Spying in the Dark: TCP and Tor Traffic Analysis

Yossi Gilad and Amir Herzberg

Department of Computer Science, Bar Ilan University, Israel

mail@yossigilad.com amir.herzberg@gmail.com

Abstract. We show how to exploit side-channels to *identify clients without eavesdropping* on the communication to the server, and without relying on known, distinguishable traffic patterns. We present different attacks, utilizing different side-channels, for two scenarios: a fully off-path attack detecting TCP connections, and an attack detecting Tor connections by eavesdropping only on the clients.
Our attacks exploit three types of side channels: *globally-incrementing IP identifiers*, used by some operating systems, e.g., in Windows; *packet processing delays*, which depend on TCP state; and *bogus-congestion events*, causing impact on TCP's throughput (via TCP's congestion control mechanism). Our attacks can (optionally) also benefit from sequential port allocation, e.g., deployed in Windows and Linux. The attacks are practical - we present results of experiments for all attacks in different network environments and scenarios. We also present countermeasures for these attacks.

1 Introduction

Internet communication is often sensitive, and security measures must be taken to protect privacy against attackers. The exact measures depend on the exact threat; in particular, encryption protocols such as TLS [9] are necessary to protect *content* from an eavesdropping attacker.

However, encryption is insufficient to prevent *traffic analysis*, and in particular, to prevent exposure of the identities of the communicating peers. Users concerned against traffic analysis by eavesdropping attackers, use anonymizing services such as Tor. Other users may simply assume that the adversary is off-path (non-eavesdropping), and expect privacy against such (weaker) attackers.

We present three traffic-analysis attacks against these scenarios. Two attacks identify clients that communicate to a specific server directly over TCP (without anonymizing intermediaries such as Tor). Such attacks do not require eavesdropping at all, and may be launched by weak, off-path attackers, even for commercial motivations. In fact, since the attacks do not involve eavesdropping, they may even be deemed to be *legal* (not wiretapping). We believe technical measures should (and can) prevent such off-path traffic analysis.

Our third traffic analysis attack is against Tor users. It requires eavesdropping abilities only on the client side, and only spoofing abilities on the server side. We

believe that this is an important scenario, since Tor clients are often concerned about attackers which can eavesdrop on their connection to the Tor relay, since the client-relay connection may be insecure (e.g., Internet cafe) or controlled by a potential snoopy organization (employer, government, etc). Our evaluation of this attack is yet incomplete; however, the preliminary results which we present in this paper provide a warning about a new attack vector on the Tor anonymity network.

Our attacks exploit different *side-channels*, providing useful information on a TCP/Tor connection to an off-path attacker (for Tor, attacker can eavesdrop, but only on the client). Side channels have been extensively used in attacks on cryptographic systems, e.g., [22], but also in attacks on privacy, e.g., [13], and more recently also applied to traffic analysis [6, 26, 35].

Our attacks on direct (i.e., non-anonymous) TCP connections can be viewed as instances of an *attack pattern* which we call *Query-Probe-Query*, illustrated in Figure 1; this is a generalization of the well-known *idle (stealth) scan* attack [25, 34], and of the measurement method used in [5]. In the Query-Probe-Query attack pattern illustrated in Figure 1, the first query measures some 'pre-probe state'; the probe may cause some change on the state, where the change depends on the property measured, e.g., whether a specific client port is open; and the final query measures the 'post-probe state'. This attack pattern can be applied with different queries and probes, to measure different values.

In our implementations, each probe is a packet, or few packets, sent to the client $\mathcal{C}$. The probes test for some event e, such that if e holds then $\mathcal{C}$ will send some packet(s), and otherwise he will not send a packet (or send less packets). We use the pre/post probe response to infer on e: we measure the increase in the IP-ID counter, or measure time between receipt of the two response packets (1b. and 3b in Figure 1). Table 1 summarizes our results for traffic analysis on direct TCP connections.

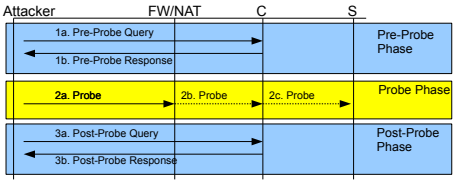


Fig. 1. *Query-Probe-Query* off-path attack pattern. Attacker uses spoofed source address for probe.

	Side Channel	Adversary Location	
		Local	Remote
Success Rate	IP-ID (Section 3)	1	0.92
	Timing (Section 4)	0.74	0.58
Attack Duration (seconds)/ Data Sent (MB)	IP-ID (Section 3)	14/0.6	38/2
	Timing (Section 4)	70/5	50/4

Table 1. Results for probing direct (TCP) connections. Attacker location is relative to the client (victim). Success rates can be improved by repeating the attack.

In the recent years, there is growth in the use of anonymity mechanisms such as *Tor* [10]. Tor is a low latency, circuit based, anonymity network that is widely used and increasing in popularity (according to [1], increase of about 70% in

recent year). Tor is designed to ensure traffic privacy, even when the adversary is able to eavesdrop on either $\mathcal{C}$ or $\mathcal{S}$. However, due to its low latency, Tor cannot ensure traffic privacy against attackers eavesdropping at *both* ends ($\mathcal{C}$ and $\mathcal{S}$), see discussion on related works below.

We show how an adversary capable of eavesdropping on the client, $\mathcal{C}$, but not on the server, $\mathcal{S}$, is able to detect that $\mathcal{C}$ is communicating with $\mathcal{S}$. This attack uses a side channel as well, but does not follow the query-probe-query pattern; here the attacker causes a reduction in the rate of packets that would reach $\mathcal{C}$ in case that he is communicating with $\mathcal{S}$, and then tests whether reduction had occurred.

Our attacks on Tor are *active*, i.e., involve *sending* packets to Tor exit relay. When there is a known, distinct traffic pattern to the communication of specific server (*website fingerprint*), then alternative passive attacks may be applicable as well, e.g., [18, 19, 28]. It may be possible to extend our techniques to also take advantage of such site fingerprint, when available.

1.1 Related works

IP-ID side-channel and off-path traffic analysis. We show how the use of globally-incrementing IP-ID field in IP headers, provides side-channel allowing effective off-path traffic analysis. The use of globally-incrementing IP-ID is recognized, in [17], as a common practice with known security implications; e.g., both globally-incrementing and per-destination incrementing IP-ID allow interception, injection and discarding of fragmented traffic [15]. Globally-incrementing IP-ID can allow estimation of the number of packets sent [32], stealth-scan for open ports (idle scan) [31] and counting hosts behind NAT [5].

The technique that we present for the case of a client that uses a globally-incrementing IP ID and is not connected via a firewall/NAT (see Figure 1) is a variant of idle-scan [25, 34]. The difference is that while idle-scan probes a server for an open (i.e., 'listening') port, our attack probes a client for a connection with a server.

