Anonymity Analysis of TOR in Omnet++

Carmelo Badalamenti <RollsAppleTree@gmail.com>

Mini Workshop on Security Framework 2006, Catania, December 12, 2006
"Security in Mobility"
Introduction: what's Anonymity?

What's Anonymity?

- Anonymity is the state of being anonymous, not known by others.

Who needs anonymity, and why?

- Journalists, Dissidents, Whistle blowers
- Censorship resistant publishers/readers
- Socially sensitive communicants
  - Chat rooms and web forums for abuse survivors, people with illnesses
- Law Enforcement
  - Anonymous tips or crime reporting
  - Surveillance and honeypots
Introduction: what's Anonymity?

Who needs anonymity, and why?

- **You:**
  - Where are you sending email (who is emailing you)
  - What web sites are you browsing
  - Where do you work, where are you from
  - What do you buy, what books do you read, ...
Introduction: what's Anonymity?

Who needs anonymity, and why?

- And yes, criminals:

  But they already have it.
  We need to protect everyone else.
**Introduction: what's Anonymity?**

Anonymity in the Internet:

Let Alice would communicate with Bob

**forward anonymity:**
no one could know who Alice is

Alice ➔ Bob

**reverse (or backward) anonymity:**
no one could know who Bob is

Bob ➔ Alice

Badalamenti C. <RollsAppleTree@gmail.com>  TOR & Omnet++
Introduction: How To Achieve Anonymity

Mainly used approach:

- Mixing
- Proxy
What's Anonymity?

How to achieve anonymity

Introduction: How To Achieve Anonymity

What does a mix do?

Randomly permutes and decrypts input.

Badalamenti C. <RollsAppleTree@gmail.com>  TOR & Omnet++
Introduction: How To Achieve Anonymity

What does a mix do?

Adversary can't say which cyphertext corresponds to a given message
What does a mix do?

Each server have a Key, they encrypt the message with layer of crypto like an onion.
Introduction: How To Achieve Anonymity

What does a mix do?

Server 1

\[ m1 \rightarrow m2 \rightarrow m3 \]

Server 2

\[ \text{decrypt and permute} \]

Server 3

\[ \text{decrypt and permute} \]

\[ \text{decrypt and permute} \]

\[ m2 \rightarrow m1 \rightarrow m3 \]

Badalamenti C. <RollsAppleTree@gmail.com>  TOR & Omnet++
Introduction: How To Achieve Anonymity

What does a mix do?

One honest server preserve anonymity

Badalamenti C. <RollsAppleTree@gmail.com>  TOR & Omnet++
Introduction: How To Achieve Anonymity

But if we need quick interaction?
Introduction: How To Achieve Anonymity

Anonymizing Proxy

- Communications appears to come from the proxy, not from the real hosts
- Advantage: Simple, Focuses lots of traffic for more anonymity
- Disadvantage: Single point of failure, compromise, attack
Let's use the advantage of both the approach:

- Use public key crypto to establish the circuit
- Use symmetric key crypto to move the data
  - Like SSL - TLS based proxies
- Distributed trust like mixes

**TOR: THE ONION ROUTER**
How Tor work: Circuit Setup

- Client proxy establish session key + circuit with the **Onion Router 1**

Diagram:

- Client Initiator
- Onion Router 1
How Tor work: Circuit Setup

- Client proxy establish session key + circuit with the **Onion Router 1**
- Proxy tunnels through that circuit to extend to **Onion Router 2**
How Tor work: Circuit Setup

- Client proxy establish session key + circuit with the **Onion Router 1**
- Proxy tunnels through that circuit to extend to **Onion Router 2**
- Etc ...

![Circuit Setup Diagram](image-url)
How Tor work: Circuit Setup

- Client proxy establish session key + circuit with the **Onion Router 1**
- Proxy tunnels through that circuit to extend to **Onion Router 2**
- Etc...
- Client application connect and communicate over the TOR circuit
How Tor work: Feature

- **Perfect forward anonymity**
  - Incremental building of circuit thought session key negotiation with each next node.
  - Periodic renewal of circuit

- We can multiplex more TCP streams in one circuit to reduce latency
How Tor work: Feature

- Leaky pipe topology (no end-to-end attack)

- TLS in every transaction between nodes
  - Can't modify data in transit
  - Impossibility of portray a Router
How Tor *in depth* works: Overview – What?

Tor is an Open Source Program, written in ANSI C. The project was in origin developed by the U.S. Army, then donated to the Open Source Community. It lives thanks to the care of million of people, and operate thanks to the bandwidth donated by normal people who set up a TOR node. *The more we are, the better anonymity we could have!!!*
How Tor work: Feature - How?

