Corvinus University of Budapest
Doctoral School of Economics, Business and Informatics

THESIS COLLECTION

Selján Gábor

Machine Learning in Security Vulnerability Research

Seed Selection with Binary Classification for Directed Fuzzing

Doctoral dissertation

Supervisor:
Dr. Racskó Péter, Csc

Budapest, 2025

Table of Contents

1. Background and motivation **3**
1.1 Theoretical justification 3
1.2 Methodological framework 4

2. Concept and methodology **6**
2.1 Literature review 6
2.2 Tool development 7
2.3 Empirical evaluation 8
2.4 Key findings 9

3. Results and conclusion **13**
3.1 Summary 13
3.2 Artifacts 15
3.3 Limitations 16
3.4 Conclusion 17

List of main references **19**

Chapter 1

Background and motivation

1.1 Theoretical justification

The rapid adoption of artificial intelligence (AI) has transformed traditional methodologies in various industries, including cybersecurity. Although AI-driven technologies enhance the ability of defenders to detect and mitigate cyberthreats, they also enable sophisticated attackers to execute increasingly complex cyberattacks. This study builds on existing research in machine learning (ML) and fuzz testing to identify alternative methodologies and introduce innovative solutions to recurring challenges.

The increasing prevalence of cyberattacks underscores the importance of robust cybersecurity measures. According to Kim and Solomon (2021), cybersecurity involves protecting individuals, businesses, organizations, and governments in cyberspace, with all actors responsible for protecting their information and data. Hamadah and Aqel (2020) examined various AI-based cybersecurity models and concluded that AI and ML introduce new possibilities and provide more effective solutions to address security challenges.

This research evaluates the impact of AI on cybersecurity by focusing on three main areas that address distinct aspects of the topic. At this stage, broader research areas are introduced without detailing specific research questions, which will be presented in subsequent sections.

- **AI in cybersecurity** - Examining the role of AI in advanced threat detection, real-time monitoring, and improved security protocols.
- **AI in security vulnerability research** - Analyzing how AI techniques can enhance vulnerability discovery, focusing on the exploitation phase of the cybersecurity kill chain.
- **AI in fuzz testing** – Exploring how AI can optimize fuzz testing methodologies, making vulnerability detection more efficient.

The primary practical contribution of this dissertation is the development of an ML-based tool to optimize the seed selection process in the fuzz testing methodology. The dissertation consists of the following four logical units:

1. **AI in cyberattacks** – This section examines how adversaries use AI to analyze vast amounts of data, uncover software vulnerabilities, and orchestrate highly effective cyberattacks. Given the increasing sophistication of AI-driven threats, defenders must deploy AI-based solutions to counter them.

2. **Directed fuzz testing** – This section examines how automated tools, using randomized input data, uncover security flaws. Although effective, this process remains resource-intensive, especially for large codebases. AI-assisted fuzzing presents a promising optimization strategy that reduces the time and computational resources required.

3. **ML-based seed selection** – Here, the research introduces a binary classification model that assesses the similarity between input samples and known crash instances. By analyzing the characteristics of the labeled samples, the model predicts which input is most likely to trigger crashes, thus refining the fuzz testing process.

4. **Case study evaluation** – The final section evaluates the effectiveness of ML-assisted seed selection through an empirical study. By fuzzing the Windows graphics component with EMF metafiles, the research uncovered several zero-day vulnerabilities, demonstrating the practical applicability of the proposed approach.

1.2 Methodological framework

The research follows a structured approach that integrates theoretical investigation with practical implementation. To ensure rigor in the development of an AI-assisted fuzz testing tool, the study adopts the *design science research methodology*, combining empirical analysis with a case study to demonstrate real-world applicability. The methodology consists of the following key stages:

1. **Literature review** – The study begins with an extensive review of the AI applications in cybersecurity, using a snowball methodology to identify relevant academic sources. The review highlights a research gap on the role of machine learning (ML) in fuzz testing.

2. **Exploration of AI-driven cyberthreats** – The study contextualizes the role of AI within the exploitation phase of the cyber kill chain. To compromise a system, adversaries must acquire zero-day exploits or conduct independent vulnerability research to identify security flaws and develop the corresponding attack vectors. Once

inside a system, adversaries prioritize remaining undetected for prolonged periods to maximize their impact.

3. **Design science research** – This dissertation employs the *design science research* approach to develop a practical AI-assisted fuzz testing tool. The methodology iteratively designs, implements, and evaluates the tool based on real-world security scenarios. The tool aims to improve the efficiency of seed selection, thereby optimizing the fuzz testing process to identify vulnerabilities.

4. **Platform and attack surface selection** – The investigation focuses on the Windows operating system, specifically targeting the graphics component as an attack surface. Despite a decline in its use, the EMF file format remains a valid vector for vulnerability research, as demonstrated in previous studies (Jurczyk, 2016). Its continued support in Internet Explorer and integration with Microsoft Office and print spooling systems make it a relevant subject for security evaluation.

5. **Case study implementation** – To assess the practical viability of the proposed method, the research applies AI-assisted seed selection to fuzz testing within a controlled experimental framework. Specifically, the tool is deployed to fuzz the Windows graphics component using EMF metafiles, leading to the identification of previously unknown vulnerabilities. The results provide empirical validation of the effectiveness of the method and illustrate its contributions to security vulnerability research.

