr). N denotes the number of nodes of the respective HMM. The HMMs with the highest precision and highest TP ratio are highlighted for each dimensionality reduction approach. For example, among the test cases deemed interesting by the HMM with 51 nodes using `CAPC` for dimensionality reduction, five are also considered interesting by `AFLnwe`, while 109 are not.

`AFLnwe` considers 3,220 out of 672,214 test cases to be interesting in total, while out of the HMMs, `AE`/51 considers the most test cases interesting, with a total number of 121 (see Table I). Furthermore, it shows that HMMs with more nodes tend to consider more test cases interesting compared to the HMMs with fewer nodes. In addition, our experiments show that the HMMs using `PCA` for preprocessing tend to identify less test cases interesting.

In addition, we report the distribution of test cases deemed interesting by a selection of HMMs over the total sequence of test cases generated during the fuzzing campaign in Fig. 3. Note that the figure uses a logarithmic scale for the x-axis. Test cases that are deemed interesting by both the HMM and `AFLnwe` are shown as blue circles. The boxes span from the lower quartile to the upper quartile, and the whiskers show the farthest point within the 1.5 interquartile range. The median is shown as vertical line in the box.

Fig. 3 reveals that (1) the majority of test cases deemed interesting by the HMMs are executed at the beginning of the fuzzing campaign, (2) `AFLnwe` considers test cases interesting throughout the entire campaign, and (3) most test cases deemed interesting by both the HMMs and `AFLnwe` are located within the first 1,000 test cases. One reason for this weight on the first few test cases could be an early full coverage of the HMMs. If the HMMs are fully covered, no new paths can be covered, and thus no more test cases are

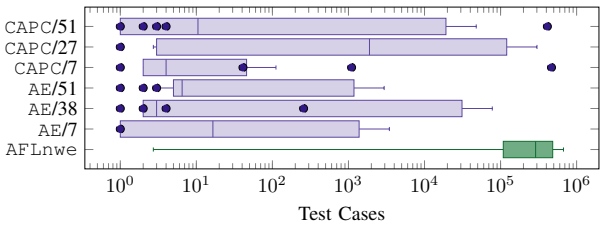

Fig. 3. Distribution of test cases that are considered to be interesting by the HMMs and AFLnwe. Test cases that are deemed interesting by both an HMM and AFLnwe are shown as circles. Most test cases deemed interesting by both the HMMs and AFLnwe are located in the first 1,000 test cases.

deemed interesting. However, analyzing the relative coverage of the HMMs over the course of the fuzzing campaign shows no full coverage (see Fig. 4). For example, CAPC/51 reaches a final HMM coverage of 6 %, and AE/18 reaches 19 %.

D. Discussion

To interpret the results presented above, two insights need to be considered. First, AFLnwe has a more fine-grained view on the SuT's behavior. AFLnwe considers the number of basic blocks to analyze the interestingness of a test case, while the HMMs use the transitions of their hidden states. The number of basic blocks is much higher than the number of transitions in the HMMs. For ProFTP, for example, AFLnwe considers 27,392 basic blocks, while an HMM with 51 nodes considers 2,499 transitions. Thus, AFLnwe can identify smaller changes in the SuT's behavior, and it is expected that AFLnwe deems more test cases interesting than the HMMs.

Second, as AFLnwe is based on AFL, it considers test cases as interesting if they lead to new coverage [8]. Consequently, the interestingness of a test case depends on the sequence of test cases that was sent and analyzed before, and on how the behavior of the SuT was assessed with respect to these test cases. Since the HMMs are expected to represent the behavior of the SuT differently than the code coverage representation considered by AFLnwe, differences in the number of interesting test cases are to be expected.

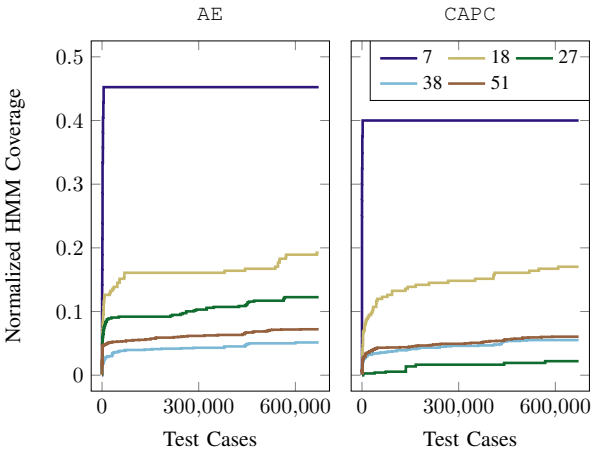

Fig. 4. Coverage of the HMMs with varying numbers of nodes achieved during fuzzing, using either CAPC or AE for dimensionality reduction. The values are normalized to the interval $[0, 1]$, where a value of 1 would represent 100 % coverage. None of the HMMs reaches full coverage.

Supporting the first expectation, our experiments show that the HMMs tend to identify fewer test cases as interesting, compared to AFLnwe. Additionally, HMMs with more nodes tend to consider more test cases interesting compared to HMMs with fewer nodes.