But ... how TOR can do this???

```c
/* Copyright (c) 2001 Matej Pfajfar. */
/* Copyright (c) 2001-2004, Roger Dingledine. */
/* Copyright (c) 2004-2006, Roger Dingledine, Nick Mathewson. */
/* See LICENSE for licensing information */
/* $Id: onion.c 8479 2006-09-24 17:20:41Z nickm $ */
const char onion_c_id[] =
"$Id: onion.c 8479 2006-09-24 17:20:41Z nickm $";

/**
 * \file onion.c
 * \brief Functions to queue create cells, and handle onion skin parsing and creation.
 **/

#include "or.h"

/** Type for a linked list of circuits that are waiting for a free CPU worker
to process a waiting onion handshake. */
typedef struct onion_queue_t {
  circuit_t *circ;
  time_t when_added;
  struct onion_queue_t *next;
} onion_queue_t;

/** 5 seconds on the onion queue til we just send back a destroy */
#define ONIONQUEUE_WAIT_CUTOFF 5

/** Global (within this file) variables used by the next few functions */
static onion_queue_t *ol_list=NULL;
static onion_queue_t *ol_tail=NULL;
/** Length of ol_list */
```
How Tor *in depth* works: Overview

Let's see it in depth!

```c
/* Copyright (c) 2001 Matej Pfajfar.  
 * Copyright (c) 2001-2004, Roger Dingledine.  
 * Copyright (c) 2004-2006, Roger Dingledine, Nick Mathewson. */  
/* See LICENSE for licensing information */  
/* $Id: onion.c 8479 2006-09-24 17:20:41Z nickm */ */  
const char onion_c_id[] =  
    "$Id: onion.c 8479 2006-09-24 17:20:41Z nickm $";

/**  
 * \file onion.c  
 * \brief Functions to queue create cells, and handle onionskin  
 * parsing and creation.  
 ***/

#include "or.h"

/** Type for a linked list of circuits that are waiting for a free CPU worker  
 * to process a waiting onion handshake. */
typedef struct onion_queue_t {  
circuit_t *circ;  
time_t when_added;  
struct onion_queue_t *next;
} onion_queue_t;

/** 5 seconds on the onion queue til we just send back a destroy */
#define ONIONQUEUE_WAIT_CUTOFF 5

/** Global (within this file) variables used by the next few functions */
static onion_queue_t *ol_list=NULL;  
static onion_queue_t *ol_tail=NULL;
/** Length of ol_list */
```
How Tor in depth works: Overview

Step 1: Alice’s Tor client obtains a list of Tor nodes from a directory server.
How Tor in depth works: Overview

Step 2: Alice’s Tor client picks a random path to destination server. **Green links** are encrypted, **red links** are in the clear.
How Tor *in depth* works: Overview

**Step 3**: If the user wants access to another site, Alice’s Tor client selects a second random path. Again, **green links** are encrypted, **red links** are in the clear.

- Alice
- Dave
- Jane
- Bob
How Tor *in depth* works: Cells

Indicate which virtual circuit is associated with the cell. In fact, we can create a different circuit for each connection we establish. In reality, it is better to multiplex more communication in one circuit.
How Tor *in depth* works: Cells

It holds the value of one of the command supported by the protocol. It can assume this values:

- 0 – **PADDING** (Padding)
- 1 – **CREATE** (Create a circuit)
- 2 – **CREATED** (Acknowledge create)
- 3 – **RELAY** (End-to-end data)
- 4 – **DESTROY** (Stop using a circuit)
- 5 – **CREATE_FAST** (Create a circuit, no PK)
- 6 – **CREATED_FAST** (Circuit created, no PK)
- 7 – **VERSIONS** (Negotiate versions)
- 8 – **NETINFO** (Time and MITM-revention)
How Tor *in depth* works: Cells

Contains the data we want to transmit to another node. The interpretation of 'Payload' depends on the type of the cell.

- **PADDING**: Payload is unused.
- **CREATE**: Payload contains the handshake challenge.
- **CREATED**: Payload contains the handshake response.
- **RELAY**: Payload contains the relay header and relay body.
- **DESTROY**: Payload contains a reason for closing the circuit.
How Tor *in depth* works: Cells - Create

- CircID
- Command
- OAEP padding
- Symmetric K
- 1\textsuperscript{st} part $g^x$
- 2\textsuperscript{nd} part $g^x$
- Padding

1 - CREATE

Optimal Asymmetric Encryption Padding

1\textsuperscript{st} part of Diffie Helman's key agreement Protocol

2\textsuperscript{nd} part of Diffie Helman's key agreement Protocol
How Tor *in depth* works: Cells - Create

1. CREATE

- CircID Command
- OAEP padding
- Symmetric K
  - 1\textsuperscript{st} part $g^x$