The only other previous work we found that performs traffic analysis by off-path attacker, using a side-channel, is [19]. Their attack is based on detecting changes in the round trip delays from the attacker to the DSL router; this is a rather crude side channel, much less efficient than both the IP-ID and the timing side channels we use. Indeed, they only present detection of whether a client is browsing or playing a video, based on the significant difference in bandwidth, and assuming no other traffic. Our results dramatically improve the impact of detection compared to theirs, provided that the attacker can communicate with the clients.

Tor traffic analysis. Low-latency anonymity networks are known to be vulnerable to traffic correlation attacks by an attacker that eavesdrops on both ends; this problem, and possible countermeasures, was studied in several works, e.g., [7, 24, 36]; efficiency and accuracy can significantly improve if attacker can also

manipulate traffic at the exit relay, see [30]. Indeed, Tor designers are well aware of its inability to properly protect against an attacker (eavesdropping) at both ends of the circuit.

Other attacks manipulate the traffic at the server or the last (exit) relay in the circuit, and use different techniques to detect the relay along the path based on delays [6, 12, 27]. These works assume that the attacker controls the server or the exit relay, but do not require client-side eavesdropping. In contrast, our attack on Tor requires client-side eavesdropping, but does not require control over the server or exit relay. An obvious challenge is to combine the results, and identify clients without controlling server or exit relay, and without eavesdropping at all. Our traffic detection attacks on TCP may be applicable.

1.2 Our contributions

The main contribution of this paper is identification and analysis of side channels in the TCP/IP suite and their practical implications on privacy, as we verify in experiments. We provide practical countermeasures to the problems that we identify, these allow quick patching at the firewall level and require no changes to hosts or core operating system services.

This work motivates use of cryptography in lower network layers and in particular IPsec [20] as we show that higher network layer solutions such as SSL/TLS do not prevent blind traffic analysis.

1.3 Paper Organization

In Section 2 we present our attacker models and the scenarios that we consider, we also present the criteria we use to measure the effectiveness of the attacks. Sections 3, 4 present the global-ID and timing side channels; both sections provide results of empirical experiments. Section 5 presents our attack on Tor and corresponding experiments. Section 6 presents practical defenses. Finally, Section 7 presents our conclusions from this work, as well as future research directions.

2 Model

Let $\mathcal{C}$ and $\mathcal{S}$ be communicating TCP client and server (respectively). We consider two types of adversaries, depending on how $\mathcal{C}$ and $\mathcal{S}$ are connected. In Sections 3 and 4, we consider the case that $\mathcal{C}$ and $\mathcal{S}$ have a direct TCP connection. In Section 5, $\mathcal{C}$ connects to $\mathcal{S}$ through the onion routing anonymity network, Tor [10]; i.e., $\mathcal{C}$ communicates with $\mathcal{S}$ via a circuit of relays (proxies). The goal of the attacker is to identify clients who connect to a server $\mathcal{S}$. We identify $\mathcal{S}$ using its IP address and port.

We consider two types of attackers: Mallory, an *off-path adversary*, and Eve, an *eavesdropping adversary*. The attackers can send *spoofed packets*, i.e., packets with fake (spoofed) sender IP address. Due to ingress filtering [4, 14, 21] and other anti-spoofing measures, IP spoofing is less commonly available than before,

but still feasible, see [2, 11]. Apparently, there is still a significant number of ISPs that do not perform ingress filtering for their clients (especially to multihomed customers). Furthermore, with the growing concern of cyberwarfare and cybercrime, some ISPs may intentionally support spoofing. Hence, it is still reasonable to assume spoofing ability.

We describe both adversary models in Sections 2.1 and 2.2 below. Section 2.3 presents the criteria we use to evaluate the attacks we present.

2.1 Mallory - Off-path Adversary

We assume that $\mathcal{C}$ visits a website that Mallory controls, denoted www.mallory.com. Mallory uses this (legitimate) connection, to probe whether $\mathcal{C}$ has any connections $\mathcal{S}$, see Figure 2.

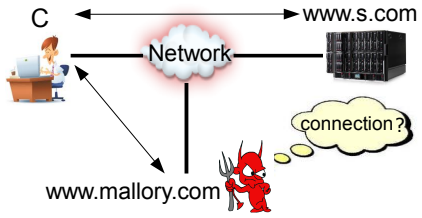


Fig. 2. $\mathcal{C}$ is surfing in both Mallory and $\mathcal{S}$'s sites, Mallory tries to detect whether there is a connection between $\mathcal{C}$ and $\mathcal{S}$.

We consider three variants of Mallory, as illustrated in Figure 3: *with-*$\mathcal{C}$*, near-*$\mathcal{C}$ and *remote*. These differ with respect to Mallory's abilities to communicate with $\mathcal{C}$; the greater the distance, the more likely it is that packet loss or reordering occurs, decreasing the quality of the side channels.

The *with-*$\mathcal{C}$ and *near-*$\mathcal{C}$ attackers are located near the client ($\mathcal{C}$); the difference between them is that the *with-*$\mathcal{C}$ adversary directly communicates with the client, allowing Mallory to take advantage of Windows globally incrementing port allocation (if $\mathcal{C}$ runs Windows). When the adversary and $\mathcal{C}$ communicate via a NAT (near-$\mathcal{C}$ or remote), we assume that the NAT uses per destination incremental assignment of external ports (e.g., as in the widely-used IP-tables NAT/Firewall provided in Linux). See in Section 3 how we exploit different client port allocation techniques. Finally, the *remote* Mallory attacker simulates an adversary that communicates with the clients from a remote location, i.e., via a high latency, jittery and lossy channel.

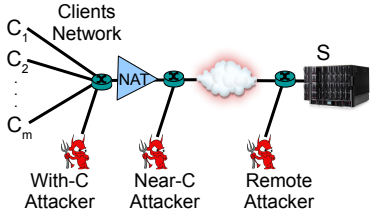


Fig. 3. Three variants of the Mallory adversary.

2.2 Eve - Adversary for Anonymized Connections

In the attacks on Tor, we consider the adversary Eve who is able to eavesdrop on many clients that use Tor, however, Eve cannot eavesdrop on the servers (see Figure 4). Such an adversary may include a government or an employer,

spying on citizens or employees. Eve's goal is to detect which of the clients is communicating (using Tor) with a particular watched/restricted site, $\mathcal{S}$.

2.3 Attack Evaluation Criteria

In addition to measuring the success, false positive and false negative rates, we consider two additional measures. The first measure is the time that an adversary (with some reasonable constant bandwidth) needs to run the attack in order to reach a particular success probability for detecting a connection. This value also provides the minimal detectable connection time. The second measure is the average amount of data per victim that the attacker is required to send to reach a particular success rate.

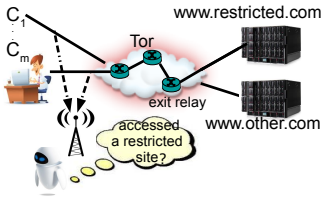


Fig. 4. Eve identifies that some of the clients she eavesdrops on are using Tor and wants to detect which of them is communicating with www.restricted.com. $\mathcal{C}$ (C_m) connects to www.restricted.com via a circuit of 3 relays.