Furthermore, our results show that the HMMs using PCA for dimensionality reduction consider less test cases interesting and, consequently, achieve less HMM coverage during fuzzing. This suggests that the PCA-based representation of the network traffic is less useful for the HMMs. This finding aligns with our observation that PCA exhibits higher reconstruction errors compared to CAPC and AE (see Section V).

Moreover, our experiments show that the HMMs with more nodes tend to achieve higher numbers of true positives, but still result in low precision. This indicates that the HMMs do not generate a representation of the SuT that is comparable to the one generated by AFLnwe. This supports our second expectation. Nevertheless, the HMM's representation might be beneficial for improving blackbox fuzzing, which we evaluate in Section VII.

Our experiments regarding the distribution of interesting test cases over time reveal that the HMMs tend to consider test cases at the beginning of the campaign interesting, while not reaching full HMM coverage. This implies that large parts of the HMMs are not necessary to represent the fuzzing network traffic. These results suggest that training the HMMs more specific to the fuzzing use case might improve the granularity of their representation of the SuT's behavior.

VII. HMM-based Fuzzing

Based on the insights presented in the previous chapters, we evaluate the performance of the entire approach by implementing an HMM-based blackbox fuzzer and assessing its performance measured in code coverage. We implement Palpebratum based on the graybox fuzzing library LibAFL [12], allowing us to reuse strategies from mutational coverage-guided graybox fuzzing. Within LibAFL, we replace the parts requiring graybox knowledge with the HMM-based interestingness assessment newly presented in this work. Additionally, we implement the two baseline fuzzers described in Section IV, RANDOM and BLACKBOX, using the same modules and strategies. Thus, the performances of Palpebratum and the two baseline fuzzers are directly comparable.

We integrate these newly implemented fuzzers to the fuzzing benchmark ProFuzzBench [34]. This benchmark enables automated execution of a fuzzer and a SuT, as well as automated coverage analysis. Note that Palpebratum is a full blackbox fuzzer, but our experimental setting and the SuTs are graybox, allowing us to measure the code coverage achieved by the considered fuzzers.

A. Coverage Measurement

ProFuzzBench expects a fuzzer's corpus to include all test cases that lead to new coverage at the end of the fuzzing campaign, as these are the test cases a graybox fuzzer finds interesting and adds to the corpus. As a result, ProFuzzBench only calculates the coverage of the test cases that were collected in the corpus to determine the final code coverage. This approach is sound for graybox fuzzing, as a graybox fuzzer has full access to the coverage information during fuzzing. In blackbox fuzzing however, the corpus consists of those test cases for which the fuzzer expects an increasing coverage based on its approximation. However, this approximation may be inaccurate. Thus, we need to replay *all* test cases generated by a fuzzer to actually calculate the full coverage of a blackbox fuzzer. We enhance the implementations of ProFuzzBench and LibAFL to allow for this refined coverage measurement.

B. Experimental Setting

We run 30 trials per fuzzer, and run each trial for 24 hours, following the guidelines suggested by Klees et al. [19]. As we base our experiments on ProFuzzBench (commit 8573ec8), we run the fuzzers against one of the targets provided by ProFuzzBench, ProFTP, and use the provided seed files. We measure the code coverage of *all* test cases generated by the fuzzers using the coverage measurement provided by ProFuzzBench, which is based on llvm-cov [35]. We conduct all experiments on a server equipped with an AMD Ryzen Threadripper PRO 5975WX CPU with 32 cores and 130 GB of RAM, and running Ubuntu 23.10.

C. Results

This section presents the results of the previously described experiments, while Section VII-D discusses them. We first report the coverage in basic block hits achieved by the two instances of Palpebratum and the two baseline fuzzers in Fig. 5. This coverage is calculated over all test cases generated by the fuzzers throughout the fuzzing campaign. The figure presents the mean of 30 runs for each fuzzer as well as the 95 % confidence interval. It shows that $\texttt{Palpebratum}_{\texttt{AE}}$ and BLACKBOX outperform $\texttt{Palpebratum}_{\texttt{CAPC}}$ and RANDOM. Moreover, our experiments show that BLACKBOX achieves higher coverage earlier in the campaign but remains mostly constant after approximately 12 hours.

Table II presents the final coverage achieved by the fuzzers as well as the number of test cases each fuzzer generated over the course of the 24 hours. All numbers represent the mean of the 30 runs we conducted for each fuzzer. We report the absolute number of basic blocks each fuzzer covered at the end of the campaign, as well as the relative number, showing the relation to the total number of basic blocks in ProFTP. For example, BLACKBOX covers 4,374.23 basic blocks, corresponding to 15.89 % coverage, while $\texttt{Palpebratum}_{\texttt{AE}}$ covers 4,379.33 basic blocks or 15.91 % of ProFTP. In particular, the number of generated test cases highlights differences between the baseline fuzzers and the two instances of Palpebratum, with the baseline fuzzers being able to generate far more test cases during the 24 hours. For example, BLACKBOX generates 15,433.27 test cases, while $\texttt{Palpebratum}_{\texttt{AE}}$ only generates 7,171.10 test cases.

