Alzheimers

Ngoh Si Yao Donovan, Mervin Tham,Suriya Sivakumar, Lim Jun Xiong Aledxander
Infocomm Technology Cluster
Singapore Institute of Technology
Singapore
2303695@sit.singaporetech.edu.sg, 2303720@sit.singaporetech.edu.sg, 23030729@sit.singpaoretech.edu.sg,
23030715@sit.singpaoretech.edu.sg

I. Introduction

Our project, Alzheimers, is designed to push the boundaries of rootkit evasion by combining two traditionally separate domains: low-level memory obfuscation and high-level behavioral camouflage. The goal is to create a framework that not only hides malicious code from forensic inspection but also generates realistic, benign-looking system and network activity to mask its presence. By blending these two layers of stealth, Alzheimers introduces a new paradigm in adversarial simulation and detection evasion.

At its core, Alzheimers leverages Page Table Entry (PTE) manipulation and Memory Address Space (MAS) remapping to conceal runtime activity. These techniques allow malicious code to execute while presenting altered or decoy memory views to forensic tools, effectively disrupting the assumptions made by memory scanners and debuggers. On top of this memory stealth layer, the framework employs a machine learning model trained on benign system logs to generate plausible, context-aware event sequences. This ensures that injected events blend seamlessly with normal user and system behavior, unlike simplistic log spammers that produce repetitive or random noise.

We chose the name Alzheimers because it reflects the project's central theme, just as the medical condition disrupts memory recall, our framework disrupts the ability of forensic tools and analysts to accurately reconstruct system memory and activity. Instead of erasing evidence, which is often a red flag for investigators, Alzheimers overwhelms the environment with realistic, benign traces, making it extremely difficult to distinguish genuine activity from synthetic camouflage

II. Background Research

A. Literature Review

I. Hiding Process memory Via Anti-Forensic Techniques

In "Hiding Process Memory via Anti-Forensic Techniques", Palutke, Block, and Reichenberger describe how attackers can undermine memory forensics by altering the way operating systems manage and expose process memory. The authors outline three primary anti-forensic strategies: Page Table Entry (PTE) manipulation, Virtual Address Descriptor (VAD) removal, and direct memory tampering. Each of these methods targets a different layer of the memory management system. PTE manipulation changes the low-level mappings between virtual and physical memory, VAD removal disrupts the metadata that links memory regions to processes, and direct tampering overwrites or spoofs memory contents to confuse forensic tools. Together, these techniques demonstrate how fragile forensic assumptions can be when adversaries deliberately subvert memory structures [1].

For our project, the most relevant of these are PTE manipulation and Memory Address Space (MAS) remapping. While the paper focuses on PTE manipulation and VAD removal, the same principles extend naturally to MAS remapping, which operates at a broader region level. PTE manipulation allows fine-grained control over individual pages, making malicious code appear non-present, non-executable, or mapped to benign frames. MAS remapping, by contrast, enables entire memory regions to be redirected to alternate physical frames, masking large allocations such as heaps or image sections.

II. Machine Learning in Digital Forensics and Anti-Forensics

This paper includes the Techniques, Limitations and Recommendations by Yaacoub et al. provides a comprehensive analyis of the evolving integration with machine learning (ML) is becoming crucial for both forensic analysis and anti-forensic detection. To start with, a new branch of "Anti-Anti-Forensics" or "Counter Anti-Forensics" has emerged which heavily relies on machine learning for early and intelligent detection of anti-forensic activities. This is necessary as ML can analyze vast datasets and identify subtle patterns indicative of evidence tampering that would be impossible for human analysts to detect manually. [2]

Unlike traditional anti-forensics, Alzheimers does not delete or conceal evidence and instead drowns it. Instead of creating inconsistencies that raise suspicion, it introduces excessive consistency. To elaborate, our tool employs a pretrained machine learning model trained on user-behavior datasets to generate plausible, context-aware event sequences. Rather than relying on simple log-spamming or random noise generation, it simulates full behavioral personas and simulates realistic actions file edits, scheduled tasks, browser queries, service interactions, even sleep/wake cycles. Alzheimer's objective is not to hide malicious actions, but to bury them beneath mountains of convincingly legitimate activity, forcing forensic analysts and ML-based detectors to sift through overwhelming volumes of semantically valid data.

