Using AI for Automated Penetration Testing

Linnea Gilbertson
gilb0348@morris.umn.edu
Division of Science and Mathematics
University of Minnesota, Morris
Morris, Minnesota, USA

Abstract

Penetration testing (pentesting) is a key practice in cybersecurity, helping to identify security vulnerabilities in devices and applications. Due to system complexity and time constraints, pentesting often involves the use of specially designed tools. However, these tools are likely to introduce a high rate of false positives. To reduce the time required for pentesting and enhance the accuracy of tests, machine learning has been integrated into pentesting tools. This paper covers one such tool, the BERT QA RL + RS, that integrates both deep reinforcement learning (DRL) and a bidirectional encoder representations from transformers (BERT) large language model (LLM).

Keywords: penetration testing, large language model, deep reinforcement learning, bidirectional encoder representations

1 Introduction

The number of electronic devices per person worldwide has grown in the last couple of years, increasing from 2.4 devices per person globally in 2018 to 3.6 devices per person in 2023. Not only do devices currently outnumber people [13], but the number of vulnerabilities found is also increasing yearly. In 2024, 40,077 new vulnerabilities were found and published in the Common Vulnerabilities and Exposures (CVE) database, which is an increase of 38% from the number of new vulnerabilities published in 2023 [5]. In this cyber-vulnerable world, penetration testing is of increasing importance. Penetration testing (pentesting) is a series of processes or tests that mimic a real world attack on a complete and complex system or application to identify its vulnerabilities [4]. More about penetration testing is covered in Section 2.1.

Pentesting often involves the use of tools to analyze or find vulnerabilities and security risks, or tools that assist in the exploitation process (getting unauthorized access to the application or system). These tools are useful, but often result in high rates of false positives, which in this context would mean the tools show vulnerabilities or exploitation routes that do not exist. Recently, there have been attempts to create new, more accurate tools to assist in pentesting via the use of Machine Learning (ML). Reinforcement learning (RL) is a ML technique that is able to rapidly react to diverse target applications or systems and perform exploitation in a way that closely mimics reality. However, RL alone is not enough, as proof of concept tests are needed to validate any findings from the RL. Also, to ensure the scope of the RL's exploration is narrow and thus does not take too long, very specific goals must be set for the RL agent. This is where a large language model (LLM) is helpful, as it can be prompted to give the needed proof of concept tests based on past data, as well as send specific and understandable commands to the RL agent to ensure the scope of its exploration is narrow. The BERT Question-Answer RL + Recommendation system (BERT QA RL + RS) leverages a bidirectional encoder representations from transformers (BERT) LLM along with a deep reinforcement learning agent to make a new penetration testing tool greater than the sum of its parts [10].

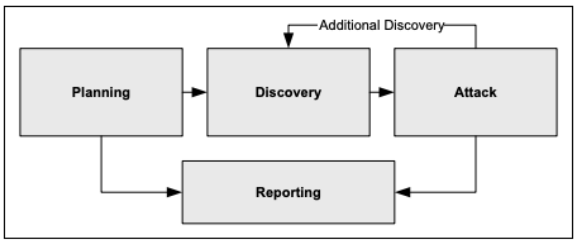

Figure 1. Steps for black-box penetration testing [12]

2 Background

2.1 Penetration Testing

To increase the security of applications and devices, penetration testing (pentesting) is often performed by human cyber security experts (pentesters) [4]. Pentesting is designed to mimic real world cybersecurity attacks and involves launching real attacks to break into a system, or otherwise damage it, using technology and techniques that are commonly used by malicious hackers. Pentesting is also helpful for evaluating the cyber defenders' ability to detect and respond to attacks on their system [12].

Black-box penetration testing is penetration testing performed without prior knowledge of the system's structure or source code. It is designed to mimic the way a real hacker would try to break into a system. Black-box pentesting is the best technique for assessing the security of the interfaces between different components of a system, and the security of interactions with the external environment, its users, or other systems it interfaces with. The National Institute of Standards and Technology Special Publication 800-115 (NIST

SP 800-115) defines a structured series of black-box penetration testing steps, as outlined in Figure 1 [12]. The BERT QA RL + RS System was designed to follows these steps, and uses tools commonly used by human pentesters for each of these steps, integrating them into its own system [10].