TABLE II
FINAL COVERAGE OF PALPEBRATUM AND THE BASELINES FUZZERS (TOTAL NUMBER OF BASIC BLOCKS AND RELATIVE COVERAGE) WITH TOTAL NUMBER OF TEST CASES (MEAN OF 30 RUNS).

Fuzzer	Final Coverage	# Test Cases
BLACKBOX	4,374.23 (15.89%)	15,433.27
RANDOM	4,307.60 (15.65%)	11,155.30
$\texttt{Palpebratum}_{\texttt{AE}}$	4,379.33 (15.91%)	7,456.93
$\texttt{Palpebratum}_{\texttt{CAPC}}$	4,309.80 (15.66%)	7,171.10
AFLnwe	5,227.40 (18.99%)	-

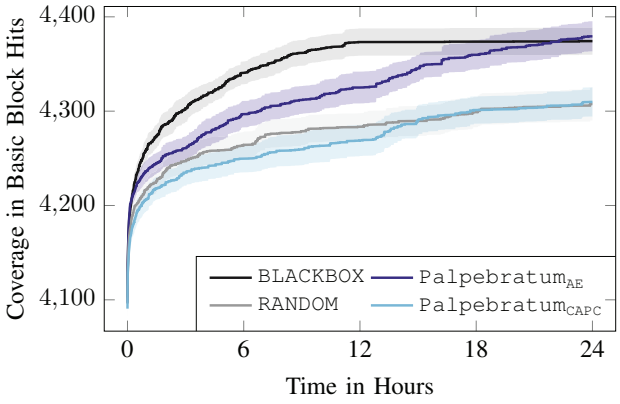

Fig. 5. Coverage in basic blocks for the four fuzzers over the course of 24 hours. Each line represents the mean of 30 runs while the shaded area represents the 95 % confidence interval. $\texttt{Palpebratum}_{\texttt{AE}}$ and BLACKBOX outperform $\texttt{Palpebratum}_{\texttt{CAPC}}$ and RANDOM.

AFLnwe reaches a mean final coverage of 5,227.4 basic blocks. Note that Fig. 5 excludes the results for AFLnwe for visualization purposes. AFLnwe does not report the total number of generated test cases, and thus this number is not included in Table II. Nevertheless, we can report the number of test cases that AFLnwe considers interesting, which results in a mean of 3,074.2 test cases.

To allow for a statistical assessment of the fuzzers' performance, we present the results of a two-sided Mann-Whitney U Test [36] with a significance level of $\alpha = 0.05$. Table III presents the corresponding p-values. For each pair of fuzzers, the null hypothesis, claiming that the two fuzzers achieve the same coverage, is rejected if the p-value is less than α. In correspondence to the visual representation in Fig. 5, our statistical analysis shows that this null hypothesis cannot be rejected for $\texttt{Palpebratum}_{\texttt{AE}}$ and BLACKBOX, and $\texttt{Palpebratum}_{\texttt{CAPC}}$ and RANDOM, respectively. The corresponding cells in Table III are highlighted.

D. Discussion

Our experiments regarding the performance of the HMM-based fuzzers provide the following key insights.

TABLE III
RESULTING P-VALUES OF A TWO-SIDED MANN-WHITNEY U TEST. THE NAMES OF THE TWO INSTANCES OF PALPEBRATUM ARE ABBREVIATED.

	BLACKBOX	RANDOM	Pal_{AE}	Pal_{CAPC}
BLACKBOX	-	<0.001	0.824	<0.001
RANDOM	<0.001	-	<0.001	0.912
Pal_{AE}	0.824	<0.001	-	<0.001
Pal_{CAPC}	<0.001	0.912	<0.001	-

Impact of HMM: Our results suggest that the choice of the underlying models, especially the dimensionality reduction approach, significantly influences a fuzzer's performance. $\texttt{Palpebratum}_{\texttt{AE}}$, based on an HMM with 51 nodes using AE for dimensionality reduction, achieves significantly higher coverage than $\texttt{Palpebratum}_{\texttt{CAPC}}$, which also uses an HMM with 51 nodes but utilizes CAPC for dimensionality reduction. This aligns with the experiments conducted on the performance of the dimensionality reduction approaches (Section V). These experiments suggest that AE has a higher performance with respect to out-of-domain generalization. However, our experiments with respect to the performance of the HMMs presented in Section VI showed no difference between the two considered HMMs. This again stresses that an evaluation of the full pipeline is necessary to assess the performance of the different building blocks in conjunction.

Efficiency: In line with Gopinath et al. [37], we define the efficiency of a fuzzer as the number of test cases that are generated and executed within a fixed time frame. Our experiments show that the two baseline fuzzers generate more test cases over the course of 24 hours, compared to the HMM-based fuzzers (see Table II). The HMM-based fuzzers include the overhead of processing the network traffic and querying the HMM to determine the interestingness of each test case. In contrast, for BLACKBOX, this decision is trivial as each test case is considered to be interesting, and for RANDOM this decision is based on one query to a random oracle.

