Time analysis Based Attacks Simulation in Tor Networks.

Simulazione di Sistemi

July 17, 2015

Davide Berardi 0000712698

Matteo Martelli 0000702472

Università di Bologna.

Table of contents

Introduction

Standard *shapes* of information security:

- Confidentiality
- Integrity
- Availability

There is a new security that we want to obtain: **Anonymity**
Anonymity [...] means that the personal identity, or personally identifiable information of that person is not known.

Introduction

anonymity methods

There are a lot of anonymity driven software online, like *i2p*, *freenet* or *Tor*, we will talk about the last one because is the most used and expanded in the real world (2 million of client per day!).

Onion Routing

The onion routing model is a way to gain anonymity on the net:

- Provides anonymity
- Protects from sniffing

Introduced by David Goldshlag, Paul Syverson and Michael Reed in the 1999.

It recalls an onion because every step **peel** a layer.

Let us see an implementation.

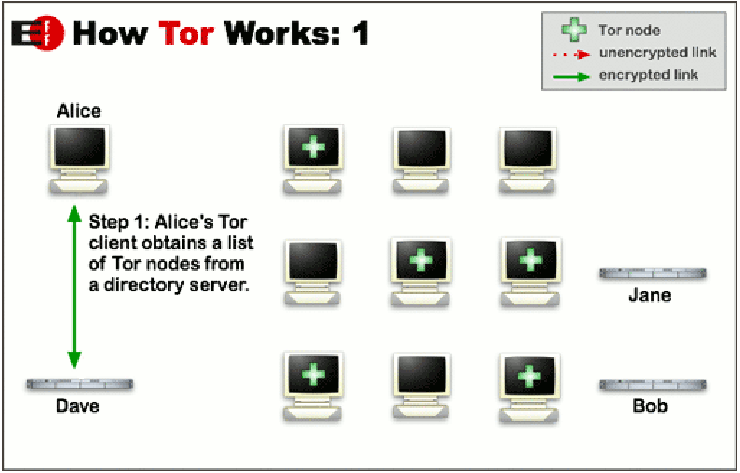
Overview

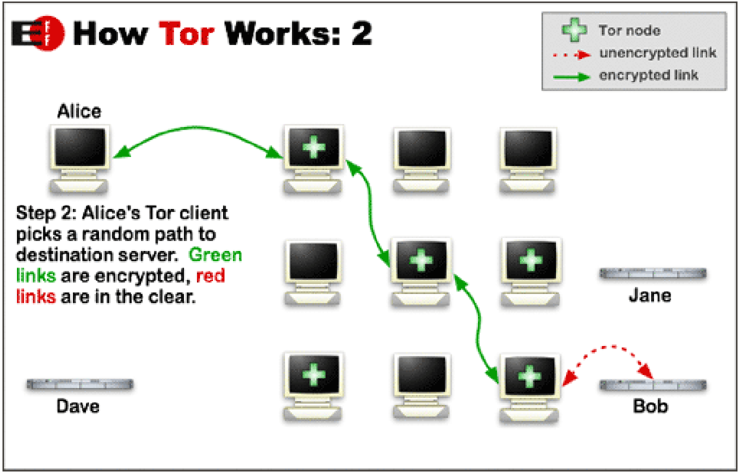
Tor is a group of volunteers that operates to defend anonymity online.
The system is based on an interconnection of machines, called **routers**.
It operates over the network level 4.

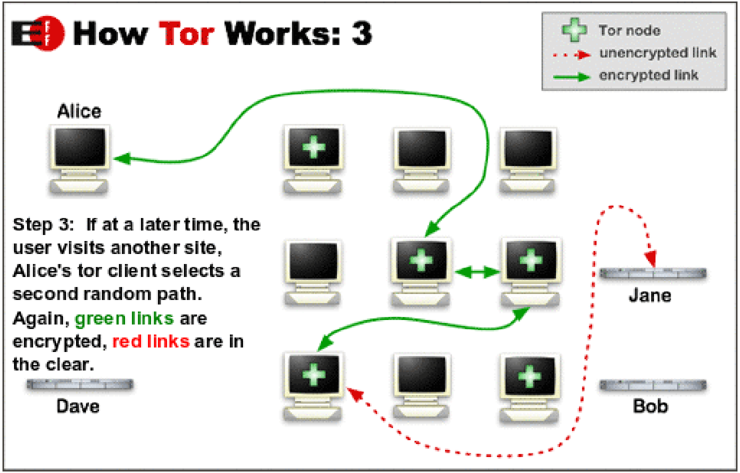
It operates as follow:

