Timing Analysis Attack Simulation in Tor Networks

Matteo Martelli, Davide Berardi

June 12, 2015

Abstract

In this document we talk about blablabla

1 Introduction

Protecting data privacy on the Web is a very hot topic nowadays. Users of the Web may want to surf without the risk that their personal informations can be read by other users. One of the most largely-used architecture for this purpose is the *Onion Routing* and its protocol implementation: *Tor*[2]. In fact, the latter is modeled with several techniques with the aim to provide communication security and data privacy to its network users. Anyway there have been recent papers pointing out the Tor's vulnerabilities. As the Tor community itself stages, "Tor does not provide protection against end-to-end timing attacks"[1], thus the chance for an attacker to eavesdrop a Tor communication traffic and discover the users involved in it by a timing analysis is a well known vulnerability of Tor.

In this kind of analysis, in order to identify the source of a communication, the attacker should be able to trace the outgoing traffic and the incoming traffic from both the entering and exiting node of the communication path. Clearly a timing analysis is feasible only under a certain amount of conditions that are often hard to satisfy. As instance, discovering that a generic user U is connecting to a server S over a Tor communication may require tracing the traffic of many users in the network, as the attacker cannot know which users may be interested in connecting with S. Also, there may be the need of tracing more than just the interested server because the attacker can find a better time relation between the user U and another server S' than between U and the interested server S, thus the attacker could exclude U to be a possible connection source for S. In the section 2 we will better describe how Tor works and how the end-to-end timing attack could be performed.

In order to test the feasibility and the parameters involved in a time analysis attack over the Tor network, we set up a simulation scenario in which a series of simulation runs have been performed and some interesting empirical results have been taken out and analyzed. At the end we will point out how the Tor time analysis vulnerability can be critical and we will introduce some proposals to enhance Tor with the view of preventing this kind of attacks.

2 Tor

Tor is an implementation of the onion routing architecture model. The onion routing consists in a technique that provides anonymous connections over a computer network[3].

Tor is born with the aim to allow people to improve their privacy and security on the Internet. Its architecture is based on the Onion Routing model and it's widely used by many user over the world. Users may be interested on using Tor for different purposes such as avoiding website tracking, communicating securely over Internet messaging services, or just web surfing with the access on the services blocked by their local Internet providers.

The idea behind Tor, and Onion Routing as well, is to protect people against a common form of Internet surveillance known as "traffic analysis". Traffic analysis can reveal information about the network traffic such as the source, destination, size, timing and more of the analyzed traffic packets. This can be possible even if the packets are cyphered because the traffic analysis focuses on the header part of the packets that are in plain text.

Thus, simply listing between the sender and recipient on the network, a traffic analysis can be performed. Moreover, spying on multiple parts of the Internet and using some statistical techniques, some attackers can track the communications patterns of many different organizations and individuals.

In the next paragraphs we will discuss more in details about Tor communications and the flaws of the model.

2.1 Tor Internals

The figure 1 shows how a message is cyphered before the communication begins. The communication source, before sending the message, choses a communication path of nodes which the keys are known to the sender. Then the source node is able to create a stack of encryption starting with the key of the last relay node and then continuing backwards with the keys of the other relay nodes in the chain. In this way every node in the communication path can decrypt the package and read the next hop address. After that the final node receives the message, he can send the response back to the originator of the data stream. In this phase the response message is encrypted sequentially by each node in the chain. With this method each relay node can gain access to the previous and the next node addresses only. Anyway the last node of the Onion Routing path, called exit node, send the message to the end point as plain text.

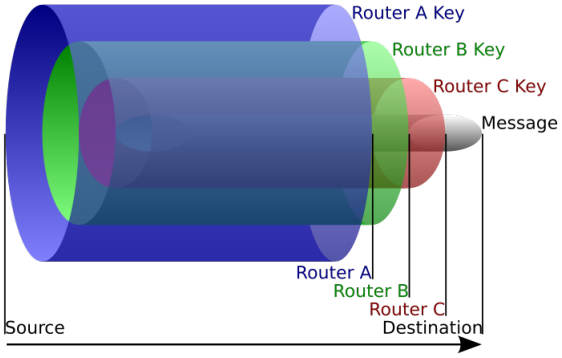

Figure 1: Message encryption layers.

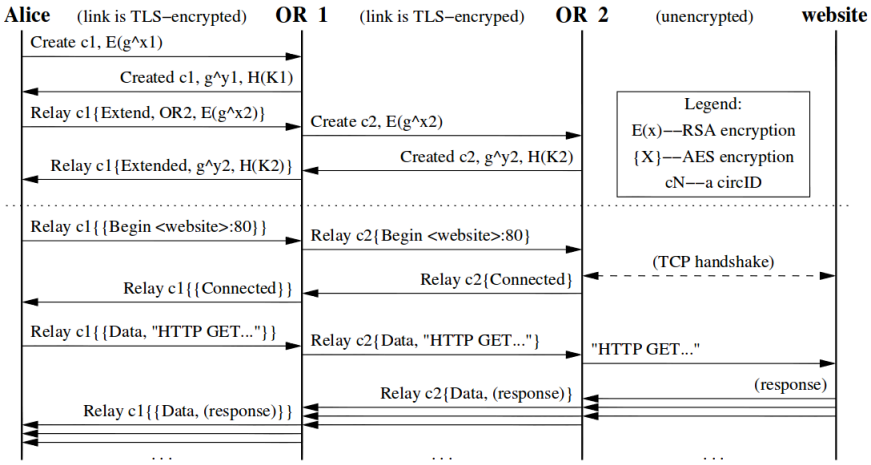

Figure 2: A two relay Onion Router communication.[2]