We exclude the HMMs' training time from our efficiency considerations because it is negligible compared to the duration of the fuzzing campaign. Training the HMMs used for $\texttt{Palpebratum}_{\texttt{CAPC}}$ and $\texttt{Palpebratum}_{\texttt{AE}}$ required approximately 7 seconds and 15 seconds, respectively.

Relative Achieved Coverage: In our experiments, $\texttt{Palpebratum}_{\texttt{AE}}$ and BLACKBOX significantly outperform $\texttt{Palpebratum}_{\texttt{CAPC}}$ and RANDOM in terms of coverage achieved by all test cases generated over the course of 24 hours. Combined with the results regarding the efficiency of the fuzzers discussed above, this suggests that the current implementation of the HMMs introduces too much overhead, which outweighs the additional information they provide. Nevertheless, the performance of $\texttt{Palpebratum}_{\texttt{AE}}$ highlights the potential of Palpebratum. Despite generating only about half the number of test cases as BLACKBOX, $\texttt{Palpebratum}_{\texttt{AE}}$ achieved comparable coverage. Moreover, the coverage achieved by $\texttt{Palpebratum}_{\texttt{AE}}$ is still increasing at the end of the campaign, while the coverage achieved by BLACKBOX seems to stagnate.

This suggests that the HMM-based approaches could surpass the baselines if implemented more efficiently. Currently, the HMMs are implemented based on the Python library pomegranate v0.6.0 [38], while the fuzzer itself is implemented with LibAFL, which uses Rust. Implementing the HMMs directly in Rust could potentially reduce the overhead and improve performance.

As expected, our experiments show that the graybox fuzzer AFLnwe significantly outperforms all blackbox fuzzers.

VIII. DISCUSSION

This section consolidates the results of the various experiments and presents insights related to the research questions outlined in Section I. Additionally, it discusses the broader implications of our findings and acknowledges the limitations of our study.

A. Research Questions

Our work addresses three main research questions that concern the performance of the used HMMs as well as the fuzzers utilizing these HMMs for blackbox fuzzing.

RQ1 *How does the behavior approximation of the HMMs compare to that of the graybox fuzzer AFLnwe?*

Our experiments lead to the following main outcomes. First, the HMMs consider fewer test cases interesting compared to AFLnwe. Second, HMMs with more nodes tend to lead to more interesting test cases than those with fewer nodes. Both outcomes are in line with the general insight that a model with more nodes can represent the considered information with a higher granularity. Thus, models capable of representing smaller changes in the behavior of the SuT are expected to detect more of these changes, and consequently, consider more test cases interesting. As AFLnwe considers the graybox coverage measured in basic blocks, it represents the SuT's behavior with the highest granularity.

Furthermore, comparing the specific test cases that are deemed interesting by the HMMs and AFLnwe reveals significant differences. For example, AE/51 deemed 121 test cases interesting, of which only 5 were also deemed interesting by AFLnwe. Nevertheless, this difference does not necessarily imply that the HMMs are unsuited to improve the performance of blackbox testing, as they might still represent information relevant for testing.

RQ2 *Does the HMM-based behavior approximation improve blackbox fuzzing in terms of code coverage?*

Our experiments show mixed results with respect to this research question. $\texttt{Palpebratum}_{\texttt{AE}}$ significantly outperforms the random baseline fuzzer RANDOM in terms of achieved coverage, but does not outperform the blackbox baseline BLACKBOX. Recall that BLACKBOX considers all test cases interesting while RANDOM considers random test cases interesting. Thus, as $\texttt{Palpebratum}_{\texttt{AE}}$ outperforms RANDOM, it shows that the interestingness approximation provided by the HMM leads to better performance than a random approximation. Moreover, $\texttt{Palpebratum}_{\texttt{AE}}$ generates fewer test

cases than the two baseline fuzzers, due to the overhead the HMM introduces, but still leads to a better performance than RANDOM.

However, as $\texttt{Palpebratum}_{\texttt{AE}}$ does not outperform BLACKBOX, it also shows that the approximation of the HMM does not lead to a better performance than just classifying each test case as interesting for the fixed time frame of 24 hours. Nevertheless, $\texttt{Palpebratum}_{\texttt{AE}}$ reaches a comparable coverage to BLACKBOX with only half the number of test cases. Improving the overhead the HMMs introduce to the test case generation might also improve the performance of the HMM-based fuzzers.

Moreover, it shows that $\texttt{Palpebratum}_{\texttt{CAPC}}$ neither outperforms the RANDOM nor BLACKBOX. Thus, in contrast to $\texttt{Palpebratum}_{\texttt{AE}}$, for $\texttt{Palpebratum}_{\texttt{CAPC}}$, the overhead of the HMM outweighs the improvement with respect to the interestingness approximation.

Furthermore, our experiments demonstrate that BLACKBOX significantly outperforms RANDOM. As RANDOM can be considered as a fuzzer using an unsuitable model for the interestingness approximation, this stresses that choosing a blackbox approach might still yield better results than using a model-based approach that uses an unsuitable model.