To follow these black-box pentesting steps, first the pentester should plan, or in other words set testing goals and notify management of the upcoming pentest. Then comes the discovery phase, which can be split into two parts. The first part is reconnaissance, in which information about the system is gathered [12]. One common tool used for this purpose is Network Mapper (Nmap), which performs network scanning, port scanning, and operating system detection [4]. Part 2 of the discovery phase is vulnerability analysis, where the pentester compares the discovered services and operating systems against one or more databases of known vulnerabilities and system weaknesses [12]. Such databases that will be utilized in the BERT QA RL + RS include the Common Vulnerability and Exposures (CVE) database, which is a published list of security vulnerabilities, and the Common Weakness Enumeration (CWE), which is a public list of underlying weaknesses in applications that can lead to vulnerabilities, often linking to one or more CVEs [10].

Then comes the attack phase, also called the exploitation phase. Using the vulnerabilities and system architecture identified in the previous steps, the pentester finds a way to get into or otherwise damage the system involved, exploiting the vulnerabilities. Oftentimes this first attack does not result in getting the needed level of access, but it often reveals more information about the system. This new information can be used to repeat vulnerability analysis, and then another exploitation is attempted [12]. The BERT QA RL + RS uses Metasploit in order to help simulate this process. Metasploit is a tool that can assist in selecting optimal attack paths as well as develop exploit code [10]. Once the pentester is in the system, they often perform a sub-phase of the attack called post-exploitation, where information can be stolen using the access gained, or the pentester can use additional vulnerabilities to elevate their level of access in the system or gain access to additional connected systems. And last is the reporting phase, where vulnerabilities and other findings are reported so they can be fixed or otherwise mitigated [12].

2.2 BERT Language Models

A large language model (LLM) is a machine learning model that is able to generate natural language in response to a prompt [3]. LLMs are able to do this thanks to their billions of internal weights, or in other words, parameters that allow them to capture patterns in language and predict words in a response by creating a probability distribution of the most likely next words. LLMs train on large amounts of text data, adjusting their internal weights to get better responses. All language models, including BERT, make use of tokens. Tokens in language models represent the smallest data units that a language model processes, whether they be words, individual characters, or something else. The first step in training language models is transforming the input to the model into tokens (tokenization), then using different methods to detect patterns and relationships in the text. Also, after tokenization is completed, the large language model assigns a unique ID to each token [7].

The Bidirectional Encoder Representations from Transformers (BERT) that is used in the BERT QA RL + RS is a kind of masked language model (MLM) designed by Devlin et al. BERT was created to be easily fine-tunable for a wide range of tasks, including question answering and inference, without many architecture modifications. Part of what makes the BERT so easily fine-tunable is that it is a MLM. MLMs randomly remove some of the tokens from the input, training themselves by predicting the IDs of the removed tokens based only on the given context [6].

Tokens are then used to train embeddings. Embeddings are numerical representations of categorical feature. Categorical features are attributes of real or virtual objects that represent distinct classifications or categories. For example, the type of fruit could be a categorical feature. An example predetermined set of classifications for the fruit categorical feature could be "apple", "banana", or "strawberry". BERT uses 3 different types of embeddings. The token IDs in BERT contain token embedding, meaning each token ID is not random, but is instead based on which tokens the BERT model consider most similar to one another during training. Bert also uses segment embeddings, which capture whether a token is part of the question or answer being input to the machine in a BERT QA. Lastly, the BERT uses position embeddings to label the initial position of the token in the input text or prompt [2].

2.3 Deep Reinforcement Learning

Deep reinforcement learning (DRL) is a method that combines deep neural networks (elaborated on in Section 2.4) and reinforcement learning (RL) [8]. Reinforcement learning consists of an agent, which is the decision maker that acts in accordance with its policy and engages with an environment [11]. The environment is the system that the agent gets data from and interacts with. In this paper, the environment will be the target system the BERT QA RL + RS is pentesting [10]. The policy is a plan that controls the agent's decisions by having it do certain actions on certain states. A state is a representation of the current environment, and an action is a possible decision an agent can take. The agent's goal is to maximize the overall reward it receives, and the reward is the feedback it receives from the environment based on its current decision. The agent learns how to maximize this reward by convergence, the process in which the policy changes and moves more towards the optimal policy for getting the most reward possible.