III. State-of-the-Art in Adversarial Generation for Evasion

AdvTG: An Adversarial Traffic Generation Framework demonstrates how a Large Language Model (LLM), fine-tuned with Reinforcement Learning (RL), can generate malicious network traffic that evades deep learning detectors [3]. Its key insight is the separation of traffic into "functional fields" (the essential malicious command) and "non-functional fields" (headers, padding, etc.). The model learns to generate benign-looking non-functional data around the malicious core, creating packets that are both functional and evasive.

PROVNINJA: Evading Provenance-Based ML Detectors with Adversarial System Actions focuses on evading host-based intrusion detection systems that analyze system event graphs [4]. Instead of executing single, conspicuous actions,

PROVNINJA substitutes them with multi-step, statistically-common "gadget chains" of legitimate processes that achieve the same outcome. Furthermore, it "camouflages" each step by adding benign side-effect events based on frequency analysis of normal system behavior.

B. Existing Tools and Solutions

i. MASHKA (Malware Analysis System for Hidden Knotty Anomalies)

MASHKA is a solution where a resilient memory acquisition and analysis system is used to uncover hidden kernel objects. It works by systematically walking page tables, capturing page dumps together with their physical address mappings, and applying dynamic and signature-based detection methods to reveal concealed processes or drivers. In essence, MASHKA is designed to counter many OS-level anti-forensic tactics by validating memory content at multiple layers instead of trusting standard API or OS-reported structures [5]. While traditional tools try to hide evidence, Alzheimers adds an overwhelming volume of plausible evidence, creating a crime scene with many false leads which makes it harder and longer for investigators to perform forensic analysis.

ii. Dementia

Dementia is a tool that bypasses forensics tools which inspect attacker's footprints in a Windows computer's memory. It is designed to hide various artifacts inside the memory dump. It is able to hide operating system artifacts such as processes and threads from applications such as Volatility. Dementia evades memory capture by intercepting NtWriteFile() calls through the use of inline hooking and a file system mini-filter [6].

iii. Shadow Walker Rootkit

Shadow Walker is a rootkit that evades memory analysis and capture by differentiating and filtering execute, read, and write accesses to its own memory sections. A page fault handler hooks and desynchronizes the two TLBs (Translation Lookaside Buffers) utilized by the processor for address translation caching. When a read operation occurs, the rootkit responds by feeding random garbage data to the request. Hence, this effectively avoids detection and analysis [7].

iv. TrafficManipulator

TrafficManipulator (2021-present) uses Particle Swarm Optimization algorithms for adversarial traffic generation, successfully evading Kitsune NIDS and other learning-based network security systems [8]. The Adversarial Robustness Toolbox (ART), maintained by the Linux Foundation, provides comprehensive frameworks for evasion, poisoning, extraction, and inference attacks against ML models [9].

III. PROPOSED SOLUTION

We will be creating a tool that hides the activities of the rootkit by mimicking processes and network activity based on specific benign system activity patterns of the target environment and memory forensics evasion techniques.

i.Layer 1: Memory Evasion (Hiding the Agent)

We're using PTE manipulation and MAS remapping together to hide what the rootkit is doing at runtime. PTE changes let us control access to specific memory pages, making malicious code run while staying hidden from normal inspection tools. MAS remapping helps by redirecting larger memory regions to different physical locations with altered access settings. Combined, these techniques let us mask the rootkit's activity and bypass standard detection methods.

ii.Layer 2: Behavioral Camouflage (Hiding the Actions)

Instead of deleting evidence, which is easily detectable, our tool will operate by generating and injecting large amounts of synthetic, benign-looking system and network events. We do this by using a machine learning model trained on our own dataset to generate plausible, context-aware event sequences based on next sequence prediction on events that the system/user has already performed. This sets it apart from simplistic log spammers that generate repetitive, random noise.

1. Data Collection and Model Training

Our approach begins with creating a custom logging system to capture comprehensive behavioral data over a period of multiple days. This data collection phase focuses on normal user activity patterns including process creation and termination, file I/O operations, registry access, network connections, and DLL loading sequences. We plan to gather data from different user types such as developers, office workers, and casual users to create diverse behavioral profiles that our model can learn from.

The machine learning model we will implement uses a hybrid architecture that addresses a critical challenge in modeling system events: the effective representation of text-based features like file paths and process names. Simple categorical encoding fails to capture their inherent structural and semantic relationships, so our framework will employ a sophisticated feature representation strategy inspired by state-of-the-art NLP techniques.

We will begin by implementing a embedding-based approach, similar to that used in the PROVNINJA study, where each unique entity (such as a complete file path like C:\Windows\System32\cmd.exe) is mapped to a dense vector via an embedding layer. This treats each unique file path, process name, and registry key as a discrete token in our vocabulary, creating a lookup table where every unique string gets its own learned vector representation. These vector representations will be processed by our LSTM-Transformer architecture to learn high-level behavioral sequences and temporal patterns.