RQ3 *How does the network data preprocessing impact the performance of both the HMMs and the fuzzers?*

Our experiments suggest that the dimensionality reduction approaches impact the performance of the HMMs as well as the performance of the subsequent fuzzers.

Regarding the performance of the HMMs, our experiments demonstrate that the HMMs based on PCA tend to deem fewer test cases interesting compared to preprocessing based on AE and CAPC. The HMMs based on AE and CAPC achieve comparable results.

In contrast, the fuzzers based on HMMs using AE and CAPC, $\texttt{Palpebratum}_{\texttt{AE}}$ and $\texttt{Palpebratum}_{\texttt{CAPC}}$, show significant differences in performance. Specifically, $\texttt{Palpebratum}_{\texttt{AE}}$ achieves significantly more coverage on average than $\texttt{Palpebratum}_{\texttt{CAPC}}$ over a 24 hour fuzzing campaign. This observation suggests that the choice of dimensionality reduction approach significantly impacts the downstream fuzzer performance.

B. Limitations

We base our experiments on ProFTP as SuT, since FTP is a network protocol commonly used in evaluations regarding network fuzzing and is widely utilized in industrial use cases. Nevertheless, testing Palpebratum against more SuTs and especially including actual OT components would yield insights that could be considered to be more generalizable. Especially targeting OT components would provide deeper insights into the balance between the overhead introduced by the HMMs and the benefits they bring to blackbox testing. As testing OT components introduces additional overhead to fuzzing, such as sending inputs via the network and monitoring the OT component's services, the overhead in test case generation introduced by the HMMs might be less noticeable.

IX. Related Work

As Palpebratum is located in the domain of industrial blackbox fuzzing and utilizes HMMs, we present related work in these two domains. Pulsar, as presented by Gascon et al. [32], is especially closely related to Palpebratum. Therefore, we detail this approach in Section IX-C.

To the best of our knowledge, no previous work uses an HMM or a different modeling approach to model the network traffic generated during testing to improve the interestingness assessment in blackbox fuzzing.

A. Blackbox Fuzzing

Fuzzing is a testing technique that is increasingly applied to the industrial domain. While Lan and Sun and Wei et al. both present overviews of industrial fuzzing [39, 40], the following description focuses on approaches related to Palpebratum.

In literature, several approaches utilize blackbox information to derive a state machine of the SuT. The work presented by Doupé et al. and Drakonakis et al. is located in the domain of web applications and infers a state machine from the HTTP responses received from the SuT [41, 42]. Aichernig et al. utilize the responses received via MQTT, a protocol widely used within the Internet of Things, to infer a state machine. In these works, the resulting state machine is directly used to guide the fuzzing process by suggesting the states of the SuT that should be tested next. In contrast, Palpebratum uses the derived HMM to assess the interestingness of a test case.

Other approaches also utilize blackbox information to assess the interestingness of a test case. Kim et al. leverage the response of the SuT received via the industrial network protocol Modbus and the corresponding response time to assess the interestingness [44]. ICPFuzzer and R0fuzz both utilize the response codes sent by the SuT for interestingness assessment [45, 46]. In the corresponding case studies, ICPFuzzer focuses on the network protocol DLMS/COSEM, while R0fuzz considers Modbus and DNP3. In contrast to these works, Palpebratum processes and utilizes the whole network traffic in a protocol-agnostic way. For our experiments, we focus on FTP.

B. Markov Chains and HMMs

Several approaches from literature utilize Markov chains and HMMs to model different aspects of network traffic. Wright et al. use a Markov chain to model and generate user events that lead to network traffic, which is then used to test network applications [47]. Dainotti et al. propose to use an HMM to model network packet behavior from internet traffic sources [48]. More specifically, they model the inter-packet time and the packet size with the HMM.

Moreover, several publications use Markov chains and HMMs to model different aspects of blackbox and graybox fuzzing (see e.g. [49] for an overview). Böhme et al. and Rawat et al. present approaches that apply a Markov chain to model the seed selection problem in graybox fuzzing [50, 51], while Salem and Song use a Markov chain for blackbox test case generation [52].

C. Pulsar

Gascon et al. present Pulsar, an approach providing a generative model of a network protocol client which is used to analyze and simulate network traffic to support a blackbox network fuzzer [32]. More specifically, Pulsar builds a second-order Markov chain representing the state machine of the protocol under test, as well as a set of templates and rules that define the specifics of a network packet and the dependencies between network packets. The Markov chain is then transformed into a deterministic finite automaton. This automaton is used to generate fuzzing responses to requests received by the SuT, leveraging so-called *fuzzing masks* which indicate which parts of the network packet should be fuzzed next. Gascon et al. evaluate Pulsar using FTP and the proprietary network protocol OSCAR.