Basic reinforcement learning must always have a model of the environment, which is its internal representation of the environment and states in the environment [8]. However, one of the reasons Moreno et al. chose deep reinforcement learning is that DRL does not need to have this model of the environment. This is because it can rely on Quality Learning (Q-learning), which is a model-free RL method [11]. This is important because in pentesting environments, changes in services and potential vulnerabilities are common, making the states too large in number and too changeable to easily make the model of the environment needed for typical RL [10]. Also, DRL makes use of deep neural networks, which can help create more intricate policies by picking up on the mappings of numerous states to actions, and is able to be used with far more states and actions than a typical RL could [8].

2.4 Neural Networks

This section will cover neural networks (NN), as they are used both in the BERT language model and in deep reinforcement learning. A neural network is a machine learning model that identifies and learns patterns directly from data. In NNs, nodes are the most basic units that receive inputs from the initial data and from previous nodes. Nodes have an activation function, which is the formula that calculates what the node should output based on what inputs are fed to that node. Connections are the links between nodes that transfer the data or inputs, and they are regulated by weights and biases. Weights and biases are numerical parameters that are used to determine the strength and influence of the effect of the connection on the activation function of the node it's connected to.

In NNs, the initial data is first fed to a layer of nodes called the input layer. Each node in this layer corresponds to an individual feature of the initial data. Then it is fed through one or more hidden layers. These hidden layers are where most of the computation is performed via the weights and biases of the connections and the activation functions of the nodes. The output of the hidden layers is then fed to the output layer. The output layer produces the final output of the model, which are the final numerical value or values that actually mean something. Different models create different final outputs depending on the task the model performs. During training the NN puts its data through all of these steps and then adjusts its weights and biases to improve accuracy on its specific task by minimizing the difference between the expected model output, which is the output that is desired and would be output if the model was perfect, and the actual model output, with the actual model output being the final numerical value or values the model yielded [14].

The BERT QA part of the BERT QA RL + RS system uses a Feed-Forward neural network (FNN) [10]. An FNN is a NN where data can only move and be transformed in one direction, forward, and each layer uses the data values transformed from the previous layer [9]. The reinforcement learning agent of the BERT QA RL + RS uses a deep neural network (DNN) as it uses deep reinforcement learning [10], as detailed above. A DNN is just a feed-forward neural network with many hidden layers [1].

3 BERT QA RL + RS Overview

The BERT QA RL + RS is a model that uses both an LLM and DRL to identify cybersecurity vulnerabilities and possible tests to execute to confirm or deny that the system being tested has such vulnerabilities. As seen in Figure 2, the BERT QA component of the model is first trained on many CWE cases in the different domains of pentesting. These cases include those on recognition, vulnerability analysis, and exploitation.

As well as using these cases to train, the BERT will integrate the information into its database for use in future responses. Once the BERT QA is done training, the pentester can query or send a question to the BERT QA model and the model will attempt to infer an answer or response. The question is based on what the pentester knows about the system architecture and hopes to accomplish.

If a response can be generated, the BERT will output it. If no response can be generated the question is sent to the RL agent automatically. If the pentester is not satisfied with the response, they can manually send the question to the RL agent. The RL agent imitates the steps a human pentester would take based on the steps recommended by the NIST SP 800-115, as outlined in Section 2.1. These steps include Reconnaissance, Vulnerability Analysis, and Exploitation.

Then, the RL agent assembles the results into a new JSON data set and sends it back to the database. The BERT then undergoes a new round of training, incorporating the new data. This will enable the BERT QA to produce new responses when queried, specifically responses pertaining to the question asked earlier that was either unable to produce a useful response or unable to produce any response. The BERT will not attempt to infer this new response unless the pentester sends the same or a similar question.

Combining the RL Agent with the BERT QA to make the BERT QA RL + RS produces a helpful tool that can not only provide recommendations based on past CWE cases, but can also pentest the target system itself to provide suggestions when necessary. In the next two sections, the different parts of the BERT QA RL + RS will be elaborated on in depth [10].