2. Padding
  - 2\textsuperscript{nd} part $g^x$

Optional Asymmetric Encryption Padding

- Encrypted with RSA
  - 1\textsuperscript{st} part of Diffie Hellman's key agreement Protocol
  - 2\textsuperscript{nd} part of Diffie Hellman's key agreement Protocol

Badalamenti C. <RollsAppleTree@gmail.com>  TOR & Omnet++
How Tor *in depth* works: Cells - Create

1. CREATE

- CircID Command
- OAEP padding
- Symmetric K
  - 1st part $g^x$
- 2nd part $g^x$

Padding

- Optimal Asymmetric Encryption Padding
- Encrypted with RSA
  - 1st part of Diffie Helman's key agreement Protocol
- Encrypted with Symmetric Key
  - 2nd part of Diffie Helman's key agreement Protocol
We make this choice for efficiency reason. Infact, this hybrid model make use of asymmetric crypto to grant a secure exchange for the key, that will be used to cyphè a big amount of data with better efficiency.
How Tor *in depth* works: Cells - Created

- CircID
- Command
- g^y value
- KH
- Padding

2 - CREATED

2^{nd} part of DH protocol

Hash (SHA1) of the DH Key. Demonstrate the knowledge of the entire DH Key

Badalamenti C. <RollsAppleTree@gmail.com>  TOR & Omnet++
The $g^{xy}$ computed value is used to derive $K$ of 100 bytes according to the follow scheme:

$$K = H(g^{xy} | [00] | H(g^{xy} | [01]) | ... H(g^{xy} | [04])$$

where | is the concatenation of its operand, $[NN]$ is a byte which value is NN and $H(x)$ is the SHA1 hash of x.

From $K$ will be consequently derived further value like derivative key ($KH$), forward digest ($Df$), backward digest ($Db$), forward key ($Kf$) and backward key ($Kb$) according the following schema: first 20 byte of $K$ are $KH$, bytes 21-40 are $Df$, bytes 41-60 are $Db$, 61-76 are $Kf$ and 77-92 are $Kb$. 
How Tor in depth works: Cells – Relay

Can assume the following values:

1 -- RELAY_BEGIN
2 -- RELAY_DATA
3 -- RELAY_END
4 -- RELAY_CONNECTED
5 -- RELAY_SENDME
6 -- RELAY_EXTEND
7 -- RELAY_EXTENDED
8 -- RELAY_TRUNCATE
9 -- RELAY_TRUNCATED
10 -- RELAY_DROP
11 -- RELAY_RESOLVE
12 -- RELAY_RESOLVED
How Tor in depth works: Cells – Relay

If this field is not equal to 0 the decrypted cell is routed to the next circuit's Onion Router, otherwise the payload content is processed by the same node.

Used to determine the cells that belongs to the same data stream.
How Tor in depth works: Cells – Relay

- CircID
- Command
- Rel command
- Recognized
- StreamID
- Digest
- Length

Dati

Encrypted with forward key R_i

Padding

Badalamenti C. <RollsAppleTree@gmail.com>  TOR & Omnet++
How Tor *in depth* works: Cells – Relay Extend

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CircID</td>
<td></td>
</tr>
<tr>
<td>Command</td>
<td></td>
</tr>
<tr>
<td>Recognized</td>
<td></td>
</tr>
<tr>
<td>StreamID</td>
<td></td>
</tr>
<tr>
<td>Digest</td>
<td></td>
</tr>
<tr>
<td>Address</td>
<td></td>
</tr>
<tr>
<td>Port</td>
<td></td>
</tr>
<tr>
<td>OAEP padding</td>
<td></td>
</tr>
<tr>
<td>Symmetric K</td>
<td></td>
</tr>
<tr>
<td>1st part $g^x$</td>
<td></td>
</tr>
<tr>
<td>2nd part $g^x$</td>
<td></td>
</tr>
<tr>
<td>Hash Pk</td>
<td></td>
</tr>
<tr>
<td>Padding</td>
<td></td>
</tr>
</tbody>
</table>

Payload of a CREATE Cell

3 - RELAY

6 - RELAY_EXTEND
How Tor *in depth* works: Cells – Relay Extend

```
<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CircID</td>
<td></td>
</tr>
<tr>
<td>Command</td>
<td></td>
</tr>
<tr>
<td>Rel command</td>
<td></td>
</tr>
<tr>
<td>Recognized</td>
<td></td>
</tr>
<tr>
<td>StreamID</td>
<td></td>
</tr>
<tr>
<td>Digest</td>
<td></td>
</tr>
<tr>
<td>Length</td>
<td></td>
</tr>
<tr>
<td>Address</td>
<td></td>
</tr>
<tr>
<td>Port</td>
<td></td>
</tr>
<tr>
<td>OAEP</td>
<td></td>
</tr>
<tr>
<td>padding</td>
<td></td>
</tr>
<tr>
<td>Symmetric K</td>
<td></td>
</tr>
<tr>
<td>1st part g^x</td>
<td></td>
</tr>
<tr>
<td>2nd part g^x</td>
<td></td>
</tr>
<tr>
<td>Hash Pk</td>
<td></td>
</tr>
<tr>
<td>Padding</td>
<td></td>
</tr>
</tbody>
</table>
```