While Pulsar, similarly to Palpebratum, utilizes HMMs to approximate the network traffic before and during fuzzing, the two approaches differ. Pulsar models the network protocol in detail to generate new packets as responses to a SuT's request. In contrast, Palpebratum learns an approximation of the SuT's behavior in order to identify differences in behavior, which are then used to assess the interestingness of a test case. Furthermore, while Pulsar provides a full fuzzing framework based on the HMM, Palpebratum offers a substitution for the interestingness assessment in graybox fuzzing. This allows Palpebratum to be used by arbitrary graybox approaches that are based on an interestingness assessment, allowing for an easy adoption of new advances in graybox fuzzing. Additionally, due to the different modeling targets, Pulsar requires more initial training data than Palpebratum. For example, for the use case of FTP, Pulsar used network traffic including 987 traces, while Palpebratum used only 137 traces. In summary, while both approaches utilize HMMs to model certain aspects of blackbox network fuzzing, they are applicable to different use cases and follow different methodologies.

X. Future Work

Future work building upon the insights provided by Palpebratum could include improving the performance of the HMMs and extending the presented experiments.

The implementation used for this work utilizes the fuzzing framework LibAFL, written in Rust, and the HMM library pomegranate, written in Python. This setup requires inefficient communication between these two programming languages. Thus, one approach to improving performance would be to implement the HMMs in Rust as well.

Another direction of research could encompass analyzing the transferability of the trained HMMs. For this, HMMs trained on network traffic generated by different FTP implementations could be compared with respect to their similarity. Moreover, the in-domain and out-of-domain performance of fuzzers based on these HMMs could be analyzed. To omit the requirement for an implementation of the considered network protocol, HMMs generated by Pulsar [32] could be integrated in Palpebratum.

In addition, other modeling approaches such as Bayesian networks [53] or transformer models [54] could be explored.

Furthermore, the experiments presented in this work could be extended by including more SuTs. With this, it could be investigated to which extent the insights provided by testing ProFTP can be transferred to other SuTs. Moreover, one could include OT components as SuTs to analyze how the balance between the additional overhead introduced by the HMMs and the additional information they provide strikes in this case. As testing OT components introduces additional overhead that is independent of the test case generation, such as sending the test cases via the network and monitoring the services of the SuTs, this balance might differ from the one observed with local FTP implementations.

XI. Conclusions

We present Palpebratum, an approach to approximate a SuT's behavior to enable the application of mutational coverage-based *graybox* fuzzing in a *blackbox* test setting. This facilitates the application of established graybox testing methodologies to the domain of OT component testing, which requires blackbox testing.

We implement and evaluate the full pipeline starting from raw network traffic generated during a blackbox fuzzing test. The network traffic is preprocessed and presented to an HMM that approximates the SuT's behavior that led to the observed traffic. This approximation is then provided to a blackbox fuzzer applying a graybox fuzzing approach. The fuzzer uses this approximation as a surrogate for graybox coverage information which is unavailable in a blackbox test setting.

Our evaluation shows that Palpebratum significantly outperforms one of the two blackbox baselines, RANDOM, in terms of achieved coverage for one of the two considered HMMs. Palpebratum performs comparably to the other blackbox baseline, BLACKBOX, while only generating half the number of test cases in the given timeframe of 24 hours. This suggests reducing the overhead introduced by the HMMs might further improve Palpebratum's performance compared to the baselines.

In summary, our work demonstrates that approximating the SuT's behavior using an HMM has the potential to generate more effective test cases compared to a full blackbox fuzzer when executed against an FTP implementation. This provides a starting point for further research, including reducing the overhead introduced by the HMMs, and conducting additional experiments to analyze the balance between the introduced overhead and the performance improvement, especially for finding vulnerabilities in OT components.

Acknowledgments

This work was partly funded by the Topic *Engineering Secure Systems* of the Helmholtz Association (HGF) and supported by KASTEL Security Research Labs, Karlsruhe.

References

[1] M. Baezner and P. Robin, "Stuxnet," ETH Zurich, Tech. Rep., 2017.

[2] A. DiPinto, Y. Dragoni, and A. Carcano, "TRITON: The first ICS cyber attack on safety instrument systems," *Black Hat USA*, vol. 2018, pp. 1–26, 2018.

[3] S. D. D. Anton, D. Fraunholz, D. Krohmer, D. Reti, D. Schneider, and H. D. Schotten, "The global state of security in industrial control systems: An empirical analysis of vulnerabilities around the world," *IEEE Internet of Things Journal*, vol. 8, no. 24, pp. 17 525–17 540, 2021.

[4] M. Kohl and S. Oberman, "Ripple 20 – CVE-2020-11896 RCE CVE-2020-11898 Info Leak," JSOF Research Lab, Tech. Rep., 2020. [Online]. Available: https://www.jsof-tech.com/wp-content/uploads/2020/06/JSOF_Ripple20_Technical_Whitepaper_June20.pdf

[5] M. Kohl, A. Schön, and S. Oberman, "Ripple 20 – CVE-2020-11901," JSOF Research Lab, Tech. Rep., 2020. [Online]. Available: https://www.jsof-tech.com/wp-content/uploads/2020/08/Ripple20_CVE-2020-11901-August20.pdf

[6] International Electrotechnical Commission, "IEC 62443 Security for Industrial Automation and Control Systems," International Electrotechnical Commission, Geneva, CH, Standard, 2019.