4 BERT QA

A BERT Question Answer (QA) Recommendation System (RS) at its core, consists of a query/question Q, which is a query that is based on what the pentester hopes to accomplish and knows about the system architecture and the specific goals for this pentest. In order to train the BERT QA,

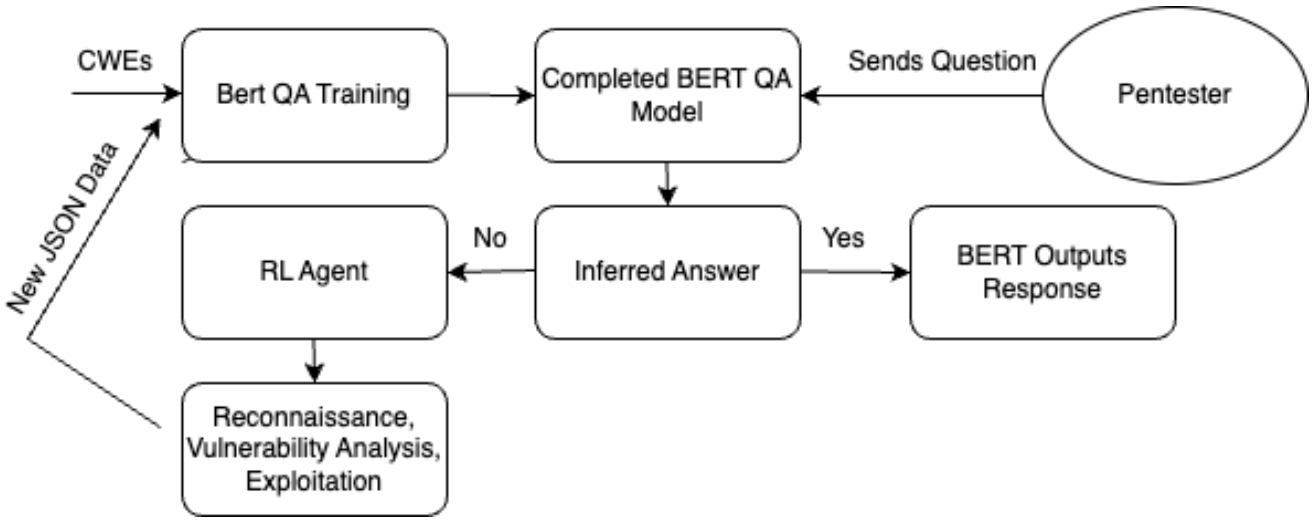

Figure 2. Diagram of the BERT QA RL + RS recommendation system [10]

example Qs must be provided that are associated with Contexts (C) and example factual responses to be predicted called Answers (A). A is a response to Q that can detail how to perform pentesting tests, or how to perform reconnaissance, identity specific vulnerabilities and how to exploit them or perform post-exploitation. Cs are contexts that are associated with the vendor (the provider of information about the vulnerability or test), one or more CWEs, description of vulnerability, and specific ways in which the vulnerability can be exploited and tested (proof of concept).

The Qs, As, and Cs used to train the BERT QA in the research article were generated using the CWEs in the NIST vulnerability repository. The researchers first identified 171 different CWE types from 43,080 vulnerabilities. The questions (Q) were formulated to contain information such as the system environment, possible system versions, and other known information about the target system. An example Question is shown below:

> *"Q = What tests do you recommend for a Class C IP address, with Ubuntu 20.0 operating system, running an Apache PHP 5.2.4 server?"*

During the training of the BERT QA, As and Cs are fed another time to the BERT QA as one sentence without a separator between them, forming AϵC, which can be thought of as A in C. AϵC is a response/answer (A) defined with context C, where the context comes first and the answer comes second, as shown below:

> *"C = According to CWE-116, CWE-79, and CWE-94, with improper neutralization of resource inputs, enabling potential remote code execution," A = "it is possible to use a proof of concept for XSS and then inject arbitrary code by modifying functions.lib.php."*

The complete inputs used in training the BERT QA end up being a tuple of Q and AϵC, with the places where A starts and ends within AϵC marked.

Next the architecture of the BERT QA will be elaborated on, as is seen in Figure 3. The BERT QA is used in order to select potential answers based upon the asked question. The inputs Q and C (which is actually AϵC) are tokenized, with the [SEP] token separating these inputs and the [CLS] token placed in front to indicate that it is a classification task for the BERT.

The tokenized input is then used to train a vector of embeddings which contain the token embeddings, segment embeddings, and position embeddings. Then a combined embedding is produced that represents the concatenation of all embeddings.

