

Countering Repacketization Watermarking Attacks on Tor Network

Dijiang Huang and Utkarsh Agarwal
{dijiang,uagarwal}@asu.edu
Arizona State University

Abstract. Watermarking attacks have drawn a lot of attention against anonymous communication systems in recent years. In this paper, we investigate Tor’s vulnerability against the watermarking attacks. We analyze the behavior of Tor nodes and find that repacketization and the delay introduced due to TCP multiplexing considerably affects the effectiveness of traditional watermarking schemes. However, if an attacker has the knowledge of Tor’s packet processing mechanism, the watermarking attack can be modified to perform traffic analysis attack on Tor effectively. If the packet sizes are manipulated to limit repacketization at Tor nodes, and the timing intervals required for watermarking are greater than the delay introduced at each hop of the Tor node, then watermarking scheme can be deployed successfully. To this end, we implement a packet repacketization approach in the Tor networks to improve the detection rate of watermarks.

To counter the discovered watermarking attacks on Tor, we also present a packet scheduling approach to handle Tor’s circuit buffer access that is involved in Tor’s buffer processing mechanism. We show that if the Tor nodes can “flag” the packets, which are suspected to be in a watermarked flow, the scheme can be defeated. Our solution requires each Tor node to monitor received packet size. This approach is lightweight compared to previous solutions, which require each node to use a watermarking matching algorithm to identify malicious watermarked traffic.

1 Introduction

Timing attacks is one of the most effective traffic analysis attacks against low-latency anonymous communication systems. Watermarking based timing attacks (or watermarking attacks for short) [8] are based on manipulating inter-packet delay to introduce traffic patterns into the network flows between an anonymous client and its destination server. The disturbance of existing Internet delay cannot destroy the introduced delay patterns, which make the watermarking attack very effective for low latency anonymous communication systems.

This is because low-latency anonymous communication systems do not remove packet timing correlation between anonymized flows and original

data flows. As a result, the invariant characteristic of delay pattern of network flows is explored by watermarking attacks to break the anonymity of communication parties. Particularly, in watermarking attacks, inter-packet delays are adjusted in order to embed a traffic pattern in network flows. If this embedded traffic pattern can be uniquely identified at its receiver, then original sender and receiver can be linked. In order to deploy this attack successfully an attacker needs to monitor and perturb network flow from potential senders and receivers. To counter watermarking attacks, naïve solutions can be “adding bogus packets, traffic padding, mixing follows, etc.”, which can dramatically downgrade the performance of corresponding anonymous communication systems.

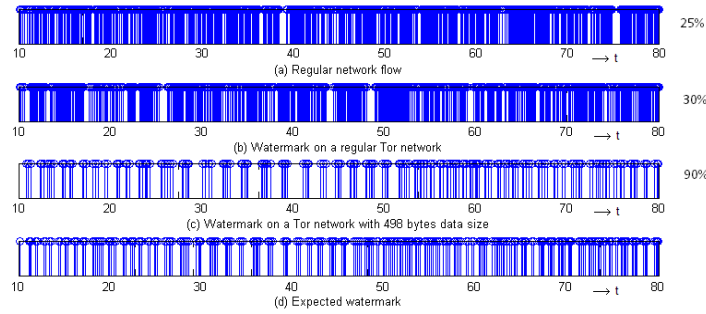


Fig. 1. Different Watermarks

The watermarking attack was successfully deployed and tested on *anonymizer.com* [1]. It required 11 minutes of active browsing to get 100% watermark detection. However, it has not been reported that The Onion Router (Tor) [3] is subjected to watermarking attacks without compromising Tor nodes. Based on our experimental studies on Tor, if the same watermark attacks as presented in [8] are performed on Tor, the watermarking attacks perform poorly and give a detection rate of $\sim 30\%$, which is similar to decoding a non-watermarked flow with $\sim 25\%$ detection rate. Figure 1(a)-(b) show the stem plot of unit length for packets arrived at the receiver when no watermarking was done on a regular network and when watermarking was performed on Tor. Our experiment showed that the injected watermark was distorted considerably during the transition of packets through the Tor network. However, if we can inject the watermarks based on controlled TCP packet size (e.g., 498 bytes for each packet), the watermarking detection rate shows significant improvements (about 90%) in Tor, which is shown in Figure 1(c) compared to the original watermarking pattern as shown in Figure 1(d). Based on this ob-

servation, in this paper, we propose a new watermarking attack based on packet repacketization, which works effectively on the Tor network. To counter the presented new watermarking attack, we propose a new scheduling scheme for Tor nodes to manage their buffer. Our contributions are highlighted in the following two paragraphs.

Our proposed packet repacketization technique is to modify the watermarking technique to accommodate Tor’s transformations on the network flow. The traditional watermarking technique suggests that random merging of packets can be accommodated if the network flow is sufficiently long. However, repacketization at the exit onion router (connect to the web server) will destroy the watermarking pattern due to the TCP buffer transmission scheduling scheme. To overcome this issue, we propose a mechanism by which the packet size is controlled at the attacker when introducing watermarks to avoid repacketization at the Tor exit routers. Further, the interval size to embed the watermark is also chosen large to accommodate delays in Tor nodes. Note that the proposed watermarking attack is based on un-encrypted traffic at the web server side, where many man-in-the-middle (MITM) attacks allow attackers to perform the presented watermarking attack. Our implementation results demonstrate that the repacketization technique is very effective in Tor networks to identify the injected watermarks.