This dissertation begins with a broad examination of AI applications in cybersecurity before refining its focus on vulnerability research, fuzz testing, and seed selection. To explore these challenges, three primary research inquiries are introduced, reflecting this narrowing of scope and providing a deeper understanding of the role of AI in addressing cyberthreats.

- **RQ1**: The first research question explores the application of AI in cybersecurity: *What are the current applications of AI and ML technologies in cybersecurity?*
- **RQ2**: The second research question investigates AI-assisted vulnerability research: *How can AI and ML technologies enhance offensive cybersecurity, particularly in the discovery of zero-day vulnerabilities?*
- **RQ3**: The third research question focuses on AI-assisted fuzz testing: *In what ways can AI and ML improve fuzz testing methodologies?*
- **RG1**: The final goal of the research is creating an ML-based fuzzing tool: *How can we implement an AI-assisted approach to optimize the initial seed selection phase of fuzz testing?*

Chapter 2

Concept and methodology

2.1 Literature review

The literature review of this dissertation adopts a methodology initially introduced by Molleri et al. (2016) and subsequently refined by Kaloudi and Li (2020). This approach ensures both comprehensiveness and reliability, reducing the likelihood of overlooking critical aspects. The process consists of defining the research topic, formulating a structured plan to gather relevant documents, and systematically reviewing the findings.

To establish the scope of the study and formulate research questions, I first defined clear objectives. The primary research inquiry that guides this investigation is the following: *How can the application of AI and ML technologies improve cybersecurity?*

To retrieve the relevant literature, I conducted an iterative search and selection process comprising three stages, as illustrated in Figure 1. The initial stage focused on identifying articles that examine the use of AI and machine learning technologies in cybersecurity, forming a foundational collection of studies for further analysis.

The review began with a manual search on Google Scholar to compile an initial set of relevant articles, ensuring an unbiased selection process independent of any specific publisher or institution. Given the novelty of the field and the challenges associated with identifying pertinent studies, a manual search was necessary to obtain comprehensive and reliable sources.

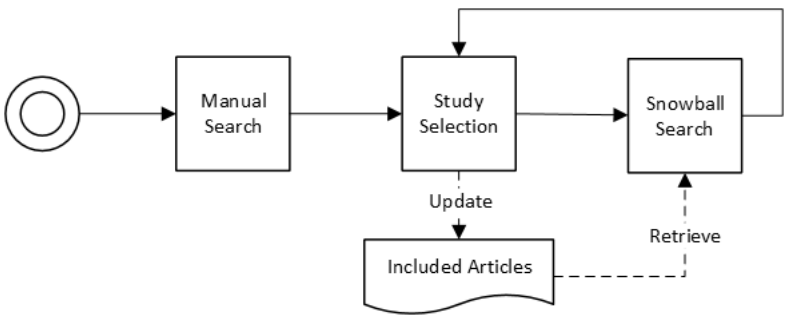

Figure 1: Overview of the search and selection process (Molleri et al., 2016)

To ensure accuracy and completeness, each search result was carefully reviewed and verified. The selected studies were chosen by analyzing their titles and abstracts to assess relevance to the research questions. Following this preliminary screening, a more detailed analysis was performed to ensure alignment with the study objectives.

The review then shifted focus to fuzz testing methodologies, with an in-depth examination of case studies on AI-assisted vulnerability research. Through an analysis of papers on various fuzz testing approaches, I assessed their objectives and the role of AI and ML technologies in enhancing effectiveness. Through a comprehensive analysis of the collected data, key insights were extracted while considering assumptions and validity to ensure the robustness of the findings.

2.2 Tool development

This research introduces an AI-assisted tool to enhance fuzz testing by optimizing seed selection for the discovery of security vulnerabilities. It develops a logistic regression model for binary classification and evaluates its ability to guide fuzzers toward security-sensitive code paths based on previous crash data.

The study follows the DS paradigm, which provides a structured approach to artifact design, construction, and evaluation. DSR, as described by Wieringa (2014), employs an iterative process to generate both practical solutions and generalizable knowledge. This methodology combines systematic inquiry with case study techniques to assess real-world applicability.

A problem-driven design approach is adopted, alternating between artifact development and evaluation. Research distinguishes between practical challenges that require implementation and theoretical inquiries that aim to advance knowledge. The central question investigated is: *How can an AI-assisted algorithm optimize fuzz testing to improve vulnerability detection?*

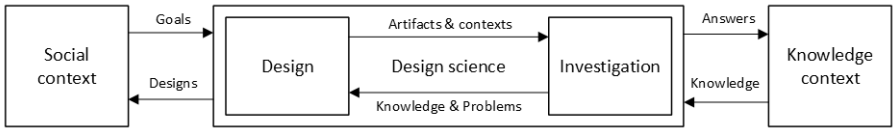

Figure 2: Design science framework (based on Wieringa, 2014)

The proposed AI-driven artifact prioritizes seed inputs with a higher probability of exposing vulnerabilities. Security researchers contribute domain expertise throughout the design process, ensuring alignment with industry needs. The methodology of the study follows five key research activities: problem analysis, requirements definition, artifact design, demonstration, and evaluation.

Johannesson and Perjons (2021) emphasize that many similar research projects focus mainly on development and evaluation. This dissertation adopts a similar approach,

designing and implementing a logistic regression model to demonstrate its application in fuzzing the Windows graphics component with EMF metafiles and experimentally assessing its effectiveness.