3 Globally-Incrementing Identifier Based Traffic Analysis

This section presents a probing technique that allows an off-path (blind) adversary, Mallory, to identify a connection between a client $\mathcal{C}$ and a server $\mathcal{S}$ when $\mathcal{C}$ uses a globally incrementing IP identifier (IP-ID)[1]. This side channel is only applicable when the TCP connection is over IPv4, since in IPv6 [8] the IP-ID field is only specified in fragmented traffic and TCP packets are rarely fragmented. In the following section we introduce a general technique that does not rely on IP-ID and also applies to IPv6.

A globally-incrementing identifier is not really hidden from Mallory, who can usually learn its value simply by receiving some packet from the victim. A globally incrementing IP identifier is used in all Windows versions we tested (including XP, Vista and 7) and is also the default configuration in FreeBSD; clients running these systems are vulnerable to the attack below. The vast deployment of Windows on client machines (more than 70% according to browser user-agent based surveys, see [33]) makes IP-ID attack vector very practical.

Section 3.1 defines a *port test* that uses the leakage in the IP-ID field to detect whether $\mathcal{C}$ is communicating with $\mathcal{S}$ through a tested port. The test depends on whether $\mathcal{C}$ is connected to the network through a NAT or a stateful firewall that keeps track of existing connections; the test used when $\mathcal{C}$ is connected through a NAT/firewall device the attack is a bit simpler. We believe that this is the more common scenario, since recent versions of Windows (XP SP2 and later) ship with a built in (stateful) firewall that is enabled by default, and furthermore, use of

[1] $\mathcal{S}$'s IP-ID implementation does not influence the probing technique.

NAT devices in small local area networks connecting clients to the Internet is common. Due to space limitations we describe only this test and include the test for the complementary scenario (no firewall/NAT) in an online technical report [16].

In Section 3.2 we describe how Mallory can identify a relatively small set of client ports to test for a connection with $\mathcal{S}$; Mallory performs the port-test for all of them. Section 3.3 presents our experimental setup and empirical results.

3.1 Port-Test for a Client Behind a Firewall/NAT

According to the TCP specification [29] (Section 3.9, bottom of page 69), the first check that a recipient conducts on an incoming packet, in case it belongs to an established connection, validates that the sequence number is within the congestion window. If this check finds the packet invalid, then the recipient discards the packet and sends a duplicate Ack feedback. A stateful firewall or NAT device connecting $\mathcal{C}$ to the network keeps track of existing connections and processes all incoming packets before they reach $\mathcal{C}$. We use the following observation: incoming packets that do not belong to an established connection will be discarded before reaching $\mathcal{C}$ (by firewall/NAT), whereas packets that belong to an existing connection, but specify arbitrary (probably invalid) sequence numbers will reach $\mathcal{C}$ who replies with a duplicate Ack.

The port test for the case of firewall/NAT deploying client is according to the general query-probe-query pattern. The probe specifies $\mathcal{S}$'s address and port as source (i.e., probe is spoofed) and $\mathcal{C}$'s address as destination, Mallory specifies a different destination port in each test. Figure 5 illustrates two iterations of the port test: in the first iteration, the firewall/NAT blocks the probe packet (i.e., no connection through the tested port). In the second iteration, the probe specifies existing connection parameters (IP addresses and ports) and therefore reaches $\mathcal{C}$ who processes the probe and sends a duplicate Ack to $\mathcal{S}$.

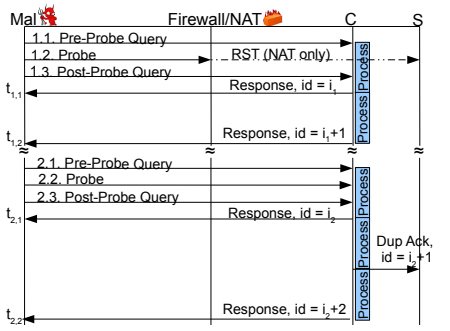


Fig. 5. Two iterations of port test.

Notice that since the probe packet appears to be from $\mathcal{S}$ (in case it specifies a valid 4-tuple), it is difficult to block the probe in firewalls without blocking the legitimate connection that $\mathcal{C}$ has with $\mathcal{S}$.

When $\mathcal{C}$ uses a global identifier, the difference in the IP-ID field in $\mathcal{C}$'s responses to Mallory indicates whether $\mathcal{C}$ had sent a packet in response to the probe (duplicate Ack). If Mallory identifies that $\mathcal{C}$ had sent a packet, then it is likely that $\mathcal{C}$ is communicating with $\mathcal{S}$ via the tested port; however, the identifier may have increased since $\mathcal{C}$ had sent an independent packet to some other peer.

Repeating this test several times allows Mallory to efficiently detect whether $\mathcal{C}$ is connected to $\mathcal{S}$ and reveal $\mathcal{C}$'s, see empirical evaluation below.

We keep a 'score' for each possible port, and increment a specific port's score by 1 point for every test that seems to indicate that there is a connection through that port. We conduct $r > 1$ rounds of the attack, where each port is tested. Finally, we decide that there is a connection if there is a port with a score higher than a threshold, *TH*.

Some firewalls have an option to randomize the IP-ID; our tests would, of course, fail if the packets pass through such randomizing firewall. The attack we describe in the following section applies even in this scenario (but is less accurate).

Implementing Test Queries/Responses. Our attacks use packets that Mallory receives from $\mathcal{C}$ to learn the effect of the (spoofed) probe packet. Mallory can cause $\mathcal{C}$ to send her such packets by using the legitimate TCP connection that she has with $\mathcal{C}$: a query is some short data packet that Mallory sends to $\mathcal{C}$, the response is the $\mathcal{C}$'s acknowledgment sent back to Mallory. This allows Mallory to bypass typical firewall defenses (e.g., Windows), since all packets in the test appear to belong to legitimate connections (requests to $\mathcal{C}$-Mallory connection, probe to $\mathcal{C}$-$\mathcal{S}$ connection). See further details in the technical report [16].

3.2 Improving the Search: Client Port Allocation Algorithms

The port test that we presented allows Mallory to test whether the client has a connection to some server via a specific port. There are 2^{16} possible ports that $\mathcal{C}$ might use to communicate with $\mathcal{S}$. However, common client port allocation paradigms allow more efficient attacks.

Below we present two common paradigms and methods to reduce the number of tests for each of them.

Globally Incrementing: the client port is incremented for every new connection (initialized to a random value) Algorithm 1 in [23] describes the implementation. This approach is used in Windows and FreeBSD. If $\mathcal{C}$ uses this port allocation paradigm, then recent connections that the client forms are likely to use 'close' ports to that $\mathcal{C}$ uses in the connection with Mallory. Hence, Mallory can test only these ports.