To counter the introduced new watermarking attacks, Tor needs to adjust its buffer scheduling schemes accordingly. To this end, we proposed a countermeasure to the revised watermarking attack by modifying Tor’s processing technique. Our design goals are three-fold: (1) the new scheme should not introduce additional delay to existing Tor connections, (2) the modification should not change existing Tor implementations significantly, and (3) the introduced computational overhead is negligible (i.e., it should not consume too much CPU time and memory of onion routers). To achieve our goal, we require each Tor node to monitor its received data packet size. If the data packet size is within the range 400-500 bytes and a continuous flow is delivered in a relatively long period, e.g., a few hundred packets. The Tor node will flag the corresponding circuit. Since the circuit (or a Tor connection) buffers in a Tor node is accessed in a round robin fashion with a variable access interval ΔT_B , where the ΔT_B is usually determined by the number of Tor circuits and the availability of Tor TCP buffer, the Tor node can purposely introduce a factor k by taking $k \cdot \Delta T_B$ ($k = 1, \dots$) to access the flagged Tor circuits. Now, when the flagged circuit is scheduled to be processed, a Tor node bypasses it for $k + 1$ times. This forces the buffers to store cells over a greater time period, which removes any present timing patterns.

Our paper is organized as follows. The next section presents Tor’s architecture along with its working flow. The watermarking scheme is also described along with its threat model on Tor. Section 3 describes the watermark based attacks. In Section 4 we provide the countermeasure to this attack. The performance analysis of the presented countermeasures is presented in Section 5. Finally we conclude the paper in Section 6.

2 System and Models

Components of the Tor Network Tor network comprises of two types of entities, namely the Onion Proxy (OP) and the Onion Router (OR). The OP is an application installed at the client/user who wants to anonymize its web traffic. The ORs are the intermediate nodes, which relay traffic within Tor overlay networks. Additionally, there are directory servers, which hold information about the ORs. Along the TOR data forwarding path, each OR maintains TLS connections to its downstream and upstream ORs. The TLS connections are used for the link encryption among ORs.

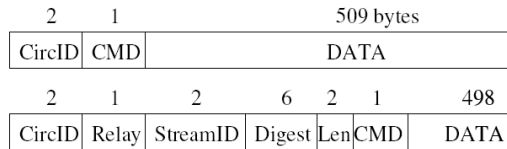
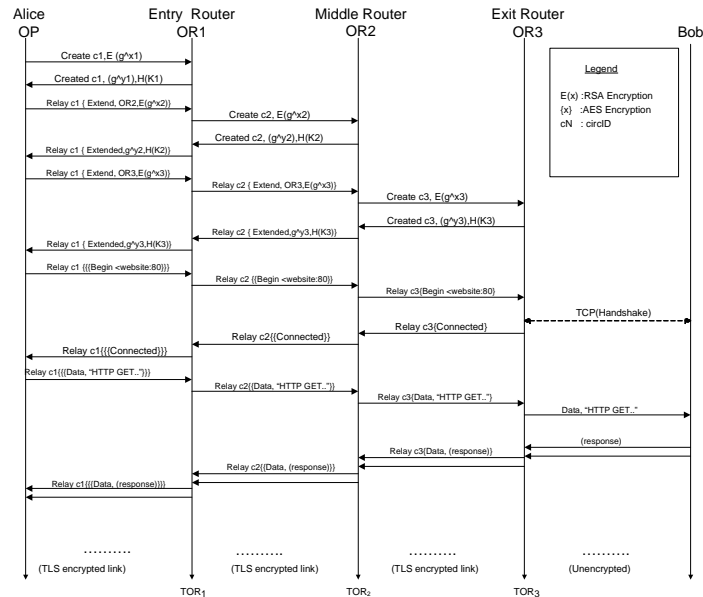
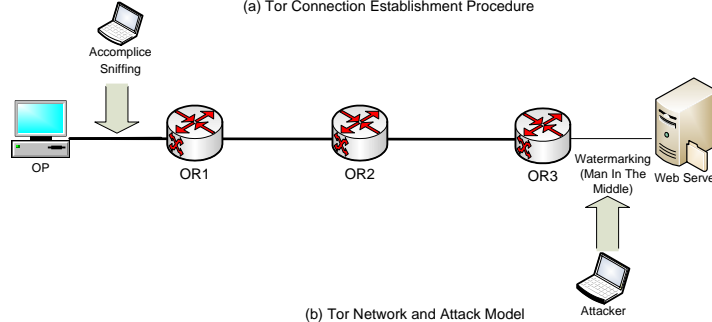


Fig. 2. Tor Cell Format.

Cell Types In Tor, TCP traffic is encapsulated in cells. There are two types of cells traversing through the Tor connections, namely *control* cells and *relay* cells. The cell structure is described in Figure 2. Each cell has a fixed size of 512 bytes, out of which 3 bytes form the header for control cells, whereas 14 bytes are used for relay cell header. The control cell header consists of a circuit identifier (*circID*) that specifies the circuit to which the cell belongs to as many circuits can be multiplexed over a single TLS connection, and a command (*cmd*) to describe the processing of the payload. The relay cells have additional header describing the stream identifier (*streamID*), checksum for integrity checking (*Digest*), length (*Len*), and relay command (*Relay*). For each relay cell, the maximum data that can be packed in a single Tor cell is 498 bytes.