Tor

Tor workings

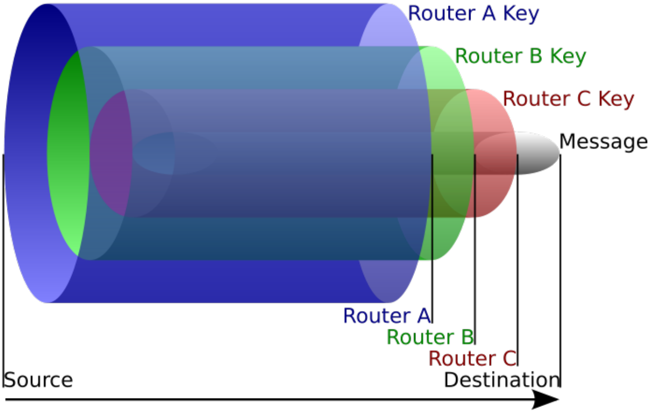



How Tor Works: 2
Tor node
unencrypted link
encrypted link
Alice
Step 2: Alice's Tor client picks a random path to destination server. Green links are encrypted, red links are in the clear.
Jane
Dave
Bob


How Tor Works: 3
Tor node
unencrypted link
encrypted link
Alice
Step 3: If at a later time, the user visits another site, Alice's tor client selects a second random path. Again, green links are encrypted, red links are in the clear.
Jane
Dave
Bob

Tor

Tor encryption



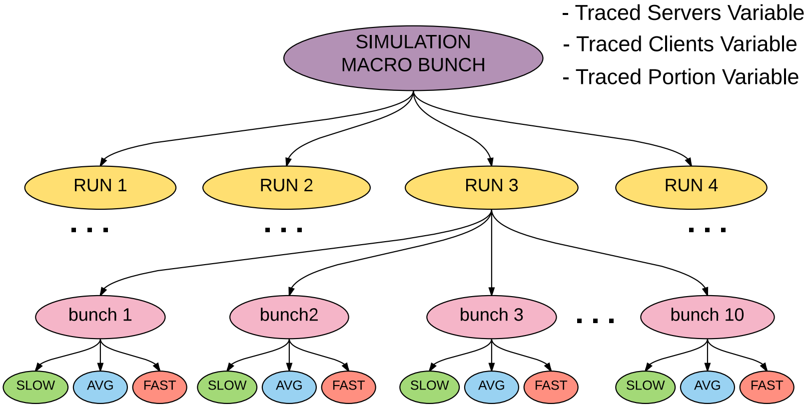
Data Analysis

- Simulation Bunches
- Simulation Handlers
- Analyzer
- Empirical Results

Simulation Bunches

1. Traced clients fixed to the 100% and increasing traced servers at each macro bunch run (0% $\rightarrow$ 100%).
2. Traced servers fixed to the 100% and increasing traced clients at each macro bunch run (0% $\rightarrow$ 100%).
3. Increainsg both traced clients and traced servers (traced portion) at each simulation (0% $\rightarrow$ 100%).

Simulation Bunches

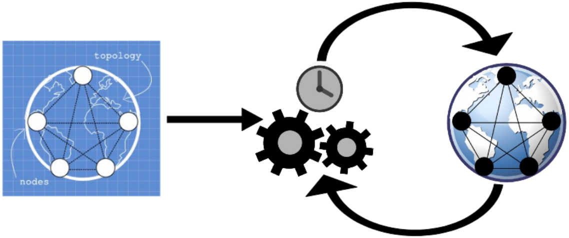


Simulation Handlers

- Netbuilder
- Launcher

Netbuilder

Genereates an XML file that describes the network



Netbuilder

Configuration

Allow the network configuration through:

- The number of TOR exit nodes in the simulation.
- The number of TOR 4authorities[1] nodes in the simulation.
- The number of clients (simpletcp) of the simulation.
- The number of servers (simpletcp) of the simulation.
- The percentage of clients tracked by an autosys plug-in.
- The percentage of servers tracked by an autosys plug-in.
- The density of the network-requests.

[1] A 4 Authority node is simply the database that keep track of the state of the TOR network and the list of the TOR relays/exit-nodes

Netbuilder

Densities

The connection densities are the sleep time thresholds between each client connection request:

- Slow: 800 (mean) - 2000 (mean) milliseconds
- Average: 80 (mean) - 1000 (mean) millisecons
- Fast: 20 (mean) - 100 (mean) milliseconds

Launcher

Algorithm 2 Launcher script

```
for (simulation_run ← 1; simulation_run <= steps; simulation_run++) do
    for (sim_id ← 1; sim_id <= simulations_per_step; sim_id++) do
        for all density in (slow, fast, average) do
            if The client trace percentage is not fixed then
                client_trace_value ← sim_id/simulations_per_step
            end if
            if The server trace percentage is not fixed then
                server_trace_value ← sim_id/simulations_per_step
            end if
            if A configuration is present for < sim_id, density > And the percentages are fixed then
                Use the previous configuration
            else
                Generate a new configuration with net-builder
            end if
            Launch the Shadow Simulator with the appropriate configuration.
        end for
    end for
end for
```