Per-Server Incrementing: the client port is incremented for every new connection with the server. Connections to different servers use different counters. This approach is used in Linux; Algorithm 3 in [23] describes the implementation. The previous 'trick' we presented does not work in this case since the port that $\mathcal{C}$ uses for the connection with Mallory does not correlate to that $\mathcal{C}$ uses to communicate with $\mathcal{S}$. However, we can still use the counter property of this paradigm: Mallory causes $\mathcal{C}$ to create x 'dummy' connections to $\mathcal{S}$ (we explain how below); since these connections all share the same counter, they are sequential. Hence, Mallory can test every port $y = 0 \pmod{x}$ and identify p, a port $\mathcal{C}$ that uses to communicate with $\mathcal{S}$. Next, Mallory checks all ports in the interval

$[p-x, p+x]$ and checks whether there are at least $x+1$ connection ports. If yes, then $\mathcal{C}$ has an 'independent' connection with $\mathcal{S}$. In this method, the attacker would test roughly $\frac{2^{16}}{x}$ different ports.

In is left to describe how Mallory causes $\mathcal{C}$ to establish multiple connections with $\mathcal{S}$. Since $\mathcal{C}$ is in Mallory's site, she can run a script (in the browser sandbox) on $\mathcal{C}$. This script, while very limited, can open connections with other servers to dynamically embed remote objects. We use it to open connections to www1.mallory.com,. . . ,wwwx.mallory.com which are domains that Mallory controls. Since Mallory controls the DNS records for these domains, she sets each of these records to point to the same IP, that of $\mathcal{S}$. Browsers open a new connection for each domain (regardless of its IP address); hence, this technique, which we verified on Internet Explorer, Firefox and Chrome, opens x new connections to $\mathcal{S}$.

The typical limitation of x is the number of connections that a browser can have simultaneously; this limitation is typically one or few dozens; e.g., 16 in Firefox. In our experiments below and in the following section, we use $x = 10$.

3.3 Empirical Evaluation

Setup. In our empirical evaluation, the client network is a class C subnet that has 5 clients running Windows 7, each of them sends on average 64 packets per second to other peers in the subnet (these packets are short, to simulate clients that usually send Ack packets or short requests). Mallory probes one of the clients in the network, $\mathcal{C}$, who connects to her (malicious) website. Mallory's bandwidth is limited to 10 mbps. We used the network topology illustrated in Figure 3, network nodes are connected through switch devices. The NAT device in the network topology is a Linux machine (kernel version 2.6.35) running IP-tables (version 1.4.4). The server machine runs Linux (kernel version 2.6.35) and uses an Apache web-server (version 2.2.14). When we evaluate the attack for the 'Remote Attacker' scenario, the adversary communicates with the clients via a traffic shaper that induces high latency (200ms), significant loss probability (0.5%) and jitter (1-10 milliseconds).

Evaluation. We first evaluated the attack in case that $\mathcal{C}$ is communicating with $\mathcal{S}$. We compared between the score of the 'connection port' (i.e., port that $\mathcal{C}$ uses for the connection) to that of the best appearing non-connection port (i.e., port with the highest score that is not the connection port) in each round (repetition of the attack, see discussion above); note that the highest scoring non-connection port may change between rounds.

Figure 6 shows results for near-$\mathcal{C}$ and remote attackers. In both environments, the score of the connection port was well above 50% of the maximal score, certainly after five or more reounds; hence, for efficiency, we can continue testing only 'high scoring' ports in advanced rounds. Namely, a port is tested in the next round only if its current score is above 50% of the maximal possible score.

We implemented an adversarial website that presents its clients a request to arbitrarily decide whether to connect ('surf') to a third-party website, $\mathcal{S}$; our website attempted to detect the clients' choice. We used an automatic client, $\mathcal{C}$, that chooses to connect to $\mathcal{S}$ with probability $\frac{1}{2}$ and implemented the port-test above.

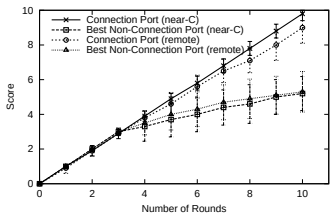


Fig. 6. Global-ID attack. Comparison of a connection port to that of the highest scoring non connection port as a function of round number. Each measurement is an average of 10 runs, error-bars mark the standard deviation values.

The choice of whether there is a connection between $\mathcal{C}$ and $\mathcal{S}$ is according to a threshold over the final score of the ports. Namely, if there exists a port with a score over this threshold, then we identify that there is a connection. Figure 6 shows that a choice of 70% of the maximal possible score as a threshold provides a good seperation between the connection port (in case it exists) and other ports. Figures 7 - 9 show the success rate in detecting whether $\mathcal{C}$ communicates with $\mathcal{S}$ for different adversary locations as a function of the duration of the attack. Figure 10 compares the average amount of data that Mallory sends (per victim) to reach different success rates and for different locations in the network.

In Figures 7 - 10, the measurements are the average of 50 runs; error-bars mark the standard deviation values (for readability, not all measurements specify the error bars). Note that the thresholds that we have used in our evaluation may not work as well in other scenarios, e.g., when the client sends much more than 64 packets per second the thresholds should be higher.

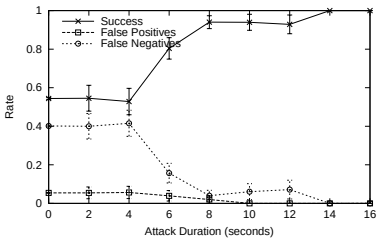


Fig. 7. Global-ID attack, with-$\mathcal{C}$ attacker.

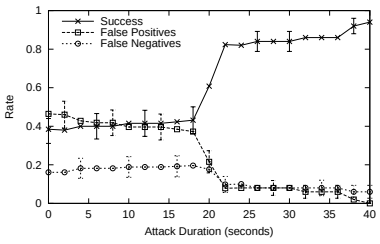


Fig. 8. Global-ID attack, near-$\mathcal{C}$ attacker.

4 Time-Based Traffic Analysis

The globally incrementing IP-ID side channel, presented in Section 3, exploits an operating system flaw. In this section we explore a more generic, timing based, side channel that is applicable when $\mathcal{C}$ is behind a firewall or a NAT. We define below a new port test which resembles the IP-ID based port test and is illustrated in Figure 5 as well.

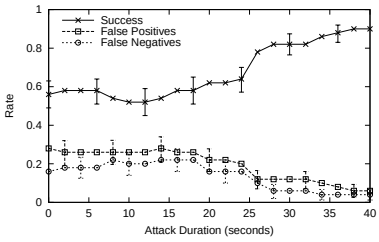


Fig. 9. Global-ID attack, remote attacker.

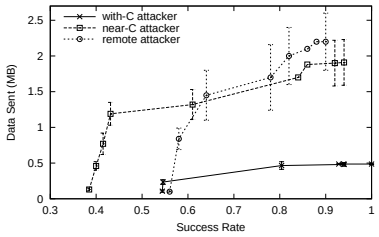


Fig. 10. Amount of data Mallory sends as a function of her success rate.