(a) Tor Connection Establishment Procedure



(b) Tor Network and Attack Model

Fig. 3. Tor Connection Establishment and Attack Model.

Tor's Circuit Establishment To set up a Tor circuit, an OP (suppose at Alice's computer) first contacts the directory server for a list of active Tor nodes. Alice's client then chooses a random path to the destination. All the data is encrypted and sent along this path, excepting the last link, which usually are not encrypted. However, the plaintexts only expose the IP address of the exit OR but not the IP address of the real sender. The process is described in Figure 3(a). The OP installed at the client chooses a series of onion routers from the locally cached directory. The default setting is to choose 3 routers, which results in a path length of 3 in Tor. The OP first selects an Exit Router OR3, which must have a suitable exit policy. Next in line is the Entry Router OR1 and then the

intermediate router OR2. These routers together constitute the circuit, on which the Tor cells are relayed. The circuit building on TLS and it is done incrementally, i.e., one hop at a time.

Once a circuit has been established, the client can send relay cells. The relay cells are encrypted multiple times depending on the number of intermediate ORs. When creating a relay cell, the OP first encrypts the cell using the forward key with the exit router OR3, with OR2, and at the end with the entry router OR1, thereby imparting cells an onion like structure. When the relay cells arrive at ORs, they are decrypted by one layer, i.e., each OR removes its share of encryption. A message digest is then checked to ensure integrity. The exit router after decryption is usually left the data with plaintext. The exit router acts as a direct proxy and opens a TCP connection with the intended destination.

The same process occurs in reverse, where each OR adds its own layer of encryption to a cell on its route to the OP. The OP treats incoming cells accordingly by decrypting the cells based on the backward keys negotiated with the ORs. The circuit tear down happens in a similar way, when either the OP can send a destroy cell or a truncate cell to incrementally disable the circuit connections.

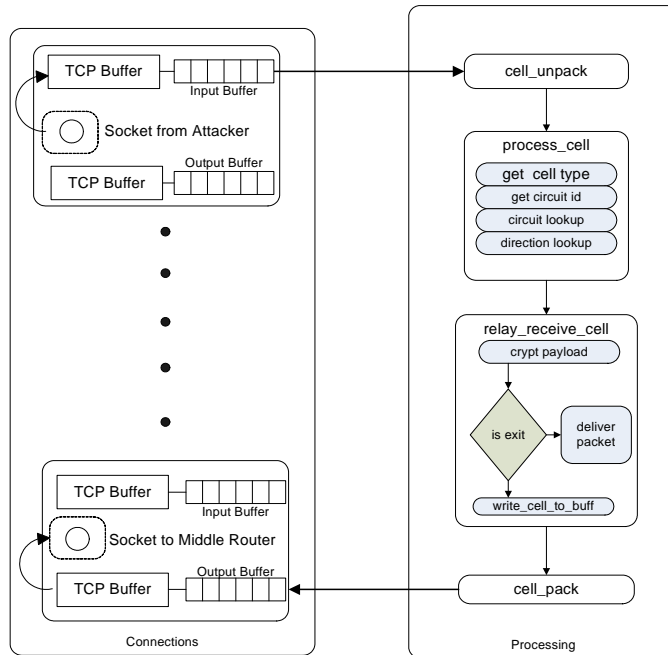


Fig. 4. OR Processing.

Data-path Latency in Tor The data-path latency in Tor can be estimated as the sum of the network latency along with the latency incurred at a Tor node. This can be estimated as $L = \sum TOR_i + T_{network}$ where L refers to the latency, TOR_i refers to the latency introduced at the i^{th} Tor node, and $T_{network}$ refers to network jitter. TOR_i can be further conceived as the time required for processing each Tor cell, along with the time cells spent waiting in the TCP buffers. In Tor the network latency $T_{network}$ usually has little contribution in disturbing watermarking patterns since the network latency will not vary much once a Tor circuit is set up. Since watermarking detection is based on inter-packet delay, the network delay can be ignored in watermark detection.

TOR_i can be modeled based on a number of processing procedures in a Tor node. In Figure 4, we present the packet processing of an exit OR. Thus, we can model the latency TOR_{exit} of an exit OR as follows:

$$\begin{aligned}
 TOR_{exit} = & \\
 & TOR_{exit,tcp-input} + TOR_{exit,circuit-input} + TOR_{exit,cell-processing} \\
 & + TOR_{exit,circuit-output} + TOR_{exit,tcp-output}. \tag{1}
 \end{aligned}$$