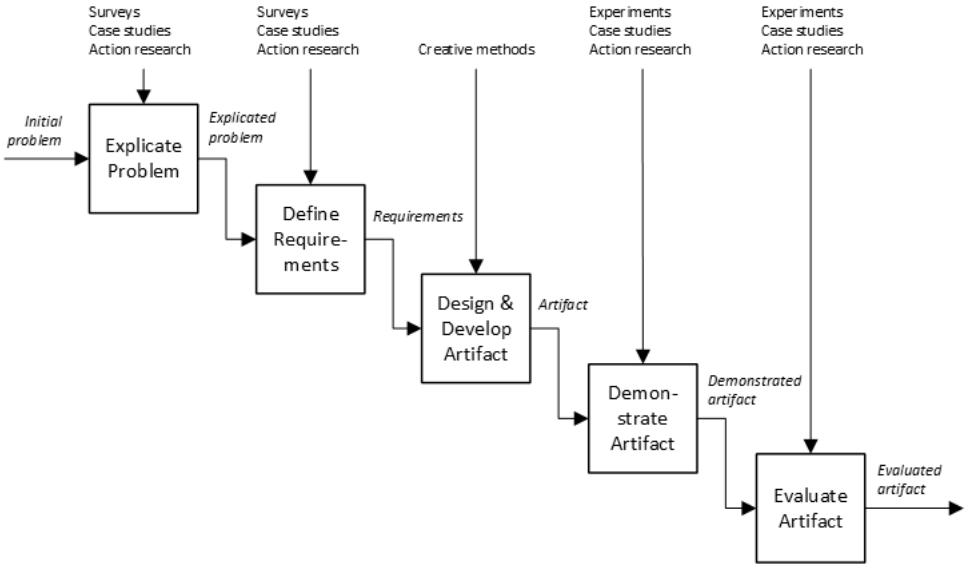

Figure 3: Examples of research strategies (Johannesson and Perjons, 2021)

Engineering and design cycles structure the iterative refinement of the artifact, without imposing a rigid development sequence. The complexity of EMF image file processing makes the Windows graphics component a suitable case study for testing and validation.

This research evaluates the feasibility of integrating machine learning into fuzz testing, with a focus on ML-assisted seed selection rather than comparing multiple ML models. Previous studies have examined ML-based tools against traditional fuzzing methods, but this study prioritizes validating the overall concept rather than engaging in extensive model comparisons.

Although formal adoption assessments using technical action research (TAR) are deferred, the effectiveness of the tool is measured through standard fuzzing techniques. Future research should investigate broader real-world impacts, including user adoption and deployment.

2.3 Empirical evaluation

The effectiveness of machine learning in fuzz testing is best assessed through the evaluation of modern software. However, this process is inherently time-consuming and presents certain limitations. During the development of the `SeedClass` tool, multiple zero-day vulnerabilities were discovered, highlighting the practical viability of the method. These findings serve as empirical evidence supporting the method's utility in vulnerability research.

Further experiments detailed in this section demonstrate that leveraging insights from previous fuzzing campaigns facilitates a more systematic approach to seed selection. To quantify the impact of seed selection informed by binary classifier predictions, a series of experiments were conducted on an outdated version of Windows 10, which is known to contain multiple security vulnerabilities. Each experiment evaluated distinct seed selection strategies while utilizing an identical candidate dataset.

- **Randomization:** Representing a baseline, this strategy selects seed files randomly without optimizing the fuzz testing process.
- **Minimization:** Selects test cases that provide optimal code coverage while reducing redundant input, using a modified `winafl-cmin.py` tool.
- **Prediction:** Leverages prior crash data to identify seed files with high bug-finding potential, using a trained binary classification model within `SeedClass`.
- **Combination:** Combines minimization and prediction strategies to maximize efficiency, first reducing the corpus and then applying the trained model to select the final set of seed files.

A candidate dataset of 1,000 files was randomly selected from a larger corpus of 24,245 files, ensuring representative sampling. To avoid bias, these files were excluded from the training dataset. From this subset, 20 seed files per experiment were selected using distinct strategies, followed by 24-hour fuzz testing campaigns. All experiments were carried out on the same physical device with identical configurations, maintaining consistency between evaluations.

Each campaign used 20 fuzzers running in parallel, resulting in approximately 20 days of total testing time. The shared memory mode facilitated efficient data transfer between the test harness and the fuzzer, while dynamic instrumentation leveraged `edge` coverage tracking on the `GdiPlus.dll` module across 5,000 execution iterations. A standardized 32-bit fuzzing harness was applied across all tests to ensure uniform execution parameters. Although campaign durations varied slightly (±10 minutes), this deviation had no measurable impact on the results.

2.4 Key findings

After each 24-hour testing period, the complete output folder from each campaign was collected. This folder contained files that triggered program crashes, along with the `fuzzer_stats` and `plot_data` files, which recorded various test metrics during fuzz testing. To extract and analyze these metrics, the `CrashResults.py` helper script was employed. This script is partially derived from the original `winafl-whatsup.py` script included in the WinAFL project.

Table 1 presents a comparative analysis of fuzz testing campaigns based on the seed selection strategies employed, randomization, minimization, prediction, and combination.