The timing attack is based on the following observation: if $\mathcal{C}$ is protected by a firewall or connects through a NAT device, then in case that Mallory tests the correct port, $\mathcal{C}$ sends an additional packet to $\mathcal{S}$ (response to the probe); this delays processing of following packets, and in particular the post-probe query; see illustration in Figure 5. We use this delay to identify the connection.

4.1 Timing-Based Port Test

A significant challenge is the jitter in the network, i.e., latencies may vary while testing different ports. Thus, identifying the longest time difference between two responses and testing whether it is over a threshold is likely to produce an incorrect result. We cope with this challenge by *relatively comparing* ports: we assign each port to a small group of s arbitrary ports.

Ports in each group are tested one after the other; we assume that jitter does not vary much during the short time interval of testing a specific (small) group. After testing a group, each port is assigned with a relative rank according to the time difference between responses in the corresponding port-test; the lower the (group-relative) rank, the greater the time difference and the more likely is a connection through that port. We conduct several rounds of this attack (to reduce the probability of errors).

Similarly to the attack presented in the previous section, we keep a score for each port and after each round increase a port's score according to its rank: denote by σ_i the number of points that a port gains if it has rank i within the group, these weights are normalized; i.e., $\sum_{i=1}^{s} \sigma_i = 1$, and for every $i < j$, $\sigma_i \geq \sigma_j$. The values of s and the vector $\sigma = (\sigma_1, \cdots, \sigma_s)$ depend on the channel between Mallory and $\mathcal{C}$. We employ a machine learning approach (genetic algorithm) to learn appropriate value for the vector σ; see details of the algorithm in [16]. Let μ, μ' denote the expected scores of connection and non connection ports respectively. The target function of the learning algorithm is to maximize $\mu - \mu'$. In our empirical evaluation below we explain how Mallory obtains measurements of μ, μ' for different values of s, σ.

4.2 Empirical Evaluation

The environment we used to evaluate the timing attack is as described in Section 3.3, except that the client machines run Linux (kernel version 2.6.35) instead of Windows; hence, the attacker cannot employ the global IP-ID based attack. All Linux distributions ship with IP-tables firewall, its rule-set is empty by default; we therefore evaluated only the scenarios where Mallory is near-$\mathcal{C}$ remote (see Figure 3), i.e., Mallory communicates with $\mathcal{C}$ and $\mathcal{S}$ via a NAT device.

The first task is to obtain a good estimation of the optimal values of s, σ for the channel between $\mathcal{C}$ and Mallory (this depends on Mallory relative location to $\mathcal{C}$). The machine learning algorithm we employ uses the connection that Mallory has with $\mathcal{C}$ (see Figure 2): since for this connection Mallory knows the client's port, he is able to obtain measurements for different group sizes (s) and weights (the vector σ), see more details in [16]. We found that these values significantly differ between the two attacker locations; e.g., in our setup we found $s = 31$ to be suitable for a near-$\mathcal{C}$ attacker while $s = 4$ appeared optimal for the remote attacker. Figure 11 compares the connection-port score to that of the highest scoring non-connection port as a function of the number of rounds.

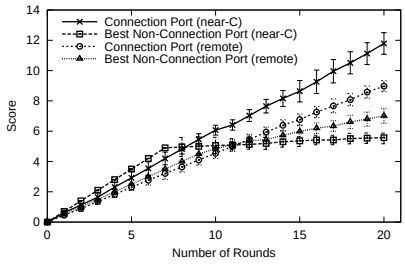


Fig. 11. Timing Attack. Comparison of a connection port to that of the highest scoring non connection port as a function of round number. Average of 10 runs, error-bars mark the standard deviation values.

Next, we derive two thresholds for promoting ports to following rounds according to their current score, this is similar to the experiments in Section 3.3. According to the training set results, a choice of 60% of the maximal score for the near-$\mathcal{C}$ attacker scenario and 40% in for the Interent attacker scenario appear to be reasonable. As in Section 3.3, these thresholds require further research for other scenarios, e.g., thresholds are effected by the victim's transmission rate.

We implemented the timing attack and conducted an experiment similar to that presented in Section 3.3. We set the threshold for deciding whether a connection exists between $\mathcal{C}$ and $\mathcal{S}$ according to the difference between the expected scores of a connection port (μ) and a non connection port (μ') as derived from our training measurements. See analysis in [16]; in this experiment we set the threshold to $0.2\mu' + 0.8\mu$. We measured our success rate in probing whether $\mathcal{C}$ is communicating with a (third-party) website, $\mathcal{S}$. Results are in Figures 12 - 14.

Comparing these results to those of the ID based attack, more time is required to obtain similar success rates, and the maximal success rates reached are lower. However, the results show that the timing attack does provide information on the connection (since success ratio is greater than 0.5); but its hint is often

misleading (since success ratio is significantly less than 1). Attacker can repeat the attack several times and select by the majority.

Figure 15 illustrates the average amount of data that Mallory needs to send in order to reach a particular success rate for different locations in the network and number of probes in each test.

In Figures 12 - 15, the measurements are the average of 50 runs; error-bars mark the standard deviation values.

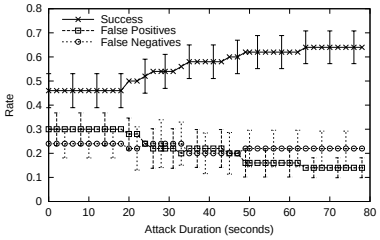


Fig. 12. Timing attack, near-$\mathcal{C}$ attacker. Mallory sends 2 probes in each test.

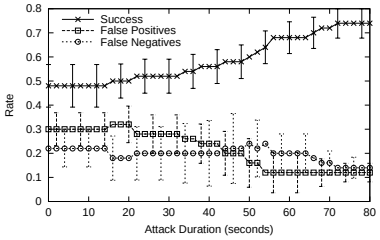


Fig. 13. Timing attack, near-$\mathcal{C}$ attacker. Mallory sends 5 probes in each test.

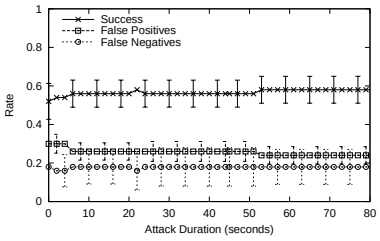


Fig. 14. Timing attack, remote attacker. Mallory sends 5 probes in each test.

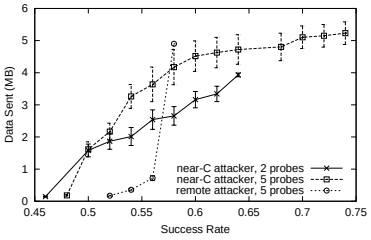


Fig. 15. Amount of data Mallory sends as a function of her success rate.

5 Traffic Analysis for Tor Clients

In this section we consider the second scenario presented in Section 2, where $\mathcal{C}$ uses an onion routing infrastructure to connect to $\mathcal{S}$. We focus on the popular Tor network, but similar attacks may apply to other low latency anonymity networks. In this section we assume that the attacker, Eve, is able to eavesdrop on $\mathcal{C}$ (but not on $\mathcal{S}$).