The subscripts in (1) described where the latency is considered. The unencrypted TCP packets received from a web server (or an attacker) is first pushed into a TCP input buffer, from where the OR needs to perform repacketization to form cells in the OR’s circuit buffer and then create encrypted cells. Once the cell formation is down, a cell is pushed into circuit output buffer and waits for TCP call to put the cell into TCP output buffer for transmission. During this process, the latency spent in TCP input buffer and Tor circuit input buffer, and the cell processing time can be considered as constant time intervals. The major latency variation introduced by an OR is at the circuit output buffer and TCP output buffer. In fact the $TOR_{exit,circuit-output}$ and $TOR_{exit,tcp-output}$ are highly correlated. The availability of TCP output buffer is determined by TCP sliding window algorithm used in congestion control. Since multiple Tor circuits can be set up between the exit router and the intermediate router, OR takes a round-robin fashion to retrieve the cells from each circuit buffer and pushes cells into the connection’s TCP buffer for transmission. We can use T_B to represent the latency incurred due to this operation:

$$T_B = \alpha TOR_{exit,circuit-output} + TOR_{exit,tcp-output},$$

where $\alpha = 1, \dots$ is an integer. It is determined by the TCP congestion status. If the TCP connection has no congestion, $\alpha = 1$. If the TCP connection is highly congested, $\alpha > 1$, which means Tor cells need to stay in the circuit queue and wait for the congestion window is released. In [7],

the authors investigated the latency of Tor network and they found that the maximum delay occurs due to the congestion control used in TCP, which is T_B .

2.1 Watermarking-based Threat Model

Tor does not protect against global adversaries. It however, does consider traffic analysis attacks where an attacker can compromise a section of the network. The adversary can mount passive and active attacks. Our attack model falls within the Tor’s threat model and compromises anonymity without compromising Tor nodes. This is contrast to many existing Tor attacks (e.g., [5]) that attackers need to compromise Tor nodes. The goal of attackers is to analyze the vulnerability of Tor against watermarking attack and to evaluate the attack’s effectiveness by changing the packet size and timing interval for embedding the watermark.

Attack System Setup The attack is based on adding a pattern in the network flow from the server that a Tor client is trying to access. The repacketization attacks are deployed using real Tor network that is illustrated in Figure 3(b). We establish web connections through Tor networks between an OP and a web server, which behave normally. The attacker can sniff the traffic received by the OP. It also can intercept the network flow between the server and the exit Tor Node using man-in-the-middle (MITM) attacks (e.g., ARP spoofing). It then introduces delays (i.e., watermarks) and reshapes packets size. We must note that the attack model requires attackers to perform repacketization on intercepted packets, thus it does not work on encrypted flows (e.g., using SSL).

Once the attacker intercepts the network flow, he needs to change the packet size avoid repacketization at the exit Tor node. This is because Tor processes packets into fixed size cells of 512 bytes and pushes them in a Tor circuit buffer. Thus, to improve the detection rate, it is desired that a single packet sent by the attacker forms a single cell at the exit Tor node. The inter-packet delay is manipulated according to the watermarking scheme. This delay needs to be adjusted carefully taking into consideration the latency introduced by Tor nodes, and also the overall latency in the network flow since Tor might change its associated circuit. Thus, if the timing interval for the watermark is small than the latency introduced by Tor, the watermark would be garbled. Also if the packets are not resized, Tor’s repacketization mechanism would hinder the watermarking process. Thus, if the above precautions are not taken the watermark would be considerably distorted.

Watermarking Model Here, we present the basic construction of watermarking attacks and its principal along with its formulation. Watermarking technique [6] extends the functionality of the previous watermarking techniques [8][9] and provides robustness against not only against timing perturbation but also repacketization. This technique uses timing of packets, an invariant characteristic of network flow, for traffic analysis. The duration of each flow is partitioned into small intervals and inter packet delay between packets is adjusted for manipulating the packet count within an interval. The population of packets within the intervals determines the watermarking bit encoded in them.

According to the model proposed by Wang et al. [6] watermarking can be explained as follows: Let F be a flow selected from a large pool of interactive traffic flow. The selected flow F can have packets P_1, P_2, \dots, P_n , indexed according to the arrival timing order. The flow F is divided into I_i intervals ($i > 0$) each of duration T starting from a random offset o (> 0). An interval I_i then contains X_i contiguous packets, where X_i is a random variable representing the number of packets in the interval I_i . Then, r (> 0) pairs of two consecutive intervals are selected randomly. For each pair of intervals, the first interval is denoted as $X_{1,k}$, whereas the second interval is denoted as $X_{2,k}$ ($k = 1, 2, \dots, r$). For a long lasting flow $X_{1,k}$ ($k = 1, \dots, r$) are independently and identically distributed (*iid*) in the large population ($X_i, i > 0$). Similarly $X_{2,k}$ ($k = 1, \dots, r$) are also i.i.d. However, each pairs ($X_{1,k}$ and $X_{2,k}$) may not be independent of each other, depending on the distribution of X_i . The mean packet count difference of the above selected interval pairs is:

$$\bar{Y}_k = \frac{X_{1,k} - X_{2,k}}{2}, \quad \text{where } (k = 1, 2, \dots, r). \quad (2)$$

$X_{1,k}$, $X_{2,k}$, and \bar{Y}_k ($k = 1, 2, \dots, r$) are i.i.d. Now, with sample size r , representing redundancy, the sample mean of \bar{Y}_k is defined as:

$$\bar{Y}_r = \frac{1}{r} \sum_{k=1}^r \bar{Y}_k. \quad (3)$$

Since \bar{Y}_k ($k = 1, 2, \dots, r$) are i.i.d., thus \bar{Y}_r is symmetric about the mean with its variance decreasing as we increase r . Actually we are using r redundant bits for each watermark bit, hence entirely $2 * l * r$ intervals are required to be selected.