Furthermore, we will explore a tokenization strategy which involves separating filepaths and commands into individual folder/file names (for example, C:\Program Files\Microsoft Office\WINWORD.EXE becomes the sequence: C:, \, Program Files, \, Microsoft Office, \, WINWORD.EXE).

This approach allows our GAN to generate more contextually aware events that can generalize to file paths not seen in the training data.

2. Host-based Event Generation

The event generation system works by capturing the system state using our custom logger. The system events will be the input for our GAN LSTM-Transformer model. The model will use the last N events as context. Based on this history, it predicts and generates a sequence of plausible next-step events that are statistically and semantically consistent with the ongoing user/system behavior.

When the rootkit needs to perform malicious actions, it queries our ML model for equivalent gadget chains that can

achieve the same objective through seemingly legitimate means. For example, instead of directly exfiltrating data through an obvious network connection, the system might generate a sequence where explorer.exe accesses files, followed by schtasks.exe creating what appears to be a scheduled backup task, and finally bitsadmin.exe uploading data that looks like a routine cloud backup operation.

This ensures that each generated event appears authentic. The timing patterns will be matched to normal software behavior, and noise injection will be used as a smoke bomb to mask any potentially anomalous patterns that might arise.

3. Network Traffic Generation

For network communications, our framework will implement an advanced camouflage engine, adapting the core principles demonstrated in the AdvTG paper. We will separate malicious network traffic into "functional" and "non-functional" components. The functional component (the essential C2 data) is preserved, while our GAN LSTM-Transformer generates the surrounding non-functional components (headers, metadata, padding) to make the entire packet appear benign.

Our goal is to disguise command-and-control (C2) communications as routine activities like software updates, cloud service synchronization, or social media API calls.

The innovation in our approach is not the concept of payload embedding itself, but rather its integration within our dual-layer framework. While AdvTG focuses solely on the network vector, Alzheimers correlates this network camouflage with synthetically generated, benign host-based events.

REFERENCES

[1] R. Palutke, F. Block. P. Reichenberger, and D.Strunk, “Hiding process memory via anti-forensic techniques,” Forensic Science International: Digital Investigation, vol. 33, pp. 301011, 2020. [Online]. Avaliable: https://dfrws.org/wp-content/uploads/2020/10/2020_USA_paper-hiding_process_memory_via_anti-forensic_techniques.pdf

[2] J.-P. A. Yaacoub, H. N. Noura, O Salman, and A. Chehab, “Digital Forensics vs. Anti-Digital Forensics: Techniques, Limitations and Recommendations,” arXiv:2013.17028 [cs], Mar.2021, Avilable: https://arxiv.org/abs/2103.17028.

[3] G. Trovato et al., "AdvTG: An Adversarial Traffic Generation Framework to Deceive DL-Based Malicious Traffic Detection Models," in Proceedings of the ACM Web Conference 2025, 2025. [Online]. Available: https://dl.acm.org/doi/10.1145/3696410.3714876

[4] S. Mukherjee et al., "Evading Provenance-Based ML Detectors with Adversarial System Actions," in 32nd USENIX Security Symposium, 2023. [Online]. Available: https://www.usenix.org/conference/usenixsecurity23/presentation/mukherjee

[5] I. Korkin, "Applying Memory Forensics to Rootkit Detection," Jun. 2015. [Online]. Available: https://arxiv.org/search/?query=MASHKA&searchtype=all&source=header

[6] P. Zdzichowski, M. Sadilon, T. Uolevi, V. Alvaro, B. Munoz, and K. Filipczak, "Tallinn 2015 Anti-Forensic Study," 2015. Available: https://www.ccdcoe.org/uploads/2018/10/AF_with-intro.pdf

[7] S. Sparks and J. Butler, "'Shadow Walker' - Raising The Bar For Rootkit Detection," *Black Hat Briefings*, 2005. Available: https://blackhat.com/presentations/bh-usa-05/bh-us-05-sparks.pdf

[8] D. Tang, "TrafficManipulator: An automatic packet crafting tool for evading learning-based NIDS," 2021. [Online]. Available: https://github.com/dongtsi/TrafficManipulator

[9] "Adversarial Robustness Toolbox (ART) - Python Library for Machine Learning Security," IBM Research. [Online]. Available: https://github.com/Trusted-AI/adversarial-robustness-toolbox