Strategy	Execs	Speed	Paths	Pending	Coverage	Stability	TTC	Crashes
Randomize	182M	2.1k	**19k**	131k	38.81%	68.11%	01:17	46
Minimize	**252M**	**2.9k**	18k	117k	39.11%	65.48%	01:11	59
Predict (balanced)	212M	2.5k	17k	118k	39.04%	64.66%	**00:11**	**108**
Predict (imbalanced)	188M	2.2k	16.2k	109k	38.51%	**70.17%**	00:54	73
Combine	154M	1.8k	17k	**137k**	**40.37%**	69.47%	04:27	62

Table 1: Summary of fuzzing output statistics

The table evaluates key performance metrics, with the best value for each metric highlighted in bold to indicate the most effective strategy for that specific measure.

The experiments associated with the prediction strategy used two different distributions of the same training dataset. To mitigate redundancy, the balanced dataset was refined using the `CrashDistribute.py` helper script, removing most duplicate sample files. In contrast, the imbalanced dataset retained all available samples. The balanced dataset aimed to prevent the model from overfitting to a specific bug that was disproportionately represented among the crash samples. The following insights emerged from this analysis:

- **Speed:** The minimization strategy achieved the highest execution count (252M) at a speed of 2.9k executions per second, suggesting extensive testing. The randomization strategy had 182M executions, running at 2.1k executions per second, while the imbalanced prediction strategy operated at 2.2k executions per second.
- **Progress:** The randomization strategy discovered the most unique paths (19k), slightly outperforming the minimization (18k) and the combination (17k) strategies. The combination strategy had the highest number of pending inputs (137k), indicating many unprocessed test cases.
- **Coverage:** The combination strategy achieved the highest code coverage (40.37%), surpassing other strategies. The imbalanced prediction strategy exhibited the highest stability at 70.17%, ensuring consistent execution conditions.
- **Bug-finding** The balanced prediction strategy was the most effective in bug detection, producing 108 crashes with a TTC of just 11 seconds. In contrast, the randomization strategy recorded 46 crashes with a TTC of 1 minute and 17 seconds, while the combination strategy had 62 crashes, taking 4 minutes and 27 seconds to detect the first bug.

Based on the experimental results, where time to crash (TTC) and the number of crashes were the most critical metrics, the balanced prediction strategy emerged as the most effective approach for bug identification. This strategy produced a high number of crashes with a short TTC, demonstrating superior efficiency in vulnerability discovery. In contrast, the minimization strategy produced fewer program crashes and took longer to detect the first bug, suggesting a slower and less effective methodology. Although the combination strategy achieved the highest code coverage, its longer TTC and lower crash count indicate that broader code exploration does not necessarily expedite bug identifica-

tion. These findings underscore the value of leveraging prior knowledge from past program crashes to enhance fuzz testing efficiency for vulnerability discovery. The ability of different seed selection techniques to identify bugs is summarized in Table 2, which also outlines the advantages and limitations of each approach in detecting program crashes.

The *randomization* strategy demonstrated robust path exploration, identifying the highest number of execution paths (19,000), although with a relatively lower coverage rate (38.81%). This suggests that while it initially achieves a lower coverage rate, coverage gradually improves as the test progresses. Despite detecting the fewest crashes (46), it successfully discovered a unique crash in the `SetROP2()` function - along with the minimization strategy - that remained undetected by more targeted approaches. This finding highlights its ability to explore diverse code paths, potentially revealing edge cases that systematic strategies might overlook. However, its overall lower crash detection rate indicates that while it is effective for exploration, refinements could enhance its utility to maximize crash identification.

The *minimization* strategy excelled in execution speed and efficiency, achieving the highest number of executions (252 million) and the fastest cumulative speed (2,900 executions per second). In addition, it detected more crashes (59) than the randomization strategy. Despite this, the performance of the fuzzer in other metrics, including path discovery, pending test cases, and coverage, did not show notable improvements. Although it successfully identified a bug in the `NamedEscape()` function alongside the *randomization* strategy, it produced fewer unique bugs overall. This result suggests that, although the minimization of test cases optimizes the fuzzing throughput, it does not significantly improve the fuzzer's ability to discover new vulnerabilities. Instead, its primary advantage lies in maximizing execution speed rather than enhancing vulnerability detection, making it well-suited for rapid test cycles but potentially less effective in uncovering deeper system flaws.

BugId	Randomization	Minimization	Prediction (balanced)	Prediction (imbalanced)	Combination
0e5.7fb	11	16	16	4	3
2b1.cf1					1
3ce.9b5	35	10	17	10	13
520.506	1	7			
5ab.a57					5
6e7.9de				1	
8ed.260			1261	847	806
8ed.13f			3	4	2
999.7fb	1				
be9.1bc			1		
d05.7fb		6	76	2	
db3.f44	28	1	1		
eb3.7fb	11	23	34	32	16
efe.7fb	3	9	11	1	2
f33.8ed			2603	2464	1726
f33.cf7			158	104	70

Table 2: Distribution of unique bugs identified

The *prediction* strategy using a *balanced* dataset demonstrated the highest efficiency in bug discovery, identifying the most crashes (108) and achieving the fastest time-to-crash (00:11). These results align with the methodology of using historical crash data to guide the fuzzer toward inputs likely to replicate known failure patterns. Compared to the *randomization* strategy, which excelled in path discovery but detected fewer crashes, the *prediction* strategy prioritized crash detection while maintaining competitive execution speed. In contrast to the *minimization* strategy, which optimizes throughput but has a limited impact on bug discovery, the *prediction* strategy presents a more effective trade-off between execution speed and crash identification. These findings suggest that predictive techniques can enhance test efficiency by directing the fuzzer toward known vulnerable code paths without compromising execution speed or broadening code coverage.