Watermark Encoding / Decoding Here, we assume both the watermark encoder and decoder share the watermarking parameters $\langle o, T, S, w \rangle$

where σ (> 0) is the random offset, T is the interval length, S is the embedding interval selection function, and w is the binary watermark of l bits. Specifically, the interval-based watermarking encodes a watermark bit by either increasing or decreasing \bar{Y}_r by an amount of μ_x , the mean of the packet distribution, depending on the given value of the watermark bit. According to [8] and [6], increasing \bar{Y}_r can be achieved by increasing $\frac{1}{r} \sum_{k=1}^r X_{1,k}$ by μ_x , and simultaneously decreasing \bar{Y}_r can be achieved by decreasing $\frac{1}{r} \sum_{k=1}^r X_{1,k}$ by μ_x . The former is accomplished by loading each of the corresponding intervals $I_{1,k}$ ($k = 1, 2, \dots, r$) with all the packets in the preceding interval. This is done by shifting all packets of previous interval into this interval by adding a maximum delay T to each of the packets. The latter is accomplished by simply clearing each of the corresponding intervals $I_{2,k}$ ($k = 1, 2, \dots, r$). All the packets are delayed to the following interval in the same manner as shown in Figure 5. At least one non-embedding interval is required as a buffer between any two successive embedding interval pairs in order to avoid conflicts between embedding interval pairs. Decreasing \bar{Y}_r by μ_x can be done in the opposite way.

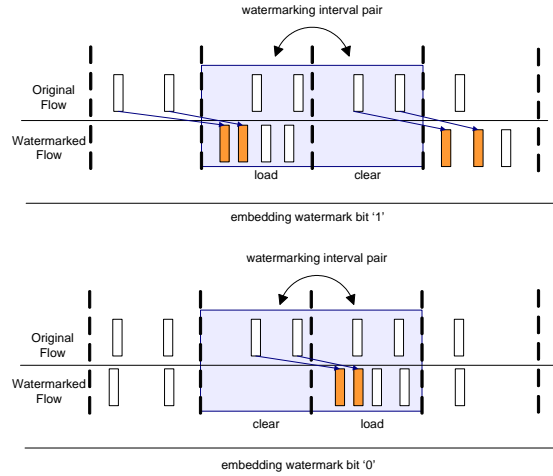


Fig. 5. Watermark Embedding.

To encode a watermark bit '1', \bar{Y}_r is increased as stated above so that the result, denoted as \bar{Y}_r^w is positive ($\bar{Y}_r^w > 0$). On the other hand, to encode a watermark bit '0', \bar{Y}_r is decreased so that the result is negative ($\bar{Y}_r^w < 0$). It is not guaranteed that \bar{Y}_r can be increased; the preceding interval may happen not to contain any packets. The use of redundant coding reduces the likelihood that a particular value cannot be coded

because of an absence of packets. In addition, the watermark itself must be long (l bits) enough to tolerate small errors in coding.

Let $Y_r^{w'}$ at a decoder be a random variable comparable to Y_r^w at an encoder. To decode a watermark w of a watermarked flow F_w , $Y_r^{w'}$ is computed over the same embedding intervals used for encoding w , and then compared against 0; a positive result implies a watermark bit of ‘1’, whereas a negative result implies a watermark bit of ‘0’. A correct correlation is achieved if $w = w'$.

3 Repacketization Watermark Attacks on Tor

According to the authors in [8], the watermarking scheme works on any anonymous communication network including Tor. They assume that various flow transformations can be accommodated with larger duration of network flow. However, Tor’s processing mechanism removes traffic delay patterns from a network flow if the packet sizes are not controlled. In the following sections we present the analysis of the watermarking technique on the Tor network. We first present an analysis of using watermarking technique on Tor without repacketization. Then, we analyze the effect of changing packet size with the watermarking technique on Tor.

3.1 Watermark Attack without Repacketization

In this section, we explain the analysis of traditional watermarking attack on Tor. Watermarking attack is based on manipulating the inter-packet delay in a network flow. The attack assumes that any repacketization during network flow can be accounted for if the flow is long enough. The basis of this claim is that if all packets are repacketized constantly the embedded delay patterns would still exist. However, this does not hold true in Tor. The reason is that Tor repacketizes the data coming from the network flow and it adds up a lot of latency into the packets while they are in transit through the ORs.