2.2 Tor attacks

Recent studies showed the existence of different kinds of possible attacks on the Tor network. Apart from Side channel attacks [1] we can classify these attacks in two major families: probabilistic attacks and path selection forcing attacks. The first ones are based on the data analysis that leak from the network. In these cases an attacker that sniff some specific data packets can perform some statistical analysis and may be aware of users identity informations. On the other hand, the path selection forcing attacks are based on some techniques able to route the network traffic through a sequence of nodes handled by the attacker. In our study case we focus on one of the probabilistic attacks: Timing Analysis Attack.

2.2.1 Timing Attack

As Tor development team states Tor is vulnerable to the timing attack analysis:

> *"Tor does not provide protection against end-to-end timing attacks[...]"*

More specifically, an attacker placing between the communication source and the Tor entry node and also between the Tor exit node and the communication destination, can observe the connection time of both end-points and point out their relation. The attacker can, for example, inject some malware code in the communication source machine or in some other entities, i.e. a router, on the network segment before the entry node. Similarly, a way to introduce some eavesdropping point into the communication destination could be found by the attacker.

[1] For example the well known *tor browser attack* .

We will focus on how much this kind of attack can be effective on a real network, considering the amount of the resources available to the attacker.

3 Simulation

In order to analyze the effectiveness of a time analysis attack in a realistic scenario, we decide to consider a simulation environment. We wanted to model the scenario in a way that most of the characteristics of the Tor network architecture would have been considered.

The **shadow simulator**[4] seemed to be a good candidate for our purpose. In the next paragraphs we will introduce shadow and we will illustrate our work based on this simulator.

3.1 Shadow

Most of the simulators used for networks experimentation do not provide the use of real external applications, which their behaviour are often simulated as well. This approach can be reasonable if the analysis is focused on the network layers below the application layer.

In our case we are interested in studying the characteristic of the Tor network which is based on the network application layer. Thus the Tor application is one of the core part of the Tor architecture as it implements the onion routing protocol that let the Tor nodes communicate each others.

It is evident that the Tor application has a significant role in the Tor architecture, thus we chose the shadow simulator as it permits the use of the real Tor applications in a simulated environment. Essentially it allows the execution of a set of local applications that communicates in a simulated computer network. Moreover we could avoid the oversimplification of the system that could have been occurred with a custom implementation.

Applications are executed inside the shadow simulator through dynamic libraries called plug-ins. These plug-ins are interfaces used by shadow to trace a selective set of system functions and re-route them to the simulator instead of letting them proceed directly to the kernel. In this way, the simulator is transparent to the application that may function as like it was running in a standard UNIX environment.

The plug-in that allows the execution of the Tor applications is scallion. The latter provides also some useful tools for the virtual Tor network topology generation.

The virtual network is modeled with a XML configuration file, called blueprint, used by shadow to understand the network structure and some network properties such as link latency, jitter and packet loss rate. The blueprint also tells Shadow what software each virtual node should run at its creation. This is specified with a plug-ins list for each virtual node entry of the XML file.

We implemented three shadow plug-ins and their relative applications able to work alongside scallion in order to experiment a time analysis attack scenario. Our plug-ins and the attack scenario will be illustrated in the next paragraphs.

3.2 Simulation Scenario

3.3 Autosys plug-in

To analyze the tor network traffic we need, at least, some information about the incoming and the outgoing connections.

To achieve this result we have more than a solution: in the first one we can place in the last network element that is under the control of an autonomous system[2] a connection listener which traces a whole subset of network nodes; otherwise we can place the listener in a, malware-like, invisible proxy on the target machine with the aim to trace every outgoing connection. In both solutions we need to place some correspondent exit tracers on the target end points. This proxies can be implemented as a simple raw-socket based sniffer. Unfortunately Shadow doesn't support raw-socket emulation[3], so we implemented the plug-in as a simple event based TCP proxy. In our attack scenario an instance of the autosys plug-in is placed between the client application and the client Tor daemon, also a second instance is placed

[2] That the reason why the plug-in is called "Autosys".

[3] If you launch a simulation with a raw socket based program the simulator inform you about the impossibility of the task.

between the server Tor daemon and the server application.