Analyzer
Log file

...	$(*)$
c;client10;1420000000	s;server7;1421023888
s;server7;1420008031	s;server2;1421156205
c;client6;1420005867	c;client8;1421160529
s;server9;1420146660	s;server3;1421318345
s;server6;1420205384	s;server0;1421332488
s;server8;1420252482	c;client7;1421487295
c;client0;1420680882	c;client4;1421634744
c;client1;1421017740	s;server9;1421726485
$(*)$	...

Analyzer

Scan

- For each client connection request *creq*, it looks for candidate server acceptances
- Nested loop "temporally" limited between thr_{MIN} (100ms) thr_{MAX} (6sec)

Time distance

Let $\Delta_t(creq, s)$ be the time distance between a *creq* time-stamp and a server candidate acceptance *s* time-stamp.

Analyzer
Scan

...	$(*)$
c;client10;1420000000	s;server7;1421023888
$\Delta_t < thr_{MIN} \rightarrow$ s;server7;1420008031	s;server2;1421156205
c;client6;1420005867	c;client8;1421160529
s;server9;1420146660	s;server3;1421318345
s;server6;1420205384	s;server0;1421332488
s;server8;1420252482	c;client7;1421487295
c;client0;1420680882	c;client4;1421634744
c;client1;1421017740	s;server9;1421726485 $\leftarrow$ already considered
$(*)$	...

Analyzer
Matching Probability

The likelihood for a server acceptance to be related to a client request can be related to their time distance.

pmatch

$$pmatch(creq, s) = 1 - \frac{\Delta_t(creq, s) - thr_{min}}{thr_{max} - thr_{min}} \tag{1}$$

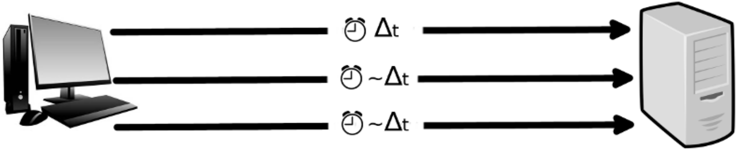
Analyzer

Matching Probability

candidate	pmatch
server9	0.992
server6	0.982
server8	0.975
server7	0.846
server2	0.823
server3	0.769
server0	0.794
...	...

The *pmatch* is higher when the server connection is closer to thr_{min}.

Analyzer
Time Gap



Acceptance Delay Correlation

If a server receives a connection request from a client after a certain time Δ_t, that server will likely receive again another connection from the same client after a time that is close to Δ_t if the Tor communication path is the same as before

Analyzer
Time Gap Average

As the *pmatch* is defined as the Δ_t normalization, let us define the *gap* average of a server s marked as candidate for a client c

$$gap_{AVG}(c, s) = \frac{\sum_{i=0}^{N(c,s)} |pmatch(creq_{i+1}, s) - pmatch(creq_i, s)|}{N(c, s)} \tag{2}$$

where $N(c, s)$ is the number of c connection requests that have been likely accepted from s.

Analyzer
Score

The *score* gained by a server s marked as candidate for a client c

$$score(c, s) = \frac{\sum_{i=0}^{N(c,s)} pmatch(creq_i, s)}{gap_{AVG}(c, s) + 1} \tag{3}$$

Analyzer
Best Candidate

client633		client637		client349	
candidate	**score**	**candidate**	**score**	**candidate**	**score**
<u>server8</u>	9.44	<u>server3</u>	59.17	<u>server0</u>	14.86
server0	7.01	server2	15.14	server1	13.81
server2	6.88	server8	13.96	server5	11.94
server5	6.83	server5	8.33	server2	11.20
...	...	...	...	...	...

Best Candidate

The server candidate with the **highest score** is the best candidate for a certain client.

Analyzer
Real Data

- How much are the estimated results close to the real ones?
- Use of real connections logged by the simple-tcp applications.
- Matched accuracy estimation
- Matched portion estimation

Analyzer

Matched Accuracy

For each client check if the best candidate is the real server that accepted its connections and mark it as **matched**.

If so calculate the client accuracy as the distance between the number of estimated connections N and the number of real connections M:

$$accuracy_c \leftarrow \frac{MIN(M, N)}{MAX(M, N)} \tag{4}$$

The *matched accuracy* is the average of matched client accuracies.

Analyzer

Matched Portion

The matched portion indicates how many traced clients found their real server:

$$matched\_portion = \frac{matched_clients}{traced_clients} \tag{5}$$