Training the model with the *imbalanced* dataset, which included all available samples, improved its performance and led to the identification of a previously undetected bug in the `GlyphDataElement<...>::GetExistingGlyph()` function. However, this adjustment did not significantly influence other key performance metrics. Bug distribution patterns indicate that predicted samples guided the fuzzer toward vulnerabilities in the `DpRegion::Set` and `DpRegion::And` functions, which were overrepresented in the dataset. This observed behavior suggests that reducing the model accuracy can help preserve the ability of the fuzzer to explore new, previously unseen code paths.

The *combination* strategy achieves a balance between crash detection and code coverage. Coverage and stability metrics indicate that this approach is well-suited for systematic bug discovery, demonstrating effective path exploration while maintaining testing stability.

Multiple seed selection strategies consistently detected certain crashes, confirming their reliability in identifying specific bug patterns. Notably, only the *randomization* and *minimization* strategies detected the bug in the `NamedEscape()` function, highlighting their potential to uncover previously unseen vulnerabilities. Meanwhile, the *combination* strategy identified bugs in the `CopyGindices()` and `CreateFontIndirectWImpl()` functions, indicating that combining seed selection methods can improve the fuzzer's ability to discover software defects.

In conclusion, the *prediction* strategy proved to be the most effective for directed fuzz testing, utilizing failure patterns from historical crash data to optimize bug discovery. The *combination* strategy provided a balanced approach, improving test stability and code coverage while maintaining consistent bug detection rates. The *randomization* strategy excelled in code path exploration and successfully identified a bug in the `SetROP2()` function. Meanwhile, the *minimization* strategy optimized testing efficiency but demonstrated limited effectiveness in detecting novel program crashes.

Chapter 3

Results and conclusion

3.1 Summary

This dissertation aimed to address a real-world challenge in security vulnerability research by leveraging insights from existing literature on the application of machine learning in cybersecurity. A key objective was the development of `SeedClass`, an ML-based tool designed to leverage the knowledge gained from long-term fuzz testing campaigns through a binary classification model.

Data is foundational to any machine learning project. To obtain training data for the classifier, I used a combination of files scraped from the Internet and testcases generated via the `FormatFuzzer` tool. For the latter, I developed an enhanced binary template for EMF files, achieving approximately 80% coverage of the graphic operations exposed through metafile records.

Beyond the development of the prototype of the machine learning model, the study emphasized its practical implementation in fuzz testing. A case study involving a selected software component demonstrated how this approach can be applied in real-world scenarios.

The case study required the creation of a dedicated fuzzing harness that serves as an interface between the fuzzer and the targeted software component. Furthermore, I integrated the `bug bucket` functionality into the `WinAFL` fuzzer, streamlining the triaging of crash samples by categorizing them according to the memory address of the instruction that triggered the crash.

Given that fuzz testing seeks to uncover security vulnerabilities manifesting as program crashes, the primary evaluation metric was the number of crashes identified. Previous research has established a correlation between increased code coverage and higher crash counts, making the number of distinct code paths explored within a given timeframe an important secondary evaluation criterion.

The evaluation considered fuzzing campaigns employing different seed selection strategies, guided by two key factors: (1) time costs - the duration required to select and analyze

initial inputs, and (2) test efficiency - the number of code paths explored and crashes identified under varying selection approaches.

Experimental results demonstrate that incorporating insights from previous fuzz testing campaigns enhances the fuzzing process by utilizing a binary classification model. Furthermore, the study identified several zero-day vulnerabilities, which Microsoft publicly acknowledged and addressed through security updates. In general, this research contributed to the development of efficient tools for identifying and mitigating security threats through fuzz testing.

This dissertation introduced an enhanced binary template for the Enhanced Metafile (EMF) file format in the 010 Editor, improving the coverage of graphic operations. The template achieves approximately 80% coverage of available metafile records, making it a valuable asset for fuzz testing Windows GDI through metafiles. Security researchers can utilize the template with the `FormatFuzzer` tool to generate input files for machine learning experiments and fuzz testing.

Furthermore, this research presented a fuzzing harness for transmitting metafiles to the target system, specifically Windows GDI, which proved to be effective in fuzz testing. The harness executes various operations on input files, such as generating thumbnails or converting EMF files to Windows Metafile (WMF) format, triggering diverse program behaviors and improving code coverage. As a critical component of the approach, the harness ensures that genuine program errors can be revealed.

To efficiently manage extensive fuzzing campaigns, I developed `WinAFL Pet`, a remote monitoring and management tool that simplifies the configuration and oversight of multiple fuzzer instances across remote machines. This application played a pivotal role in enhancing both scalability and manageability, ensuring an effective testing process.

Since the research aimed to maximize identified program crashes, the dissertation extended the `WinAFL` fuzzer with a feature that facilitates bug evaluation: the ”bug bucket” functionality. By recording memory addresses associated with crashes, this feature groups samples based on the instruction that triggered the crash, streamlining the identification of unique cases and significantly aiding the evaluation process.

Furthermore, the dissertation introduced `SeedClass`, a machine learning-based tool that uses binary classification to refine seed selection from large datasets. Implemented as a prototype in `TensorFlow`, the tool trains a binary classifier on labeled seed inputs and uses it to predict the potential of unlabeled samples to trigger program crashes, thus optimizing the fuzzing efficiency.