The various transformations that occur to data sent from a sender over the Tor network include, repacketization, where the data is broken into fixed sized Tor cells of 512 bytes with a maximum of 498 bytes available for data. There are various latency factors also involved during the course of transition of data through the network as illustrated in Section 2.1.3. The transport latency is fairly small compared to the other latencies involved. As given by [7], one of the major sources of latencies is the TCP congestion control mechanism. While multiplexing the streams is good for anonymity, it is a bad idea in terms of performance. Since data belonging to different circuits are multiplexed on to the same buffer,

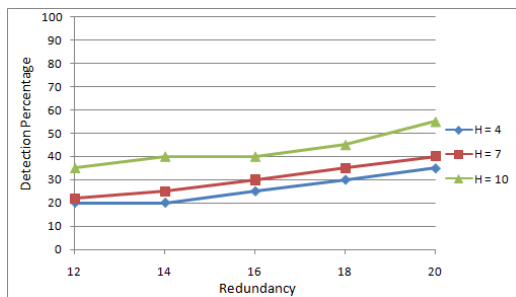


Fig. 6. Watermark on Tor.

this results in the swelling of output buffers of an OR. When packets are dropped or re-ordered, the TCP stack will buffer the available data on input buffers until the missing in-order component is available. The cells for the other circuits might still be available in-order but would be delayed due to missing cells for another circuit. This leads in considerable delays of the packets flowing through the network. Thus, after the packets are repacketized in the input buffer and put onto the output buffer to be written over the socket, they spend a lot of time waiting. This delay along with the repacketization results in the distortion of the watermark. This happens because if the packets are combined together, the number of packets in a watermarking interval is reduced. This allows the packets from the next interval to enter the current interval, creating a constant flow. The reverse happens when packets are fragmented, and the current interval gets crowded with packets and pushes the packets out from the next interval resulting in distortion of the watermark.

The watermarking parameters that were used in the traditional watermarking attack were chosen independent of the network. The experiments in [8] were performed using an interval of 500ms, with a maximum delay of 350 ms. They required 11 minutes of active browsing for a 32-bit watermark with 20 repetitions to be completely embedded. They required around 5,500 packets to penetrate the anonymity of `anonymizer.com`. The results of the experiments performed with these settings on the Tor network is given in Figure 6. In the Figure, H refers to the Hamming distance, which is basically the number of permissible bits which can mismatch. From the Figure, it shows that the watermarking performs badly without repacketization.

3.2 Watermark Attack with Repacketization

In Figure 7(a), we present a comparative study by repacketizing the intercepted traffic from the web server to a fixed packet with the following

fixed TCP payloads size 100 bytes, 200 bytes, 300 bytes, 400 bytes, 470 bytes, 480 bytes, 498 bytes, 500 bytes, and 600 bytes. We change the MTU on the attacker side to the above mentioned packet size to restrict the bottleneck of the network to this value. This is done along with disabling the Nagle’s algorithm by setting the *TCP_NODELAY* parameter on the attacker. This is done to make sure that only a single packet forms the content of a single Tor cell. The watermarking interval length is also adjusted to 1000ms, with maximum delay of 800ms. The results of the experiments done after these transformations are done are shown in Figure 7(b). We choose a watermark length of 32 bits with 12, 14, 16, 18 and 20 repetitions. The experiments show that the watermarking detection rates are significantly improved when the TCP payload is between 400 bytes and 498 bytes. This phenomenon confirms our assumption that the attacker repacketizes the packet size close to OR cell size, the distortion of watermarks due to TCP buffer queues and congestion controls will be reduced significantly.

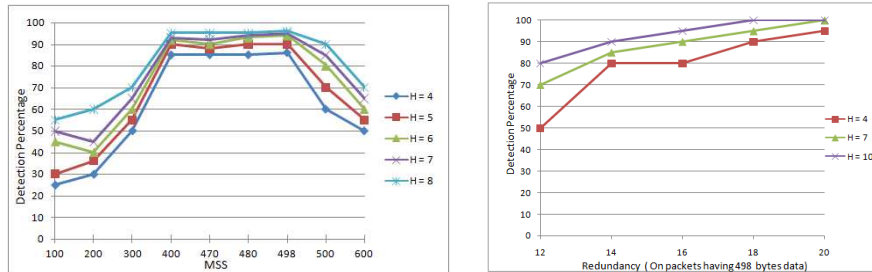


Fig. 7. (a) Maximum Segment Size vs Detection Rate (b) Revised Watermark on Tor.

Tor nodes form fixed sized cells of 512 bytes from the packets received, and place these cells on the circuit input buffer. They consider the input packets as a stream of bytes received from TCP. If the amount of data received after a *recv()* system call when the TCP payload in a packet is more than 498 bytes, the OR splits them into multiple Tor cells. If the received number of bytes is less than 498, the cell is padded with zeroes when no following packets in the TCP input buffer, or the OR will combine the data from multiple packets into a fixed 512-byte cell. This fetching of data bytes from TCP stream results in either combining multiple packets into a single cell or splitting a single packet into multiple cells. This repacketization results in the distortion of the watermarking pattern present within the packets.

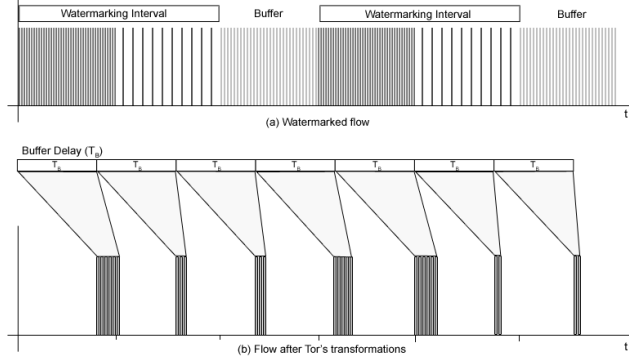


Fig. 8. Repacketization in Tor.