Algorithm 1 Autosys main loop

```
1:  function AUTOSYS(sa, sb, analyzer)
2:      while True do
3:          if Some data can be read from sa then
4:              indata ← readall(sa)
5:              writeall(sb, indata)
6:              if indata was a connection initiator then
7:                  send(< type(sa);hostname(sa);gettod() >, analyzer)
8:              end if
9:          end if
10:         if Some data can be read from sb then
11:             indata ← readall(sb)
12:             writeall(sa, indata)
13:             if indata was a connection initiator then
14:                 send(< type(sa);hostname(sa);gettod() >, analyzer)
15:             end if
16:         end if
17:     end while
18: end function
```

As we can see in the algorithm 1, where *sa* and *sb* are the two end sockets and *analyzer* is the socket descriptor of the analysis server (the logger), when some connection initiator packets flow trough the proxy the current machine time of day is sent to the analysis server [4].

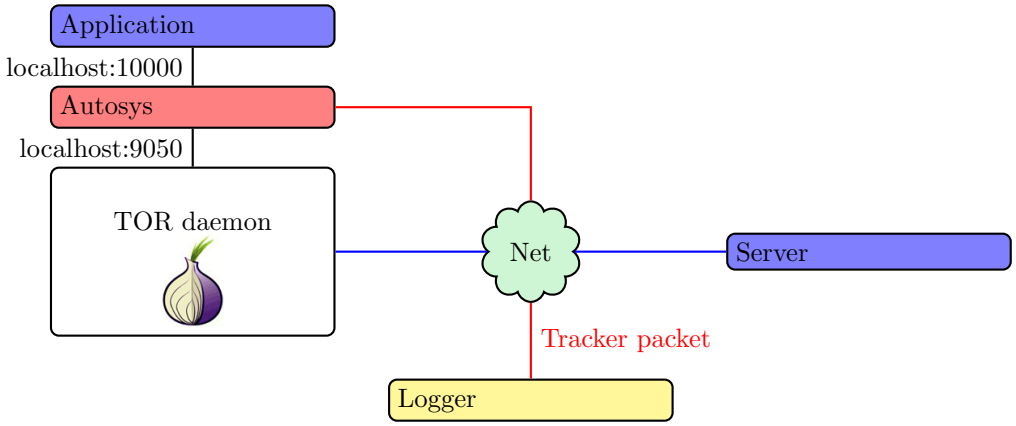

Figure 3: Autosys proxy structure.

3.4 Analyzer plug-in

Data sniffed by the autosys plug-in need to be stored for later analysis, to do so we need another plug-in. This plug-in is called the "Analyzer plug-in", "Analysis server" or simpy "Logger" which will collect all the dead-drops generated by the sniffers. The plug-in is implemented as a simple event driven UDP server, every packet received from the sniffer plug-in is so saved in a single place.

```
host_type;hostname;timestamp
```

Figure 4: Analyzer log Structure

[4] The simulator have a single clock, so we don't have any problem with the clock synchronization.

Referring to the figure 4 we have the following information in the log:

- Host Type
  c to signal that the traced packet referred to a client
  s to signal that the traced packet referred to a server
- Hostname
  The name of the tracked machine
- Timestamp
  The time-stamp of the packet (when it flows trough the autosys plug-in).

In conclusion this raw data will be passed to the phase two to be analyzed by the scripts.

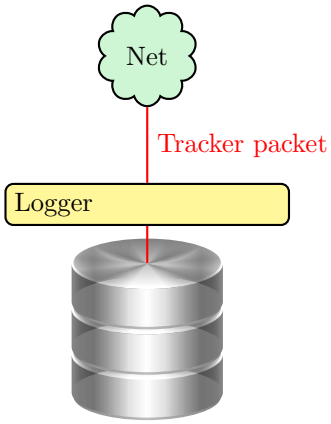

Figure 5: Logger logic structure.

As a subnote this plug-in was implemented as an UDP server for the sake of simplicity and, in opposite to TCP, the low impact on the network.
This can lend us to a little packet-loss but won't impact too much on the time analysis as we will see later on this study.

3.5 Simpletcp plug-in

The last plug-in that we implemented emulate a simple HTTP client/server communication.

3.5.1 Client mode

This plug-in, in client mode, emulate the communication making some (a customizable amount) connections to the server with a random normal distribute sleep time in between of every connection (this parameter is configurable too).
The packet transfered by the plug-in simply contains the host-name of the host machine, this is to break the anonymity of the TOR system and analyze the results in later phases.

3.5.2 Server mode

At the opposite end, a server simpletcp instance, will:

- Receive the connection packets from the clients.
- Add a time-stamp to the current received packet.
- Save the packet to a common file (in our implementation this data file is called as the host-name of the machine hosting the server). This is used in the analysis of the phase 2, to compute a matching probability between the guessed users and the real users.

3.5.3 SOCKS5 operation

4 Data Analysis

4.1 Netbuilder Script

4.2 Analyzer Script

4.3 Empirical Results

5 Future Works

6 Conclusions

References

[1] *Tor: Overview.* `https://www.torproject.org/about/overview.html.en`.

[2] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. Technical report, DTIC Document, 2004.

[3] David Goldschlag, Michael Reed, and Paul Syverson. Onion routing. *Communications of the ACM*, 42(2):39–41, 1999.

[4] Rob Jansen and Nicholas Hopper. Shadow: Running tor in a box for accurate and efficient experimentation. In *Proceedings of the 19th Symposium on Network and Distributed System Security (NDSS)*. Internet Society, February 2012.