Here, Eve actively interferes in the possible connection between $\mathcal{C}$ and $\mathcal{S}$ and then tests whether a change in the rate of packets that $\mathcal{C}$ receives occurred. If the result is positive, then it is an indication that $\mathcal{C}$ communicates with $\mathcal{S}$. As of writing this version of the paper, we only did preliminary testing of this attack; more work is required to evaluate the practicality of this attack.

A Tor client connects to a remote server via a chain of relays (proxies). The last relay in the chain, i.e., the exit relay, has a direct TCP connection with the server. The number of possible Tor exit relays is important for our attacks (since a direct connection exists between the exit relay and the server); the Tor network comprises of few thousand relays, about one thousand of which can perform as exit relays (see [1]). However the number of different exit relays that a client is *likely* to use is significantly lower: first, a client can only use online relays; second, Tor clients typically choose the exit relays according to various parameters such as stability and bandwidth. We have formed Tor circuits from two clients in different geographic locations and kept track on the exit relay that was used. The measurements show that 20% of the 2000 circuits which we created (using Tor client version 0.2.2.35) had one of 7 specific exit relays. Figure 16 illustrates our measurements.

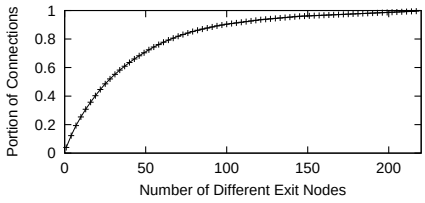


Fig. 16. The portion of 2000 circuits we created using the Tor client as a function of the number of different exit relays used.

5.1 The Indirect Rate Reduction Attack

In this section we present an attack that uses the following observation: if Eve influences the rate of communication between $\mathcal{S}$ and the exit relay, then this, in turn, will change the rate of the connection between $\mathcal{C}$ and the first (entrance) relay. Eve sees the latter connection and is able to detect the change.

Since Eve can only observe the aggregated rate of data that $\mathcal{C}$ receives from the entrance relay (since communication is encapsulated), this attack vector weakens when $\mathcal{C}$ communicates with several other servers via one Tor circuit and $\mathcal{C}$'s connection rate with $\mathcal{S}$ is relatively small to that of the other servers.

The following attack uses TCP congestion control mechanisms to fake congestion events; hence, reducing the communication rate. This attack is based on the insight previously noted in Section 3.1: by sending a (spoofed) packet to an exit relay, Eve would cause that relay to immediately send a duplicate acknowledgment (Ack) in response to $\mathcal{S}$, as long as Eve's packet appears from an existing connection between the exit relay and $\mathcal{S}$. The duplicate Ack that the exit relay sends to $\mathcal{S}$ in response, has a valid sequence number and $\mathcal{S}$ will accept it. A sequence of three duplicate Acks is interpreted by TCP as a congestion event, see [3]; when it occurs, $\mathcal{S}$' congestion window shrinks. The exact effect depends on the TCP implementation that the server runs. Until recently, TCP Reno variant was default in Linux (the common operating system of server machines); for this variant each congestion event halves the size of the congestion window. Recent Linux kernels use the TCP Cubic variant, where the TCP window size is multiplied by a constant of 0.8 for each congestion event.

The congestion window size directly effects the sender's ($\mathcal{S}$) transmission rate: $\mathcal{S}$ only sends as much as the congestion window allows. Thus, by causing the exit relay to send a sequence of 3 duplicate Acks to the server, Eve causes the latter to significantly reduce its 'sending' rate. This attack is illustrated in Figure 17, which shows the effect when Eve sends the spoofed packets to an exit relay and port through which there is a connection with $\mathcal{S}$.

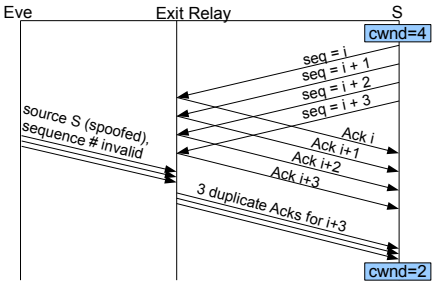


Fig. 17. Eve causes the exit relay to send 3 duplicate Acks to $\mathcal{S}$. $\mathcal{S}$' congestion window is halved as a result.

Attack Process. We use the asymmetry in the distribution of client choice for exit relays to reduce the number of packets that the attacker needs to send to perform indirect rate reduction. Namely, while there are many exit relays available, there are only few 'likely' exit relays that a client might use (see discussion above and Figure 16). For every server IP address s and likely exit relay x, Eve can optionally employ one of the attacks in the Sections 3 and 4 to identify those exit relays that communicate with the server. This optional step will reduce the effort in the following steps of the attack. The techniques in Sections 3 and 4 do not only identify the existence of a connection between two peers, but also identify the client port – if a connection exists, then this is the port with the highest score; see details on how Eve employs these techniques in [16]. Next, for each of the 'suspected' connections, she performs the indirect rate reduction attack described above and checks which of the clients had experienced 'rate reduction'. This process repeats several times for statistical coherency; after each iteration the attack is suspended to allow $\mathcal{S}$'s congestion window to recover.

An important property of this attack is that the spoofed packets that Eve sends to the exit relay in order to reduce the server's rate, are not client specific. Hence, in case that Eve eavesdrops on multiple clients (e.g., a government spying on its citizens) this attack would simultaneously check which of these possible clients has a connection with $\mathcal{S}$.

Characteristics of vulnerable connections. Since the attack repeats for several iterations with intermediate suspensions, this attack requires connections lasting several minutes (see evaluation below). Furthermore, the connection must be 'active', i.e., the server should send data to the client while the attack takes place; this allows Eve to detect rate reductions and allows the congestion window to recover when the attack is suspended. These type of connections include, for example, file transfers (over FTP or HTTP).

5.2 Analysis

Our analysis in this subsection assumes that Eve does not try to detect a direct connection between the exit relays and the server $\mathcal{S}$ (the optional step). Instead, she performs the indirect rate reduction attack on every likely exit relay and all possible ports.

When using Tor, clients connect to $\mathcal{S}$ via proxies; therefore, clients' geographic location does not hint Eve on the server IP address that they will connect to (in case $\mathcal{S}$ has multiple physical servers, e.g., for load balancing). As a result, Eve must enumerate all the IP addresses of $\mathcal{S}$ during the attack.