As seen in Figure 3, next the combined embedding undergoes many layers of transformation, which are hidden in Figure 3 under the transformer layers. The first of which is Multi-Head attention. Multi-Head attention aids in learning the relationships between words that are both close to and distant from one another.

Then the output of the Multi-Head attention is passed into the feed-forward neural network (FNN), which applies nonlinear transformations and creates a language representation that integrates information from Q and AϵC by capturing unique characteristics of elements in the sequence. Then normalization is applied to enhance generalization and convergence as well as stabilize learning. Finally, a softmax layer is applied, which outputs a probability distribution, which estimates the probabilities that the Q input corresponds to a specific answer A.

If a satisfactory Answer is not found via this process, or in other words there is no A with a high enough probability

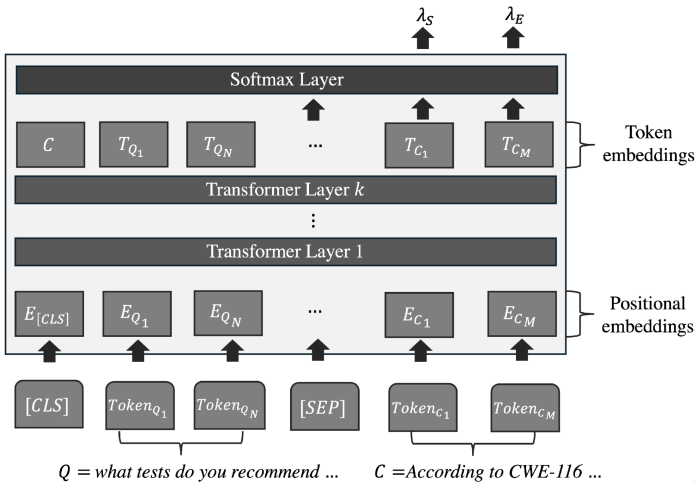

Figure 3. Architecture of BERT QA [10]

of being correct, then the response will be a note indicating that the Q was sent to the RL agent for evaluation.

5 Reinforcement Learning Agent

In order to both train itself and conduct a penetration test, the RL Agent interacts with a system D that has vulnerabilities V. In a typical RL, a state space is required that represents the possible environmental configurations that exist in V, and an action space S that consists of all possible actions a the RL agent can take. The RL agent selects an action a when given the current state, with the goal being maximizing a reward r.

The RL agent imitates the steps a human pentester would take based on the steps recommended by the NIST SP 800-115, as outlined in Section 2.1.

1. *Planning and Preparation:* The RL agent first uses the decomposition of the question asked by the pentester to define the objectives and scope of the subsequent penetration test steps it will undergo.

2. *Reconnaissance:* In order to identify useful system architecture, including host, services, and open ports, the RL agent uses Network Mapper (Nmap). More on Nmap is detailed above in background Section 2.1. The agent also correlates these findings with configurations of system architecture stored in the BERT QA RL + RS database, which speeds up the process of obtaining results for this phase.

3. *Vulnerability Analysis:* Then using the system architecture identified in the reconnaissance phase, the RL agent uses the Nmap Vulners tool to compare this system architecture with previously noticed vulnerabilities in the CVE database in order to identify possible vulnerabilities.

4. *Exploitation and Post-exploitation:* The agent then uses Metasploit modules to find the attack path, as well as methods of exploitation and post-exploitation. More on Metasploit is described in Section 2.1.

The RL agent undergoes multiple iterations of the above series of four steps, while adjusting its policies in order to achieve the maximum r. The RL agent gets rewards for performing successful actions a, with a higher reward for a complicated successful action. If an action fails, the RL agent returns to the current state and changes the policy to be more flexible until it converges. A step is concluded when the max Q-value is reached for that step. The max Q-value corresponds to the highest estimated r that the RL agent can gain from all possible actions in the states. At each stage, the agent either selects the most effective action learned so far or selects random action in S, which ensures the exploration of a wide variety of actions and a better convergence towards a maximum Q-value.