As given in Figure 8, an example is shown on the effect of repacketization on the watermarked flow. Here we present a simple analytical model to represent the complicated buffer delay model of TCP connections. We have two watermarking interval pairs which have watermark bit ‘1’ in it. When the arriving packets have a packet size of 140 bytes, they are re-arranged into fixed cells of 512 bytes. The 140 bytes include 20 bytes of TCP header and 20 bytes of IPv4 header. Thus the data part is of 100 bytes. Now, if delay between continuous packets is negligible, Tor would combine 5 such packets into a single Tor cell. After the combination of packets into Tor cells these cells wait in the output buffer queue to be scheduled onto the TCP output buffer. The time they wait in the circuit buffer can be called ΔT_B . Tor fetches data from these buffers in a round robin manner, which introduces the delay ΔT_B . Figure 8 shows the successive writes to the buffer after each buffer delay. As shown the number of packets in the first half of the watermarking interval is reduced due to repacketization and some of them are delayed into the next interval due to the buffer delay. This same applies to following buffer, which actually pushes packets belonging to the buffer into the next watermarking interval. This results in a count of almost identical packets in the watermarking interval pair. Thus, the repacketization combined with buffer delay distorts the watermark considerably over a small period of time.

To model the repacketization, we consider the number of packets in the first half of the watermarking interval to be N_P .

$$N_B = \frac{N_P}{T} \Delta T_B. \quad (4)$$

In (4), N_B refers to the number of packets fetched into the TCP buffer for one read call. T refers to the watermarking interval size, and ΔT_B refers

to the delay at a Tor node. Now, let N_R refer to the number of packets remaining after the buffer read. We can derive the following relations:

$$N_R = N_P - N_B. \quad (5)$$

Thus, N_R packets are shifted from each watermarking interval to the next interval when packets are removed from the circuit buffer to the TCP buffer. As shown in Figure 8, this results in disruption of the watermark as the packets belonging to the later interval entering current interval, transforming a watermark bit ‘1’ to ‘0’. The above presented model is accordance with the our assumption of minimizing packet splitting by controlling the packet size, which is shown in Figure 7. Since Tor would split any packet containing more than 498 bytes we observe a drop in detection rate when data is 500 bytes. Again, when the data bytes in a packet are lesser than 300 bytes Tor packages multiple packets into single cell, thus reducing the number of packets within an interval.

4 Countermeasures to adaptive Watermarking attack

In [4], the authors proposed a countermeasure to the watermarking attacks. Their solution is applicable if the attacker wants to deploy the attack multiple times over the network. They collect a series of flows and try to actively extract timing pattern among packets. However, in our approach, we monitor the nodes we suspect to be communicating. Thus, we do not need the watermark to be introduced multiple times. Since Tor nodes by default processes the circuit buffers in a round robin manner, each buffer gets a delay of ΔT_B . This delay is the sum of the delays of the round robin processing along with the TCP multiplexing.

Our countermeasure requires minimal level of changes of Tor implementation. From our experiments we have identified that the watermarking scheme would work if the packet size is within a range of 300 bytes to 500 bytes. If a Tor node actively scans network flow which have packets within this range it can flag that circuit. Now, once these packets are packaged into fixed sized Tor cells and wait in the circuit buffer to get flushed onto the TCP buffer, they are processed in a round robin manner, which gives a constant delay to all the packets in the flow. Our solution is to treat the circuits differently, which are flagged. Once a circuit has been identified which has packets arriving at a constant rate with sizes within the watermarkable range, we trigger a setting mechanism to flag the circuit. When each circuit is scheduled, a validation of whether it is flagged or not is performed. If a circuit is found to have been flagged it is bypassed for the first time. This means that the *send()* is not called on a circuit for the first time when it is encountered to be flagged. This

results in a larger delay of the packets, which is more than the interval length for embedding a watermark. Now since the buffer was not processed when it was scheduled packets from the next interval arrive and sit in the buffer completely distorting the watermark. The results of our analysis are presented in the following section.

Our strategy does not introduce unnecessary delays in connections which do not fall within the identified range. Thus, the latency of the network is not increased for all connections on the network. This solution is simple as well as does not require an intensive computation overhead at the Tor node to identify embedded watermarks. This scheme simply puts a phase shift in the watermarking technique rendering it useless. Based on the Internet traffic report, our strategy does not introduce unnecessary delays in connections which do not fall within the identified range. Thus, the latency of the network is not increased for all connections on the network. This solution is simple as well as does not require an intensive computation overhead at the Tor node to identify embedded watermarks.

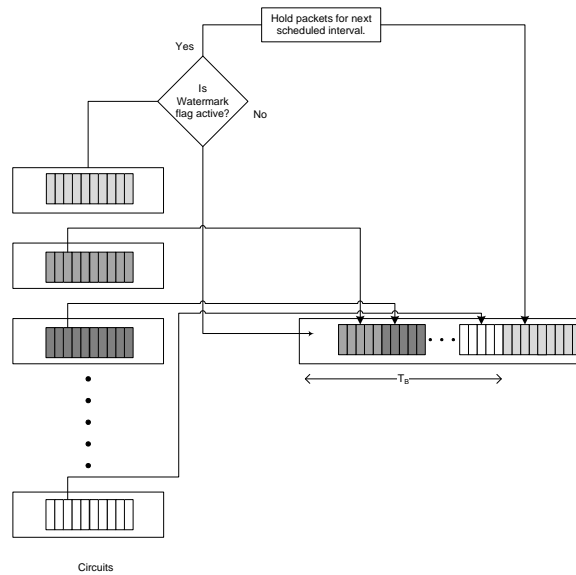


Fig. 9. Countermeasure Technique.