To support further research, I have made the dataset used to train the machine learning (ML) model publicly available, along with helper scripts for data preprocessing. During data collection, I identified more than 50 unique samples capable of reproducing security vulnerabilities, making them invaluable resources for researchers studying flaws in the Windows graphics component. To facilitate future investigations, I also published addi-

tional software artifacts, including the enhanced binary template for EMF files, the fuzzing harness for Windows graphics testing, and the implementation of the binary classifier prototype. These resources, available at GitHub, promote transparency and reproducibility in vulnerability research while providing useful tools for security professionals. By sharing these materials, I hope to foster innovation and collaboration within the field.

In conclusion, this research advances vulnerability discovery and demonstrates the potential of machine learning in security analysis. The real-world impact of the study is evident in the identification of seven zero-day vulnerabilities in the Windows graphics component, validating its practical applicability. Furthermore, the findings highlight the broader potential of machine learning techniques in improving vulnerability research methodologies, marking a valuable contribution to the field.

3.2 Artifacts

This chapter presents the key results and artifacts developed during this doctoral research, highlighting contributions that enhance various aspects of fuzz testing, including data processing and bug detection. The advances achieved in this study collectively contribute to the progress of vulnerability research.

A1: Improved binary template for the `EMF` format.

The first artifact is an improved binary template for the EMF image file format, designed for compatibility with both the widely recognized `010 Editor` and the innovative `FormatFuzzer` test case generator. This template addresses a gap in existing tools by covering approximately 80% of available metafile records. Its accuracy was validated using `010 Editor` as a parser and `FormatFuzzer` as a generator, ensuring reliability for future fuzz testing applications.

A2: Fuzzing harness for the Windows graphics component

The second artifact is a fuzzing harness, which plays a crucial role in the testing process. It is responsible for initializing the testing environment and integrating the target software component. By encapsulating the target program and preparing it to process `EMF` files, the harness enables effective interaction with the Windows graphics component. Compared to other publicly available harnesses designed for similar purposes, this implementation offers enhanced functionality, supporting file-based and shared-memory fuzzing while enabling broader testing of graphics operations, such as thumbnail creation and metafile conversion, and incorporating useful debugging features.

A3: `Bug Bucket` feature for deduplication of crashes

The third artifact is an enhancement to the `WinAFL` fuzzer, designed to streamline the process of deduplicating crash samples. During fuzz testing, the fuzzer may detect multiple distinct crashes that, in reality, stem from the same underlying vulnerability. This feature logs the memory address associated with each program crash and groups previ-

ously identified crashes within a dedicated `bug bucket`, simplifying the triaging process by separating redundant crash samples.

A4: `WinAFL Pet` fuzz manager for remote monitoring

The fourth artifact is `WinAFL Pet`, a fuzz manager developed to provide a web-based interface to remotely manage `WinAFL` fuzzer instances. This tool simplifies the monitoring of fuzzing campaigns across remote machines using a lightweight agent that operates as a system service. `WinAFL Pet` significantly improves the usability of `WinAFL` by offering an intuitive interface to manage and configure fuzzing campaigns efficiently.

A5: `SeedClass` Prototype for Seed File Selection

The fifth and primary artifact is a working prototype of the `SeedClass` tool, designed to optimize the seed selection process in fuzz testing. Coverage-guided fuzzers generate thousands of mutated test cases, some of which uncover new code paths that lead to crashes. However, given the time- and resource-intensive nature of fuzz testing, identifying high-value seed inputs is crucial to improving efficiency. `SeedClass` addresses this challenge by employing a machine learning model to predict valuable seed files within a candidate dataset. This approach facilitates more informed input selection, enabling seamless adaptation and integration with various fuzzers.

3.3 Limitations

Like all research projects, the prediction-based seed selection strategy explored in this dissertation has inherent limitations. This section outlines the study constraints and the potential challenges associated with applying the proposed seed selection approach.

The primary strength of the method also serves as its main limitation. The dissertation focused on supervised machine learning, which means that the prediction-based seed selection strategy relies on accurately labeled input samples to train the binary classifier. Consequently, this approach is unsuitable in scenarios where known vulnerabilities and their corresponding input samples, which can reproduce program crashes, are unavailable. Furthermore, to prevent the method from hindering the fuzzer's ability to discover new execution paths, proper preprocessing of the training data is required, which can limit scalability. The experiments carried out during this investigation confirmed that the proposed seed selection strategy effectively guides the fuzz testing process. However, its success remains significantly dependent on the quality and representativeness of the training samples.

This project specifically focused on optimizing seed selection in fuzz testing to increase the likelihood of discovering novel program crashes. However, assessing the exploitability of these crashes—whether they constitute security vulnerabilities or benign software bugs —falls outside the scope of this research. Exploitability analysis has emerged as a distinct research domain, with previous studies demonstrating that machine learning

methods can be used to assess software exploitability (Yan et al., 2017) and predict the probability of vulnerability exploitation (Bhatt et al., 2021). These findings emphasize that predicting exploitability requires specialized methodologies distinct from the seed selection strategies examined in this dissertation. Therefore, throughout this work, the terms *bug* and *vulnerability* are used interchangeably.

A crucial consideration in designing this seed selection method was ensuring that it did not negatively impact the fuzzer. As a result, the prototype was implemented as a separate step at the beginning of the fuzz testing process. However, this leaves open the question of whether closer integration with specific fuzzers is feasible. One promising possibility for integration would involve filtering newly discovered code paths and using the sample files identified by the model predictions as a seed corpus when restarting a fuzzing campaign.

