Poster: Towards a Secure and Practical System to Obfuscate Tor Network Traffic

Minjae Seo[†*], Myoungsung You[†*], Taejune Park[‡], Seungwon Shin[†], and Jinwoo Kim[§**]
[†]*KAIST,* [‡]*Chonnam National University,* [§]*Kwangwoon University*

***Abstract*—Tor (The Onion Routing) has emerged as a promising open-source privacy network for providing anonymous communication in sensitive tasks, such as journalism and activism. However, it is also susceptible to deanonymization attacks, particularly flow correlation attacks, which identify users by correlating unique traffic flow characteristics observed at the ingress and egress segments of a Tor connection. To mitigate these attacks, various traffic obfuscation techniques have been developed; however, they often suffer from bandwidth waste and CPU resource exhaustion, critical issues in the Tor ecosystem where efficient utilization of the limited, voluntarily-shared bandwidth and resource of nodes is crucial. In this paper, we propose `ODIN`, a Tor-tailored hardware-based security solution that hinders adversaries from launching flow correlation attacks. Unlike previous approaches that primarily added randomly crafted padding bytes to packets, `ODIN` innovatively minimizes bandwidth waste by slicing and reorganizing packets destined for the same server with the aid of the context of Tor circuits, effectively addressing the issue of wasted network resources.**

1. Introduction

Tor (The Onion Routing) is an open-source privacy network that enables Internet users, particularly those in contexts with active censorship such as journalists, activists, or whistle-blowers, to access anonymous online services. The Tor network is widely used, with over 3 million daily users and a total of 6,000 volunteer relay nodes transmitting terabytes of traffic every day.

As the nature of Tor aims to facilitate anonymous communication for sensitive tasks, Tor is naturally a target for many deanonymization attacks. Among them, the *flow correlation attack* [8] is obviously the most powerful attack correlating unique characteristics of traffic flows (e.g., packet sizes, interval times) observed from the ingress and egress segments in a Tor connection. The pioneering research shows that it is effective for deanonymizing Tor clients and servers. For example, RAPTOR [10] achieved a 90% accuracy in deanonymizing user identity. Also, the state-of-the-art flow correlation of DeepCorr [9] offered a correlation accuracy of 96% even with shorter flow observations than previous methods required.

In recognition of the threat posed by flow correlation attacks, several *traffic obfuscation* techniques [3], [4], [6] have been developed to mitigate them. However, these techniques are not well suited to the requirements of the Tor network due to the following issues:

CPU Resource Exhaustion. Existing secret Tor relay nodes (e.g., bridge nodes [4]) or pluggable transports (e.g., obfs4, meek [6]) are utilized to conceal the fact that clients are using the Tor network by randomizing packet bytes or disguising a Tor connection as other legitimate connections (e.g., Skype). However, these techniques rely on software-based systems to process individual packets, which significantly consumes CPU resources compared to using ordinary Tor relay nodes. Among them, it is particularly crucial for exit nodes to ensure efficient resource utilization, as they play a vital role in maintaining uninterrupted routing and communication in the Tor network.

Tor Bandwidth Waste. Other techniques [5], [7] add crafted padding bytes to the original packets and control packet transmission timing to obfuscate the size and interval time of individual packets. However, the inclusion of padding bytes in these methods imposes additional burdens on individual packets and can waste Tor network bandwidth, particularly in situations where a limited number of volunteer nodes are shared by many Tor clients. As a result, most existing obfuscation methods are not well adapted to be used practically in the real Tor network.

In this paper, we propose `ODIN`, a hardware-based security solution that addresses both issues raised by existing solutions. To conserve CPU resources, `ODIN` offloads all Tor traffic obfuscation logic to a programmable hardware network interface card (i.e., SmartNIC). Also, `ODIN` addresses the issue of bandwidth waste by slicing and reorganizing packets destined for the same server with the aid of the context of Tor circuits. This approach allows `ODIN` to obfuscate the original traffic pattern by assembling two distinct flows, significantly saving bandwidth and CPU resources compared to existing solutions.