The RL agent concludes either when all of the above steps have succeeded with their max Q-values, or it will truncate if the agent can for some reason no longer make progress towards the maximum Q value. After the RL agent concludes, the state action pairs with the highest rewards are assembled into a JSON data set. Each output will have a target to evaluate, the characteristics of the target, the parameters used in by Nmap in the reconnaissance step, the vulnerabilities identified via use of Nmap Vulners, and finally the steps for successful exploitation via the Metasploit modules. If the agent did not reach its goal in a specific step, then it will be marked down in the JSON as a failed attempt. Then, whether it succeeds or not, the JSON object is sent to the BERT QA, which integrates this information into its database and begins a new round of training using the obtained information.

The RL Agent was tested by using virtual environments constructed via the OpenAI Gym library in Python version 3 on two different virtual machines (VMs). One VM was based on the Linux 20.04 OS, and the other was based on Windows 8 and 10. Both VMs hosted a variety of vulnerable services. The researchers determined the most persistent vulnerabilities that still affect systems today via the maturity reports of the Cybersecurity and Infrastructure Security Agency. In the end, about 1520 vulnerable configurations were assembled for the two VMs. The RL agent has not yet been tested on real-world systems [10].

6 Overall Model Results

The researchers tested three different BERT models to find out which works best for the BERT QA RL + RS system. These models include BERT uncased, the first and most basic BERT model, RoBERTa which leverages a larger dataset and uses a different masking approach, and DistilBERT, which is a lightweight BERT version that retains most performance but has a much lower computational cost.

Training loss, which represents the discrepancy between the BERTs actual output answers and the predicted answers it should output for all answers, was used as a measurement

Table 1. BERT QA Training Metrics [10]

Model	Loss	Training (min)
BERT uncased	0.0001	1297.5
RoBERTa	0.0000	1299.8
DistilBERT	0.0043	689.1

Table 2. Accuracy Metrics for BERT QA RL + RS [10]

Model	Exact Match (%)	Precision (%)	Recall (%)
BERT uncased	97.5	98.0904	98.4848
RoBERTa	99.9998	99.9999	99.9998
DistilBERT	99.8763	99.9057	99.8763

of the models accuracy. The formula is $L = 1/N * \sum(A - \text{Expected A})^2$ The Loss and training time taken is shown above in Table 1.

The model with the best Loss value and greatest overall response accuracy was the RoBERTa, with a highly impressive lost value of 0.0000. It also had the greatest training time, taking 1299.8 minutes. Since the RL agent's average training time is 34 minutes, the average training time of the most accurate BERT QA RL + RS system about 1334 minutes, or 22.2 hours. However, if speed and computational efficiency are more crucial to the pentester, they could use the DistilBERT instead. DistilBERT has a training time of 689 minutes, so this BERT QA RL + RS version would only have a total training time of 12.1 hours. This is just over half the training time of the most accurate version.

Although the response time of the BERT QA RL + RS depends on which BERT model is used, overall it will only be approximately 2 minutes if the BERT model provides an inference or response based off of the training data. If the question is sent to the RL agent due to a lack of response or unsatisfactory answer, the RL agent will take about 21 additional minutes. It should be noted that the training and testing was conducted in intentionally vulnerable test environments instead of real-world environments. Moreno et al. predicted that both training and execution times would be higher in a real-world system.

The next component that will be discussed is the Question Answer (QA) accuracy metrics for the BERT QA RL + RS. The following metrics evaluate the quality of the responses made by the BERT QA RL + RS. These include Precision, which assesses the accuracy of the model's response. It is a measure of the proportion of correct words in the response relative to all the words in the response. Its formula: $Precision = \frac{\text{Number of correct words in response}}{\text{total words in response}} * 100$. The Recall metric is a measure of whether the model response contains the key words of the expected response. It measures the proportion of correct words in the response to the words in the expected response/answer. The formula: $Recall = \frac{\text{Number of correct words in response}}{\text{total words in expected response}} * 100$. Finally, Exact Match measures the percent of the responses that exactly match the expected/ correct response. The metric only counts the answers that match exactly with their expected answer. That formula is: Exact match = $\frac{\text{Number of correct responses}}{\text{Total responses}} * 100$. Table 2 above includes all these metrics for all BERT QA RL + RS tested, differentiated by BERT model used.

Interestingly, even the BERT QA RL + RS using the lightweight DistilBERT had better accuracy metrics in all category than the BERT QA RL + RS using the BERT uncased, despite DistilBERT having worse loss metrics than BERT uncased. Also all accuracy metrics of DistilBERT were extremely close, within .2% of the accuracy metrics for the best BERT QA RL + RS using RoBERTa. Combined with DistilBERT's lowered computational cost, this suggests that using DistilBERT in a BERT QA RL + RS system may be the best choice. This is unless someone has a demand for extremely accurate responses, in which case one should go with RoBERTa.