From an evaluative perspective, this dissertation aimed to demonstrate the applicability of a binary classification model to seed selection. However, individual evaluations of seed selection strategies and fuzz testing methodologies can lead to misleading conclusions, as the results are often difficult to compare between different studies, as noted by Klees et al. (2018). A more comprehensive evaluation of the binary classifier should consider a broader range of seed selection strategies and involve the fuzzing of various file formats.

The data encoding mechanism, responsible for defining the features that inform the model, is another crucial aspect of implementation, alongside compiling a suitable dataset for training. M. Wang and Robinson (2023) have proposed an alternative approach that presents a potential direction for future research. Their method converts the input into Unicode bytes, transforms them into a sequence of integer values, and then normalizes these values. Padding the remaining entries with null bytes to meet input size requirements without modifying the underlying input could potentially enhance model performance; however, further research is necessary to evaluate its effectiveness.

Finally, although fuzz testing the Windows graphics component served only as a case study rather than the central focus of this dissertation, the EMF+ file format presents additional complexity. It includes several previously unexplored metafile records, constituting an expanded attack surface. Applying the outlined approach to fuzz testing the EMF+ format offers a promising avenue for future research.

3.4 Conclusion

This dissertation connects machine learning and security vulnerability research by presenting an ML-powered seed selection strategy for directed fuzz testing. The proposed approach prioritizes input samples that show the characteristics of known crash patterns and demonstrates how prior fuzzing campaign data can serve as training input for an ML model. The resulting binary classifier, `SeedClass`, predicts which input samples can

guide the fuzzer toward security-sensitive code paths that have been previously affected by known vulnerabilities.

To support this core contribution, I developed several practical artifacts. A dedicated fuzzing harness was implemented for the Windows graphics component, although its core functionality can be adapted for use with other targets. Furthermore, I designed a remote management tool and enhanced `WinAFL` to enable large-scale experiments and facilitate crash detection. Furthermore, I created an improved binary template to improve input generation for ML-based fuzzing. To encourage further research, I published the complete training dataset, data preprocessing scripts, and the `SeedClass` prototype on GitHub.

Experimental results validate the effectiveness of the prediction-based seed selection strategy. Compared to conventional seed selection methods, prediction-driven fuzzing campaigns discovered the first crash more quickly and identified a higher number of bugs within the same time frame. This approach led to the disclosure of seven previously unknown vulnerabilities, underscoring its practical impact.

Despite these successes, certain limitations remain. The method's reliance on supervised learning constrains its applicability, as the classifier's performance depends on the availability and quality of labeled crash samples. Additionally, the current workflow remains an offline, one-shot preprocessing step. To generalize the approach, broader validation across diverse file formats, tighter integration into the fuzzing loop, and exploration of alternative encoding schemes, such as Unicode-based feature representations, are needed.

Looking ahead, integrating `SeedClass` into an online feedback loop could enable continuous model refinement based on newly discovered execution paths. Expanding the framework to semi-supervised or unsupervised learning methods may reduce dependence on labeled samples. By combining machine learning with vulnerability discovery, this dissertation contributes to the advancement of adaptive data-driven fuzzing frameworks and lays a foundation for further innovations in security vulnerability research.

List of main references

Bhatt, N., Anand, A., & Yadavalli, V. S. (2021). Exploitability prediction of software vulnerabilities. *Quality and Reliability Engineering International*, *37*. https://doi.org/10.1002/qre.2754 (cit. on p. 17).

Cheng, L., Zhang, Y., Zhang, Y., Wu, C., Li, Z., Fu, Y., & Li, H. (2019). Optimizing Seed Inputs in Fuzzing with Machine Learning. *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, 244–245. https://doi.org/10.1109/ICSE-Companion.2019.00096

Dutra, R., Gopinath, R., & Zeller, A. (2023). FormatFuzzer: Effective Fuzzing of Binary File Formats. https://doi.org/10.48550/arXiv.2109.11277

Guembe, B., Azeta, A., Misra, S., Osamor, V. C., Fernandez-Sanz, L., & Pospelova, V. (2022). The Emerging Threat of AI-driven Cyber Attacks: A Review. *Applied Artificial Intelligence*, *36*(1), 2037254. https://doi.org/10.1080/08839514.2022.2037254

Hamadah, S., & Aqel, D. (2020). Cybersecurity Becomes Smart Using Artificial Intelligent and Machine Learning Approaches: An Overview. *ICIC Express Letters, Part B: Applications*, *11*(12), 1115–1123. https://doi.org/10.24507/icicelb.11.12.1115 (cit. on p. 3).

Hayes, L., Gunadi, H., Herrera, A., Milford, J., Magrath, S., Sebastian, M., Norrish, M., & Hosking, A. L. (2019). MoonLight: Effective Fuzzing with Near-Optimal Corpus Distillation. *arXiv preprint*. https://doi.org/10.48550/arXiv.1905.13055

Herrera, A., Gunadi, H., Hayes, L., Magrath, S., Friedlander, F., Sebastian, M., Norrish, M., & Hosking, A. L. (2019). Corpus Distillation for Effective Fuzzing: A Comparative Evaluation. *arXiv preprint*. https://doi.org/10.48550/arXiv.1905.13055

Herrera, A., Gunadi, H., Magrath, S., Norrish, M., Payer, M., & Hosking, A. L. (2021). Seed Selection for Successful Fuzzing. *Proceedings of the 30th ACM SIGSOFT international symposium on software testing and analysis*, 230–243. https://doi.org/10.1145/3460319.3464795

Johannesson, P., & Perjons, E. (2021). *An Introduction to Design Science*. Springer International Publishing. https://doi.org/10.1007/978-3-030-78132-3 (cit. on pp. 7, 8).