We implement a full prototype of `ODIN` using the NVIDIA BlueField 2® SmartNIC [2] and conduct extensive experiments within a realistic Tor testbed environment. In our private testbed, we establish a trusted exit node, `ODIN`, responsible for transmitting Tor traffic to the server. Additionally, we implement a receiver program, utilizing recent kernel networking features, designed to operate on the destination server. The receiver program ensures that Tor services maintain transparency when deploying `ODIN` by restoring the combined packets received from `ODIN`. In our evaluation, we primarily demonstrate the effectiveness of obfuscation and performance improvements by presenting different aspects of traffic flow after the deployment of `ODIN`.

2. Technical Background and Preliminaries

2.1. Tor Circuit

Tor works by leveraging a core technology known as *onion routing*. Onion routing transmits encrypted data (in multiple layers of encryption) to the final destination (i.e.,

[*]*Co-first authors,* [**]*Corresponding author*

server) through a minimum of **three** types of relay nodes in the Tor network.

Entry node, also known as the guard node, serves as the initial *ingress* point in the Tor network, receiving connections directly from Tor clients. Its main role is to receive multi-layer encrypted traffic from clients, decrypt the outermost layer, and forward the partially decrypted traffic to the next node in the Tor network. It is worthy note that while the entry node has access to the client's *identity* (e.g., IP address) and can detect the client's utilization of Tor, it is unable to decrypt the final destination or the content of the client's traffic due to the multi-layer encryption mechanisms inherent in onion routing.

Middle node is the second node to which the Tor client connects and acts as a linking chain. It serves a crucial role by stripping the second layer of encryption and seamlessly routing Tor traffic between the entry and exit nodes. Crucially, due to the principles of onion routing, this middle node is unable to discern the identities or information of either the client or the server.

Exit node is the last *egress* point where Tor traffic finally hits the public Internet. It communicates directly with the server, which is the final destination, making it capable of identifying the server's true identity and decrypting the encrypted layers of Tor traffic. As a result, the exit node becomes a desirable target for several entities (e.g., law enforcement agencies and threat actors) to intercept Tor traffic in the middle.

Given its significance, note that several reputable organizations and privacy-conscious entities already operate *trusted exit nodes* [1] with enhanced security features and capabilities. These trusted exit nodes are specifically designed to protect both the Tor network and service operators from potential hazards and threats associated with exit nodes, thereby enhancing the overall security and integrity of the Tor ecosystem.

2.2. Flow Correlation Attack

In most scenarios of a flow correlation attack, adversaries attempt to link Tor traffic observed from the two points, *ingress* and *egress*. The two points, carrying the information of the real *identity* (e.g., client or server), can be an attractive target in order for adversaries to eavesdrop on Tor traffic in the middle. If adversaries collect enough traffic from both points, they can have the capability of performing a flow correlation attack with following pioneering traffic analyses tailored to Tor network.

Statistical Metric. Several studies have used a statistical metric to measure the flow similarity of traversed flows observed from both *ingress* and *egress* points. For example, Sun et al. [10] used the Spearman correlation coefficient, which centers on a nonparametric measure that can evaluate the statistical dependence between the rankings of two given variables. They extracted the TCP sequence and acknowledgement number from each packet trace to conduct asymmetric correlation analysis that allows adversaries to observe any direction of the Tor traffic at both points (*ingress* and *egress* points). However, one obvious *problem* arises from the fact that they consider the case where there is a long-lasting Tor connection (inevitably needs a long flow observation) rather than aims to consider an intermittent Tor connection which is pervasive in the real world Tor network.

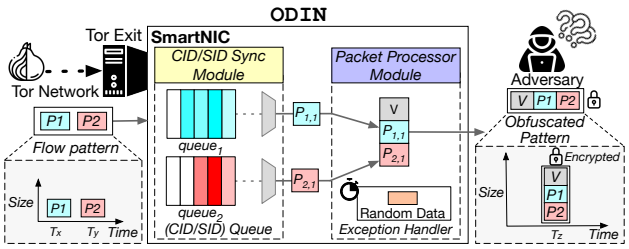

Figure 1: ODIN combines packets destined for the same server, thereby rendering the pattern of Tor traffic indiscernible to adversaries.