7 Conclusion

As the number of devices in human society increases, the potential for harm caused by malicious hackers increases in parallel. Thus, it is vital that pentester's have access to accurate pentesting tools. However, the tools currently available also have a high rate of false positives, wasting the pentesters time. In order to create more accurate tools, many researchers have attempted to integrate machine learning. Moreno and her fellow researchers created the BERT QA RL + RS, the first penetration testing tool to integrate both deep reinforcement learning and a BERT large language model.

The BERT QA RL + RS has demonstrated remarkable accuracy, with all accuracy metrics being 99% or above for both the highest performing and most computationally efficient BERT QA RL + RS model. The total training time of the lightweight model is also only 12.1 hours. When the machine is prompted, the machine takes about 23 minutes when the reinforcement learning agent is activated. This reduces both the time and resources it takes to pentest complex systems. By providing recommendations with context, it is also easier for the pentester to verify the correctness of the recommendation, saving pentester time.

It should be kept in mind that the BERT QA RL + RS has currently only been trained and tested on intentionally static and vulnerable test systems/environments. Real-world systems are typically more variable, and change incrementally over time. Real systems are also hopefully not as vulnerable as the intentionally vulnerable test environments. Thus the model's training and execution times are expected to be significantly higher in a real world environment. Future research should focus on evaluating and improving the BERT QA RL + RS performance on these real systems.

Acknowledgments

I would like to thank my advisor, Kristin Lamberty, for all the wonderful help she gave me during the development of this paper, including providing helpful suggestions, helping with latex, and helping in figure editing. I would also like to thank my course professor, Elena Machkasova, for all the wonderful suggestions and feedback.

References

[1] 2017. https://stats.stackexchange.com/questions/182734/what-is-the-difference-between-a-neural-network-and-a-deep-neural-network-and-w

[2] 2023. https://tinkerd.net/blog/machine-learning/bert-embeddings/

[3] AltexSoft. 2023. https://www.altexsoft.com/blog/language-models-gpt/

[4] Aileen Bacudio, Xiaohong Yuan, Bill Chu, and Monique Jones. 2011. An Overview of Penetration Testing. *International Journal of Network Security Its Applications* 3 (11 2011), 19–38. https://doi.org/10.5121/ijnsa.2011.3602

[5] MITRE Corporation. 2024. https://www.cve.org/about/Metrics

[6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv:1810.04805 [cs.CL] https://arxiv.org/abs/1810.04805

[7] Sloan Haywood, Genevieve Warren, and Alex Wolf. 2024. Understanding tokens. https://learn.microsoft.com/en-us/dotnet/ai/conceptual/understanding-tokens

[8] iamrajmihir. 2023. https://www.geeksforgeeks.org/a-beginners-guide-to-deep-reinforcement-learning/

[9] Adil Lheureux. 2024. Feed-forward vs Feedback Neural Networks. https://www.digitalocean.com/community/tutorials/feed-forward-vs-feedback-neural-networks

[10] Ariadna Claudia Moreno, Aldo Hernandez-Suarez, Gabriel Sanchez-Perez, Linda Karina Toscano-Medina, Hector Perez-Meana, Jose Portillo-Portillo, Jesus Olivares-Mercado, and Luis Javier García Villalba. 2025. Analysis of Autonomous Penetration Testing Through Reinforcement Learning and Recommender Systems. *Sensors* 25, 1 (2025). https://doi.org/10.3390/s25010211

[11] prateek bajaj. 2025. https://www.geeksforgeeks.org/what-is-reinforcement-learning/

[12] Karen Scarfone, Murugiah Souppaya, Amanda Cody, and Angela Orebaugh. 2021. NIST SP 800-115. https://www.nist.gov/privacy-framework/nist-sp-800-115

[13] Petroc Taylor. 2023. Number of devices and connections per person worldwide 2023 | statista. https://www.statista.com/statistics/1190270/number-of-devices-and-connections-per-person-worldwide/

[14] veena ghorakavi. 2025. https://www.geeksforgeeks.org/neural-networks-a-beginners-guide/