Jurczyk, M. (2016, October). Windows Metafiles: An Analysis of the EMF Attack Surface & Recent Vulnerabilities. https://j00ru.vexillium.org/talks/ruxcon-windows-metafiles-an-analysis-of-the-emf-attack-surface-and-recent-vulnerabilities/ (cit. on p. 5).

Kaloudi, N., & Li, J. (2020). The AI-based Cyber Threat Landscape: A Survey. *ACM Comput. Surv.*, *53*(1). https://doi.org/10.1145/3372823 (cit. on p. 6).

Kim, D., & Solomon, M. (2021). *Fundamentals of Information Systems Security*. Jones & Bartlett Learning. https://books.google.hu/books?id=DiVGEAAAQBAJ (cit. on p. 3).

Klees, G., Ruef, A., Cooper, B., Wei, S., & Hicks, M. (2018). Evaluating Fuzz Testing. *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*, 2123–2138. https://doi.org/10.48550/arXiv.1808.09700 (cit. on p. 17).

Li, Y., Ji, S., Lv, C., Chen, Y., Chen, J., Gu, Q., & Wu, C. (2019). V-Fuzz: Vulnerability-Oriented Evolutionary Fuzzing. https://doi.org/10.48550/arXiv.1901.01142

Lyu, C., Ji, S., Li, Y., Zhou, J., Chen, J., & Chen, J. (2018). SmartSeed: Smart Seed Generation for Efficient Fuzzing. *CoRR*, *abs/1807.02606*. https://doi.org/10.48550/arXiv.1807.02606

Miller, B. P., Fredriksen, L., & So, B. (1990). An Empirical Study of the Reliability of UNIX Utilities. *Communications of the ACM*, *33*(12), 32–44. https://doi.org/10.1145/96267.96279

Molleri, J. S., Petersen, K., & Mendes, E. (2016). Survey Guidelines in Software Engineering: An Annotated Review. *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. https://doi.org/10.1145/2961111.2962619 (cit. on p. 6).

Rajpal, M., Blum, W., & Singh, R. (2017). Not All Bytes are Equal: Neural Byte Sieve for Fuzzing. https://doi.org/10.48550/arXiv.1711.04596

Rebert, A., Cha, S. K., Avgerinos, T., Foote, J., Warren, D., Grieco, G., & Brumley, D. (2014). Optimizing Seed Selection for Fuzzing. *23rd USENIX Security Symposium (USENIX Security 14)*, 861–875. https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/rebert

Saavedra, G. J., Rodhouse, K. N., Dunlavy, D. M., & Kegelmeyer, P. W. (2019). A Review of Machine Learning Applications in Fuzzing. https://doi.org/10.48550/arXiv.1906.11133

She, D., Pei, K., Epstein, D., Yang, J., Ray, B., & Jana, S. (2019). NEUZZ: Efficient Fuzzing with Neural Program Smoothing. *2019 IEEE Symposium on Security and Privacy (SP)*, 803–817. https://doi.org/10.1109/SP.2019.00052

Wang, J., Chen, B., Wei, L., & Liu, Y. (2017). Skyfire: Data-Driven Seed Generation for Fuzzing. *2017 IEEE Symposium on Security and Privacy (SP)*, 579–594. https://doi.org/10.1109/SP.2017.23

Wang, M., & Robinson, M. (2023). Vulnerability Detection Through an Adversarial Fuzzing Algorithm. *arXiv preprint arXiv:2307.11917*. https://doi.org/10.48550/arXiv.2307.11917 (cit. on p. 17).

Wang, Y., Jia, P., Liu, L., Huang, C., & Liu, Z. (2020). A Systematic Review of Fuzzing based on Machine Learning Techniques. *PloS one*, *15*(8). https://doi.org/10.1371/journal.pone.0237749

Wang, Y., Wu, Z., Wei, Q., & Wang, Q. (2019). NeuFuzz: Efficient fuzzing with deep neural network. *IEEE Access*, *7*. https://doi.org/10.1109/access.2019.2903291

Wieringa, R. J. (2014). *Design Science Methodology for Information Systems and Software Engineering*. Springer. https://doi.org/10.1007/978-3-662-43839-8 (cit. on p. 7).

Yan, G., Lu, J., Shu, Z., & Kucuk, Y. (2017). ExploitMeter: Combining Fuzzing with Machine Learning for Automated Evaluation of Software Exploitability. *Proceedings - 2017 IEEE Symposium on Privacy-Aware Computing, PAC 2017*, *2017-January*. https://doi.org/10.1109/PAC.2017.10 (cit. on p. 17).

Zakeri Nasrabadi, M., Parsa, S., & Kalaee, A. (2021). Format-Aware Learn&Fuzz: Deep Test Data Generation for Efficient Fuzzing. *Neural Computing and Applications*, *33*, 1497–1513. https://doi.org/10.1007/s00521-020-05039-7

Zhu, Q., & Liang, L. (2019). Research on Security Vulnerabilities Based on Artificial Intelligence. *Intelligent Computing Theories and Application: 15th International Conference, ICIC 2019, Nanchang, China, August 3–6, 2019, Proceedings, Part I 15*, 377–387. https://doi.org/10.1007/978-3-030-26763-6_37