Deep Learning. To address the issue derived from the statistical metric, a new approach has emerged to measure the flow similarity even when considering a short-lived nature of Tor network. For example, Nasr et al. [9] proposed the state-of-the-art deep learning-based flow correlation technique, DeepCorr [9], that is able to correlate Tor traffic with high accuracy using very short-term observations of Tor connections. Due to the content encryption for each packet, they leveraged an advanced deep learning model in order to learn hidden patterns of flow-level features in Tor traffic. Specifically, they found dominant characteristic in the feature of packet *sizes* and *timings* when correlating Tor traffic. Through extensive experiments, they showed the dangers posed by exposure of flow-level features to adversaries. Thus, it is necessary to hinder adversaries from learning unique patterns through these features.

3. ODIN Overview

3.1. Threat Model

The goal of adversaries is to deanonymize clients by correlating ingress flows and egress flows. They compare several unique characteristics of Tor traffic (e.g., packet sizes and timings) instead of attempting to decrypt the content of Tor traffic. To achieve their goal, we consider adversaries who have capabilities to eavesdrop on Tor traffic (especially ingress and egress flows) passively.

We believe this scenario is practical in the real world. In recent years, many threat actors are running hundreds of malicious Tor nodes in order to intercept Tor traffic [11]. Adversaries can further leverage BGP hijacking attacks, which becomes increasingly frequent, by a means to redirect the Tor traffic to themselves [10]. There might also be more powerful adversaries, such as governmental agencies, who can perform wiretapping attacks directly on multiple Internet ASes or intercontinental fiber optics. We contemplate the deployment of ODIN on a trusted exit node, managed securely by reputable entities, mitigating the risk of adversary compromise.

3.2. System Design

As shown in Figure 1, the design of ODIN is primarily aimed at achieving a secure and practical obfuscation system that addresses the following three key aspects:

Context-aware Obfuscation. In order to develop a Tor-tailored obfuscation system, ODIN ensures consistency by leveraging the context information of established Tor circuits. As shown in Figure 1, the CID/SID Sync module within ODIN perpetually synchronizes with the circuit identifier (CID) and stream identifier (SID) information

derived from the Tor application through its default API. The CID provides specificity about the Tor circuit that a connection refers to, whereas the SID identifies distinct TCP flows. Utilizing both CID and SID, ODIN combines distinct flows that are destined for the same server.

Bandwidth-saving Obfuscation. In an environment where a limited number of Tor nodes share their bandwidth, ODIN reduces bandwidth waste effectively. Thus, the packet processor module in ODIN minimizes Tor bandwidth by piggybacking on packets already destined for the same final server. As shown in Figure 1, the packet processor module removes the headers (e.g., Ethernet) of two paired packets, and subsequently, it combines the remaining contents of both packets with a virtual header. The total size of an n-th obfuscated packet P'_n is defined as $P'_n = V + P_{r,n} + P_{r+1,n}$, where $P_{r,n}$ denotes the size of n-th packet in the r-th queue and V denotes the size of the virtual header. This approach effectively obfuscates the traffic pattern by combining two distinct flows into a single flow. After combining, the resulting packets are encapsulated, encrypted, and sent to the destination server.

Upon arrival at the receiving server, the combined packets are first decrypted. Subsequently, utilizing the data embedded within the decrypted virtual header, the receiver program reassembles them back into their original form. This reconstitution of packets is performed prior to the initiation of kernel network stack operations, thereby guaranteeing that the Tor service on the destination server receives the packets irrespective of the obfuscation process conducted by ODIN. The overall bandwidth waste is significantly reduced compared to existing padding-based methods, making ODIN a practical solution in environments where efficient utilization of limited shared bandwidth is essential.

Spatial and Temporal Obfuscation. ODIN operates with obfuscation functionalities for both size and timing under all circumstances. To this end, we consider two scenarios: *(i)* when only a single packet is in the queue or remains after pairs of packets have been processed within the time threshold; and *(ii)* when the combined size of two packets exceeds the MTU size.