[7] V. Manès, H. Han, C. Han, S. K. Cha, M. Egele, E. Schwartz, and M. Woo, "The art, science, and engineering of fuzzing: A survey," *IEEE Transactions on Software Engineering*, vol. 47, no. 11, pp. 2312–2331, 2019.

[8] M. Zalewski, "American fuzzy lop," Tech. Rep., 2016.

[9] A. Fioraldi, D. Maier, H. Eißfeldt, and M. Heuse, "AFL++: Combining incremental steps of fuzzing research," in *14th USENIX Workshop on Offensive Technologies (WOOT 20)*, 2020.

[10] V.-T. Pham, M. Böhme, and A. Roychoudhury, "AFLNet: a greybox fuzzer for network protocols," in *13th International Conference on Software Testing, Validation and Verification (ICST)*. IEEE, 2020, pp. 460–465.

[11] V.-T. Pham, M. Böhme, and A. Roychoudhury. (2020) AFLnwe. [Online]. Available: https://github.com/thuanpv/aflnwe

[12] A. Fioraldi, D. C. Maier, D. Zhang, and D. Balzarotti, "Libafl: A framework to build modular and reusable fuzzers," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2022, pp. 1051–1065.

[13] C. Doumanidis, Y. Xie, P. Rajput, R. Pickren, B. Sahin, S. Zonouz, and M. Maniatakos, "Dissecting the industrial control systems software supply chain," *IEEE Security & Privacy*, 2023.

[14] D. Golani, R. Fricke, and B. Appelbaum-Golani, "Review of the genus photoblepharon (actinopterygii: Beryciformes: Anomalopidae)," *Acta Ichthyologica et Piscatoria*, vol. 49, no. 1, pp. 33–41, 2019.

[15] B. Miller, M. Zhang, and E. R. Heymann, "The relevance of classic fuzz testing: Have we solved this one?" *IEEE Transactions on Software Engineering*, vol. 48, no. 6, pp. 2028–2039, 2020.

[16] D. dos Santos, S. Dashevskyi, J. Wetzels, and A. Amri, "Amnesia:33 – How TCP/IP stacks breed critical vulnerabilities in IoT, OT and IT devices," Forescout Research Labs, Tech. Rep., 2021.

[17] S. Mallissery and Y.-S. Wu, "Demystify the fuzzing methods: A comprehensive survey," *ACM Computing Surveys*, vol. 56, no. 3, pp. 1–38, 2023.

[18] M. Böhme, C. Cadar, and A. Roychoudhury, "Fuzzing: Challenges and reflections," *IEEE Software*, vol. 38, no. 3, pp. 79–86, 2021.

[19] G. Klees, A. Ruef, B. Cooper, S. Wei, and M. Hicks, "Evaluating fuzz testing," in *2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 2123–2138.

[20] D. Liyanage, S. Lee, C. Tantithamthavorn, and M. Böhme, "Extrapolating coverage rate in greybox fuzzing," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, 2024, pp. 1–12.

[21] S. Österlund, K. Razavi, H. Bos, and C. Giuffrida, "ParmeSan: Sanitizer-guided greybox fuzzing," in *29th USENIX Security Symposium*, 2020, pp. 2289–2306.

[22] M. Muench, J. Stijohann, F. Kargl, A. Francillon, and D. Balzarotti, "What you corrupt is not what you crash: Challenges in fuzzing embedded devices," in *Network and Distributed System Security (NDSS) Symposium*, 2018.

[23] L. E. Sucar, "Probabilistic graphical models," *Advances in Computer Vision and Pattern Recognition*, vol. 10, no. 978, 2015.

[24] L. Rabiner and B. Juang, "An introduction to hidden markov models," *IEEE Acoustics, Speech, and Signal Processing (ASSP) Magazine*, vol. 3, no. 1, pp. 4–16, 1986.

[25] Y. Liu, J. Park, K. Dahmen, Y. Chemla, and T. Ha, "A comparative study of multivariate and univariate hidden markov modelings in time-binned single-molecule fret data analysis," *The Journal of Physical Chemistry B*, vol. 114, no. 16, pp. 5386–5403, 2010.

[26] J. F. Mari, D. Fohr, and J.-C. Junqua, "A second-order HMM for high performance word and phoneme-based continuous speech recognition," in *International Conference on Acoustics, Speech, and Signal Processing Conference Proceedings*, vol. 1. IEEE, 1996, pp. 435–438.

[27] S. Russell and P. Norvig, *Artificial intelligence: A modern approach*. Pearson, 2016.

[28] K.-C. Chiu, C.-C. Liu, and L.-D. Chou, "CAPC: packet-based network service classifier with convolutional autoencoder," *IEEE Access*, vol. 8, pp. 218 081–218 094, 2020.

[29] M. Lotfollahi, M. Jafari Siavoshani, R. Shirali Hos-

sein Zade, and M. Saberian, "Deep packet: A novel approach for encrypted traffic classification using deep learning," *Soft Computing*, vol. 24, no. 3, pp. 1999–2012, 2020.