5 Performance Evaluations

We perform a performance assessment of the proposed scheme to show that the watermarking scheme would indeed not work if the scheduling of Tor buffers is not done on a fair round robin manner. Figure 9 shows

when a circuit is confirmed to have the watermark flag active it is not scheduled immediately. It is rather put on *hold* for the next available *read*.

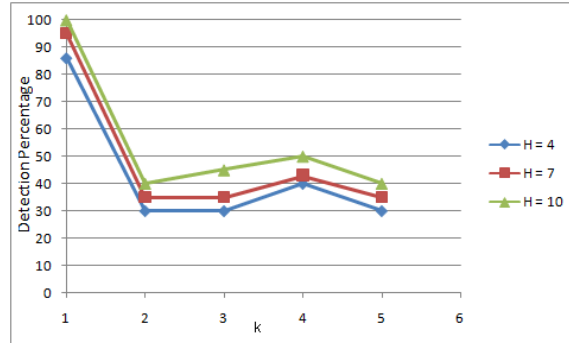


Fig. 10. Detection rate v.s. different number of delays.

Our analysis shows that when different values of k is chosen the watermark with 498 bytes of data in each packet does not perform too well. As given in Figure 10, delaying the suspected interval results in low detection rates. Basically, k refers to the number of times a scheduled circuit is skipped i.e., the buffer is ignored for another ΔT_B unit time. Note that for $k = 1$ is the watermarking scheme when the buffer is not bypassed for the flagged circuit. The choice of k is random, and should be left on a Tor node's discretion. However, k must have an upper bound to ignore the overflowing input buffers. In our analysis, the k is selected up to 5.

Our policy of delaying the packets which are in the range is justified. Based on the Internet traffic packet distribution report [2] provided by CAIDA in 2008, we observed that (1) current packet sizes seem mostly bimodal at 40 bytes and 1500 bytes, and (2) it has been observed in some cases, a strong mode around 1300-1500 bytes. These results show that flagging packets within the range 300 – 500 will not interfere with good traffic that usually has larger packet size. Since most of the TCP data traffic on the internet has more than 1000 bytes of data, this policy of holding the buffers would be only applicable to very few network flows.

6 Conclusion

We have analyzed Tor effectiveness against watermarking technique. We find that if an attacker performs a man in the middle attack and controls the packet sizes of a network flow, the watermarking attack can be deployed successfully. Our experiments show that the watermarking

technique is effective within a certain interval of packet sizes. In addition to the demonstration of the watermarking technique we propose a change in the behavior of Tor's buffer for packets in the workable range of watermarking. We analyze if the buffers store packets from an attacker for a long period of time the watermarking attack cannot be performed. We present the result of our proposal and notice that the watermarking scheme does not perform well. Thus, if the behavior of Tor's buffer processing is modeled according to our proposal Tor would be more secure against watermarking based attacks.

Acknowledgement

The authors would like to thank anonymous reviewers for their comments to improve the quality of this paper. This work is sponsored by NSF research grant CNS-1029546.

References

1. Anonymizer Inc.: <http://anonymizer.com>.
2. CAIDA: Packet size distribution comparison between internet links in 1998 and 2008, http://www.caida.org/research/traffic-analysis/pkt_size_distribution/graphs.xml (2008)
3. Dingledine, R., Mathewson, N., Syverson, P.: Tor: the second-generation onion router. In: Proceedings of the 13th conference on USENIX Security Symposium-Volume 13 table of contents. pp. 303–320. USENIX Association Berkeley, CA, USA (2004)
4. Kiyavash, N., Houmansadr, A., Nikita, N.: Multi-flow attacks against network flow watermarking schemes. In: Proceedings of the Usenix Security Symposium (2008)
5. Murdoch, S., Danezis, G.: Low-cost traffic analysis of tor. In: 2005 IEEE Symposium on Security and Privacy. pp. 183–195 (2005)
6. Pyun, Y., Park, Y., Wang, X., Reeves, D.: Tracing traffic through intermediate hosts that repacketize flows. In: Proceedings of the 26th IEEE Conference on Computer Communications (INFOCOM). pp. 634–642 (2007)
7. Reardon, J.: Improving Tor using a TCP-over-DTLS Tunnel. Master's thesis, University of Waterloo (September 2008)
8. Wang, X., Chen, S., Jajodia, S.: Network flow watermarking attack on low-latency anonymous communication systems. In: Proceedings of the 2007 IEEE Symposium on Security and Privacy. pp. 116–130 (2007)
9. Wang, X., Reeves, D., Wu, S., Yuill, J.: Sleepy watermark tracing: An active network-based intrusion response framework. In: Proceedings of the 16th International Conference on Information Security(IFIP/Sec) (2001)