To address these scenarios, the packet processor module stores the sizes of previously transmitted packets. Utilizing this information, it calculates the optimal padding size for the remaining packet as follows:

$$padding \leftarrow \textsc{CreateRandomData}(MTU - P_{r,n})$$
$$padingSize \leftarrow \textsc{GetSimilarSize}(L, P_{r,n}, padding)$$

, where MTU is the maximum transmission unit, $P_{r,n}$ is the size of an n-th packet in the r-th queue, and L is the size list of previously transmitted packets. Then, the packet processor module identifies a size that is most similar to those within the list of previously transmitted packets. This selected size is then used to create a packet that is sent to the server, thereby preventing potential adversaries from discerning any unique packet characteristics. This approach guarantees size and timing obfuscation under all conditions while preserving the system's effectiveness.

4. Evaluation

We conduct a preliminary experiment on our private Tor testbed, comprising three Tor container nodes (i.e., entry, middle, and exit) running on two physical machines with several services (e.g., video streaming and file downloading). The evaluation of ODIN primarily centers on two key aspects: (i) its efficacy in obfuscating Tor traffic and (ii) its ability to impose lower overhead compared to an existing obfuscation solution.

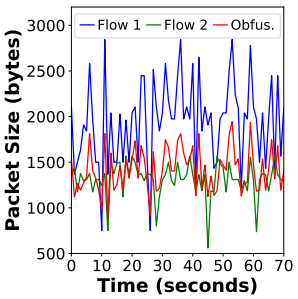

Figure 2: Traffic pattern before/after deploying ODIN.

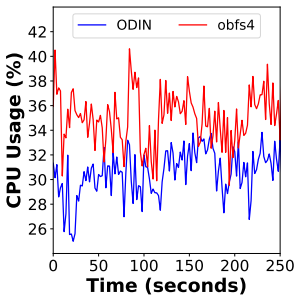

Figure 3: CPU usage of ODIN and obfs4.

Figure 2 exhibits distinct flow patterns with and without ODIN's obfuscation functionality. ODIN successfully combines two distinct traffic flows, resulting in discernible deviations from the original traffic flow patterns. Figure3 presents the measured CPU usage within a specific time window, comparing the deployment of ODIN and obfs4 for obfuscation purposes. On average, ODIN exhibits a 5% reduction in CPU overhead compared to obfs4.

5. Future Work

In future work, we will assess the resilience of ODIN-obfuscated traffic against advanced flow correlation attacks, including those using deep learning techniques.

Acknowledgment

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government. (MSIT) (No. RS-2022-00166401, 2022R1C1C1006967)

References

[1] Tor Exit Enclave. https://help.duckduckgo.com/duckduckgo-help-pages/privacy/tor-exit-enclave/, 2022.

[2] NVIDIA BLUEFIELD-2 DPU, 2023. https://resources.nvidia.com/en-us-accelerated-networking-resource-library/bluefield-2-dpu-datasheet.

[3] Tor Circumvention, 2023. https://tb-manual.torproject.org/circumvention/.

[4] Types of Relays on the Tor Network, 2023. https://community.torproject.org/relay/types-of-relays/.

[5] Kevin P Dyer et al. Peek-a-boo, I Still See You: Why Efficient Traffic Analysis Countermeasures Fail. In *IEEE Symposium on Security and Privacy*. IEEE, 2012.

[6] David Fifield et al. Blocking-resistant Communication Through Domain Fronting. *Proc. Priv. Enhancing Technol.*, 2015(2):46–64, 2015.

[7] Roland Meier et al. ditto: WAN Traffic Obfuscation at Line Rate. In *NDSS Symposium*, 2022.

[8] Steven J Murdoch and George Danezis. Low-cost Traffic Analysis of Tor. In *IEEE Symposium on Security and Privacy*, 2005.

[9] Milad Nasr et al. DeepCorr: Strong Flow Correlation Attacks on Tor Using Deep Learning. In *ACM SIGSAC Conference on Computer and Communications Security*, 2018.

[10] Yixin Sun et al. RAPTOR: Routing Attacks on Privacy in Tor. In *24th USENIX Security Symposium*, 2015.

[11] Matthew K Wright et al. An Analysis of the Degradation of Anonymous Protocols. In *NDSS Symposium*, 2002.