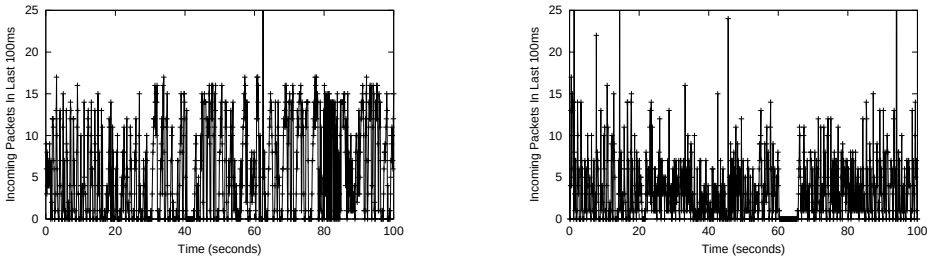
For each of the n_s server addresses and for every exit relay that Eve tries, she performs 2^{16} iterations, trying a different port in each iteration; for each port she sends three packets that would cause the exit relay to send three duplicate Acks to the server, if a connection exists through that port. These packets can be short, with only one byte of data, i.e., 41 bytes long. Hence, the overall data that Eve sends to a particular exit relay, using a particular source IP of $\mathcal{S}$ in a single attack is $2^{16} \cdot 3 \cdot 41 < 7.7$MB. As shown in Figure 16, a small set of exit relays allows a good 'hit' rate. If Eve enumerates on all n_s possible server addresses and the most likely seven exit relays, then by our measurements the attack results in a 'hit' rate of about $\frac{1}{5}$ (see Figure 16); in this attack, she sends $53 \cdot n_s$ MB in each round. As noted at the end of the previous subsection, Eve's effort is divided on the number of clients (victims) that she probes.

5.3 Empirical Evaluation

Setup. We used the Tor network to evaluate the indirect rate reduction attack. To simplify the experiment and limit the effect on other Tor users, we performed the following measures: the restricted web-site server, a Linux machine (kernel version 2.6.35) which runs an Apache web-server (version 2.2.14), had only one IP address; furthermore, when running the attack, Eve was aware of the exit relay that is used and its port used for the connection. Given these three parameters, Eve only sends 3 packets of 41 bytes, i.e., 123 bytes, to carry out a single rate reduction iteration. Below, we describe the frequency of iterations and show that we send about 0.5 KB per second; we believe that this did not load the exit relay or caused damage for other Tor users. The client machine in our experiments runs Windows 7 and uses Tor (version 0.2.1.30) to connect to web-servers. While running our evaluation, we created Tor circuits using 12 different exit relays.

Evaluation. First, we observed the effect of the rate reduction attack (three duplicate Ack technique). To measure this effect, $\mathcal{C}$ connects to one of our servers through Tor; our server reports to Eve the IP address and port of the exit relay. Eve sends her packets only to the reported exit relay and only to the specific port used in the connection with our server. Eve performs three iterations of rate reduction every second, aiming to fake three congestion events and decrease the congestion window to about half of its size (in case of cubic variant). This implies that in every second, Eve sends 369 bytes to the exit relay. In Figure

18 we compare between the rate of packets that the client receives (as observed by Eve) on normal conditions and when Eve attacks the exit relay; our attack reduces the average rate.



(a) Normal Conditions (b) Under Attack

Fig. 18. Comparison between a rate of a TCP connection (via Tor) in normal conditions and when under rate reduction attack.

We next tested the scenario considered in Section 2, i.e., of a client that connects through Tor to one of two sites. Eve uses rate reduction to test whether the client is communicating with the restricted site. We conducted the experiment as follows: the victim $\mathcal{C}$ connects to one of two servers in each time, each server is chosen with probability $\frac{1}{2}$. Regardless of the choice that the client makes, the 'restricted' server sends Eve an IP address and port, allegedly describing the exit relay connected to it. In case that the client does not connect to the restricted server, these values specify an arbitrary exit relay and port. Eve then employs the attack above, performing three rate reductions per second and sending a total of 369 Bps to the specified exit relay.

If client rate decreased by at least 20% during the last 30 seconds, then the client's score is incremented. The 20% threshold is motivated by the results in Figure 18, but may change in other scenarios, e.g., for a different server. This process is repeated; between iterations there is a 30 seconds suspension that allows the TCP connection between the server and exit relay to recover to its normal rate (in case the connection exists) and allows Eve to obtain a recent measurement of the average rate in $\mathcal{C}$'s connection. Eventually, Eve decides that $\mathcal{C}$ is communicating with the restricted site if $\mathcal{C}$ has more than half of the possible points. Figure 19 shows Eve's success rate as a function of the duration of the attack. In these experiments, the servers run TCP Cubic variant; an improvement in success rate is observed when server runs TCP Reno.

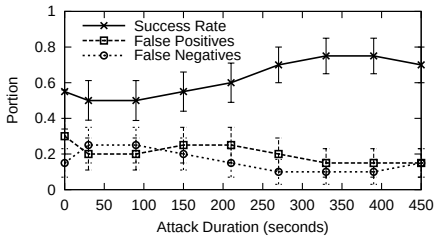


Fig. 19. Eve's success rate in detecting client access to a restricted site via Tor. Each measurement is the average of 20 runs. Server runs TCP cubic variant.

6 Defense Mechanisms

The countermeasures that we propose in this section do not completely eliminate the related side channel threat, however, they make it more difficult to exploit. These defenses are suitable for deployment on firewalls to ease deployment.

The globally incrementing IP identifier side channel, as mentioned in Section 3, is only relevant while still using IPv4. One way to avoid it is to use random IP-ID values; however, this can result in collisions and loss for fragmented traffic. The attack in Section 3 can be prevented by simply moving from globally-incrementing IP-ID to per-destination IP-ID; this would preferably be done by hosts, but until hosts do so[2], a firewall can implement this by adding (pseudo)random per-destination offset to the IP-ID. See analysis and better ways to fix the IP-ID in [15, 17].

It is more challenging to block or reduce the timing side channels and cope with the rate reduction attack presented in Section 5. The flaw that we identify is that a blind adversary is able to cause a TCP recipient an involuntary reaction by sending arbitrary (spoofed) packets. We propose keeping a small window of acceptable sequence numbers that may be processed. This window resembles the receiver's congestion window, but is more aggressive: while packets outside the congestion window cause a duplicate acknowledgment (which we use in the attacks described in Sections 3-5), packets that specify sequence numbers outside the acceptable-window are silently discarded. The acceptable-window is larger than the host's congestion window and includes it. A congestion window is usually up to 2^{16} bytes, an acceptable-window that is twice as large, i.e., 2^{17} bytes, will significantly degrade the attacker's ability to conduct all the attacks in this paper. Since the sequence number is 32 bits long, the attacker is required to send $\frac{2^{32}}{2^{17}} = 2^{15}$ times the number of packets to conduct similar attacks. However, this technique requires that the firewall will inspect the sequence numbers in incoming TCP packets, which increases the packet processing overhead.

7 Conclusions and Future Work

Our primary conclusion is that TCP implementations leak information that allows attackers to study the existence of connections via side channels as we demonstrated in three attacks.

We leave several research directions for future work. Specifically, a more extensive empirical study is required to complete the evaluation of the Indirect Rate Reduction attack on the Tor network. Furthermore, it would be desirable to provide an analytic analysis for the attacks presented in this paper.

An important question is, can we perform a more efficient and more accurate attack on Tor anonymity by combining the indirect rate reduction attack presented in this paper with other existing attacks on Tor anonymity which exploit other attack vectors, e.g., [12, 18, 19, 28].