[30] The ProFTPD Project. ProFTPD – Highly configurable GPL-licensed FTP server software . [Online]. Available: http://www.proftpd.org/

[31] A. Borcherding, "Use of accessible information to improve industrial security testing," Ph.D. dissertation, Karlsruhe Institute of Technology, 2024.

[32] H. Gascon, C. Wressnegger, F. Yamaguchi, D. Arp, and K. Rieck, "Pulsar: Stateful black-box fuzzing of proprietary network protocols," in *Security and Privacy in Communication Networks: 11th EAI International Conference, SecureComm 2015*. Springer, 2015, pp. 330–347.

[33] A. A. Neath and J. E. Cavanaugh, "The bayesian information criterion: background, derivation, and applications," *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 4, no. 2, pp. 199–203, 2012.

[34] R. Natella and V.-T. Pham, "Profuzzbench: A benchmark for stateful protocol fuzzing," in *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2021, pp. 662–665.

[35] LLVM Project. llvm-cov – emit coverage information. [Online]. Available: https://llvm.org/docs/CommandGuide/llvm-cov.html

[36] H. Mann and D. Whitney, "On a test of whether one of two random variables is stochastically larger than the other," *The annals of mathematical statistics*, pp. 50–60, 1947.

[37] R. Gopinath, P. Görz, and A. Groce, "Mutation analysis: Answering the fuzzing challenge," *arXiv preprint arXiv:2201.11303*, 2022.

[38] J. Schreiber. pomegranate. [Online]. Available: https://github.com/jmschrei/pomegranate/tree/master

[39] H. Lan and Y. Sun, "Review on fuzz testing for protocols in industrial control systems," in *Sixth International Conference on Data Science in Cyberspace (DSC)*. IEEE, 2021, pp. 433–438.

[40] X. Wei, Z. Yan, and X. Liang, "A survey on fuzz testing technologies for industrial control protocols," *Journal of Network and Computer Applications*, p. 104020, 2024.

[41] A. Doupé, L. Cavedon, C. Kruegel, and G. Vigna, "Enemy of the state: A state-aware black-box web vulnerability scanner," in *21st USENIX Security Symposium*, 2012, pp. 523–538.

[42] K. Drakonakis, S. Ioannidis, and J. Polakis, "Rescan: A middleware framework for realistic and robust black-box web application scanning," in *Network and Distributed System Security (NDSS) Symposium*, 2023.

[43] B. Aichernig, E. Muškardin, and A. Pferscher, "Learning-based fuzzing of IoT message brokers," in *14th Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 2021, pp. 47–58.

[44] S. Kim, J. Cho, C. Lee, and T. Shon, "Smart seed selection-based effective black box fuzzing for IIoT protocol," *The Journal of Supercomputing*, vol. 76, pp. 10 140–10 154, 2020.

[45] P.-Y. Lin, C.-W. Tien, T.-C. Huang, and C.-W. Tien, "ICPFuzzer: proprietary communication protocol fuzzing by using machine learning and feedback strategies," *Cybersecurity*, vol. 4, pp. 1–15, 2021.

[46] A. Sasi, K. Hariprasad, S. Cherian, A. Sharma, J. Narayanan, and V. Pavithran, "R0fuzz: A collaborative fuzzer for ICS protocols," in *12th International Conference on Computing Communication and Networking Technologies (ICCCNT)*. IEEE, 2021, pp. 1–5.

[47] C. Wright, C. Connelly, T. Braje, J. Rabek, L. Rossey, and R. K. Cunningham, "Generating client workloads and high-fidelity network traffic for controllable, repeatable experiments in computer security," in *Recent Advances in Intrusion Detection: 13th International Symposium (RAID 2010)*. Springer, 2010, pp. 218–237.

[48] A. Dainotti, A. Pescapé, P. S. Rossi, F. Palmieri, and G. Ventre, "Internet traffic modeling by means of hidden markov models," *Computer Networks*, vol. 52, no. 14, pp. 2645–2662, 2008.

[49] X. Zhu, S. Wen, S. Camtepe, and Y. Xiang, "Fuzzing: a survey for roadmap," *ACM Computing Surveys (CSUR)*, vol. 54, no. 11s, pp. 1–36, 2022.

[50] M. Böhme, V.-T. Pham, and A. Roychoudhury, "Coverage-based greybox fuzzing as markov chain," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016, pp. 1032–1043.

[51] S. Rawat, V. Jain, A. Kumar, L. Cojocar, C. Giuffrida, and H. Bos, "Vuzzer: Application-aware evolutionary fuzzing," in *Network and Distributed System Security (NDSS) Symposium*, vol. 17, 2017, pp. 1–14.

[52] H. A. Salem and J. Song, "Grammar-based fuzzing tool using markov chain model to generate new fuzzing inputs," in *International Conference on Computational Science and Computational Intelligence (CSCI)*. IEEE, 2021, pp. 1924–1930.

[53] I. Ben-Gal, "Bayesian networks," *Encyclopedia of statistics in quality and reliability*, 2008.

[54] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.