[2] We informed Microsoft to the IP-ID issues, but we are not aware of intention to fix the IP-ID in Windows.

Acknowledgements

Thanks to Moti Geva, Amit Klein, Roger Dingledine and the anonymous referees for their comments and suggestions.

References

1. Tor Metrics Portal. Network and Usage Graphs. `http://metrics.torproject.org/graphs.html`, November 2011.
2. Advanced Network Architecture Group. ANA Spoofer Project. `http://spoofer.csail.mit.edu/summary.php`, 2012.
3. M. Allman, V. Paxson, and E. Blanton. TCP Congestion Control. RFC 5681 (Draft Standard), September 2009.
4. F. Baker and P. Savola. Ingress Filtering for Multihomed Networks. RFC 3704 (Best Current Practice), March 2004.
5. Steven M. Bellovin. A Technique for Counting Natted Hosts. In *Internet Measurement Workshop*, pages 267–272. ACM, 2002.
6. Sambuddho Chakravarty, Angelos Stavrou, and Angelos D. Keromytis. Traffic Analysis against Low-Latency Anonymity Networks Using Available Bandwidth Estimation. volume 6345, pages 249–267. Springer, 2010.
7. George Danezis. The Traffic Analysis of Continuous-Time Mixes. In *Proceedings of Privacy Enhancing Technologies workshop (PET)*, pages 35–50, 2004.
8. S. Deering and R. Hinden. Internet Protocol, Version 6 (IPv6) Specification. RFC 2460 (Draft Standard), December 1998. Updated by RFCs 5095, 5722, 5871, 6437.
9. T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard), 2008. Updated by RFCs 5746, 5878, 6176.
10. Roger Dingledine, Nick Mathewson, and Paul F. Syverson. Tor: The Second-Generation Onion Router. In *USENIX Security Symposium*, pages 303–320. USENIX, 2004.
11. Toby Ehrenkranz and Jun Li. On the State of IP Spoofing Defense. *ACM Transactions on Internet Technology (TOIT)*, 9(2), 2009.
12. Nathan S. Evans, Roger Dingledine, and Christian Grothoff. A Practical Congestion Attack on Tor Using Long Paths. In *USENIX Security Symposium*, pages 33–50. USENIX Association, 2009.
13. Edward W. Felten and Michael A. Schneider. Timing Attacks on Web Privacy. In Sushil Jajodia, editor, *Proceedings of the 7th ACM Conference on Computer and Communications Security*, pages 25–32, Greece, November 2000. ACM Press.
14. P. Ferguson and D. Senie. Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing. RFC 2827 (Best Current Practice), May 2000. Updated by RFC 3704.
15. Yossi Gilad and Amir Herzberg. Fragmentation Considered Vulnerable: Blindly Intercepting and Discarding Fragments. In *Proceedings of USENIX Workshop on Offensive Technologies*, Aug 2011.
16. Yossi Gilad and Amir Herzberg. Spying in the Dark: TCP and Tor Traffic Analysis - Technical Report. `http://u.cs.biu.ac.il/~herzbea/security/TR/TR12_02`, April 2012.
17. F. Gont. Security Assessment of the Internet Protocol Version 4. RFC 6274 (Informational), July 2011.
18. Andrew Hintz. Fingerprinting websites using traffic analysis. In Roger Dingledine and Paul F. Syverson, editors, *Privacy Enhancing Technologies*, volume 2482 of *Lecture Notes in Computer Science*, pages 171–178. Springer, 2002.

19. Sachin Kadloor, Xun Gong, Negar Kiyavash, T. Tezcan, and Nikita Borisov. Low-Cost Side Channel Remote Traffic Analysis Attack in Packet Networks. In *ICC*, pages 1–5. IEEE, 2010.
20. S. Kent and K. Seo. Security Architecture for the Internet Protocol. RFC 4301 (Proposed Standard), December 2005.
21. T. Killalea. Recommended Internet Service Provider Security Services and Procedures. RFC 3013 (Best Current Practice), November 2000.
22. Paul C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In Neal Koblitz, editor, *Advances in Cryptology – CRYPTO ' 96*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. International Association for Cryptologic Research, Springer-Verlag, Germany, 1996.
23. M. Larsen and F. Gont. Recommendations for Transport-Protocol Port Randomization. RFC 6056 (Best Current Practice), January 2011.
24. Brian N. Levine, Michael K. Reiter, Chenxi Wang, and Matthew K. Wright. Timing Attacks in Low-Latency Mix-Based Systems. In Ari Juels, editor, *Proceedings of Financial Cryptography*, pages 251–265. Springer-Verlag, LNCS 3110, February 2004.
25. Gordon Lyon. *Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning*. `http://nmap.org/book/`, 2009.
26. Prateek Mittal, Ahmed Khurshid, Joshua Juen, Matthew Caesar, and Nikita Borisov. Stealthy Traffic Analysis of Low-Latency Anonymous Communication Using Throughput Fingerprinting. In Yan Chen, George Danezis, and Vitaly Shmatikov, editors, *ACM Conference on Computer and Communications Security*, pages 215–226. ACM, 2011.
27. Steven J. Murdoch and George Danezis. Low-Cost Traffic Analysis of Tor. In *IEEE Symposium on Security and Privacy*, pages 183–195. IEEE Computer Society, 2005.
28. Andriy Panchenko, Lukas Niessen, Andreas Zinnen, and Thomas Engel. Website Fingerprinting in Onion Routing Based Anonymization Networks. In *Proceedings of the 10th annual ACM workshop on Privacy in the electronic society*, WPES '11, pages 103–114, New York, NY, USA, 2011. ACM.
29. J. Postel. Transmission Control Protocol. RFC 793 (Standard), September 1981. Updated by RFCs 1122, 3168, 6093, 6528.
30. Ryan Pries, Wei Yu, Xinwen Fu, and Wei Zhao. A New Replay Attack Against Anonymous Communication Networks. In *IEEE International Conference on Communications (ICC)*, pages 1578–1582, 2008.
31. Salvatore Sanfilippo. A New TCP Scan Method. `http://seclists.org/bugtraq/1998/Dec/79`, 1998.
32. Salvatore Sanfilippo. About the IP Header ID. `http://www.kyuzz.org/antirez/papers/ipid.html`, Dec 1998.
33. Wikipedia. Usage Share of Operating Systems. `http://en.wikipedia.org/wiki/Usage_share_of_operating_systems`, 2011.
34. M. Zalewski. *Silence on the wire: a field guide to passive reconnaissance and indirect attacks*. No Starch Press, 2005.
35. Sebastian Zander and Steven J. Murdoch. An Improved Clock-Skew Measurement Technique for Revealing Hidden Services. In Paul C. van Oorschot, editor, *USENIX Security Symposium*, pages 211–226. USENIX Association, 2008.
36. Zhu, Fu, Graham, Bettati, and Zhao. On Flow Correlation Attacks and Countermeasures in Mix Networks. In *International Workshop on Privacy Enhancing Technologies (PET), LNCS*, volume 4, 2004.