3 - RELAY

6 - RELAY_EXTEND

Encrypted with RSA with public key of the Onion Router that have to extend the virtual circuit

Encrypted with Symmetric Key

Badalamenti C. <RollsAppleTree@gmail.com>  TOR & Omnet++
How Tor *in depth* works: Cells – Relay Extend

CircID
Command
Rel command
Recognized
StreamID
Digest
Length
Address
Port
OAEP padding
Symmetric K
1st part g^x
2nd part g^x
Hash Pk
Padding

3 - RELAY

Encrypted with forward key R_i
How Tor *in depth* works: Cells – Relay Extended

Payload of a CREATED Cell

- CircID
- Command
- Rel command
- Recognized
- StreamID
- Digest
- Length

$g^y$

KH

Padding
How Tor *in depth* works: Cells – Relay Extended

- CircID
- Command
- Rel command
- Recognized
- StreamID
- Digest
- Length

$g^y$

Encrypted with backward key $R_i$
How Tor in depth works: Circuit

The client OP choose the exit node

OP

R_1

R_2
How Tor *in depth* works: Circuit
How Tor *in depth* works: Circuit

OP => CREATE => R_1
How Tor in depth works: Circuit

Circuit

Command

OAEP padding

Symmetric K

1st part g^x

Padding

2nd part g^x

Decrypted with public key

OP

R_1

R_2

Badalamenti C. <RollsAppleTree@gmail.com>  TOR & Omnet++
How Tor *in depth* works: Circuit

- **OP**
- **R_1**
- **R_2**

### Circuit Description

- Circuit ID
- Command
- OAEP padding
- Symmetric key
- 1st part $g^x$
- 2nd part $g^x$
- Padding

Badalamenti C. <RollsAppleTree@gmail.com>  TOR & Omnet++
How Tor *in depth* works: Circuit

Decryption with symmetric key

Padding

1\textsuperscript{st} part g^x

2\textsuperscript{nd} part g^x

Symmetric K

OAEP padding

Command

CircuitID

R_1

R_2

OP
How Tor *in depth* works: Circuit
How Tor in depth works: Circuit

Calculate $g^{xy}$ and derive $K$
How Tor *in depth* works: Circuit

Send the CREATED Cell

OP → R_1 → R_2
How Tor in depth works: Circuit

Calculate $g^{xy}$ and derive $K$.

Compare first 20 bytes of $K$ derivative key with $KH$ received.
How Tor *in depth* works: Circuit

1. OP
2. $R_1$
3. Send the EXTEND_RELAY Cell

**Encrypted with forward key $R_i$**

**Diagram Details:**
- **CircuitID**
- **Rel command**
- **Recognized**
- **StreamID**
- **Length**
- **Address**
- **Port**
- **OAEP padding**
- **Symmetric K**
- **1\text{st part} \ g^x**
- **2\text{nd part} \ g^x**
- **Hash Pk**
- **Padding**

**Contact:**
Badalamenti C. <RollsAppleTree@gmail.com>  TOR & Omnet++
How Tor in depth works: Circuit

Decipher with forward key

Badalamenti C. <RollsAppleTree@gmail.com> TOR & Omnet++
How Tor **in depth** works: Circuit

Control Relay_cmd=6, RELAY_EXTEND

Control Recognized -> 0. So process the cell's content.

Control IPv4 and Port of the node in which extend circuit
How Tor in depth works: Circuit

Extract the Onion skin from extend cell's payload

OP

R_1

R_2

CircID
Command
Rel command
Recognized
StreamID
Digest
Length
Address
Port
OAEP
padding
Symmetric K
1st part g^x

2nd part g^x
Hash Pk
Padding
How Tor *in depth* works: Circuit

Insert the extracted payload in a CREATE cell
How Tor *in depth* works: Circuit

Overview

Introduction

How TOR works?

How TOR *in depth* works?

Open Problems

Web traffic

**Overview**

**Cells**

**Circuit**

**Web traffic**

---

**Badalamenti C. <RollsAppleTree@gmail.com>**

TOR & Omnet++
How Tor *in depth* works: Circuit

- OP
- R_1
- R_2

Decrypt with SimmKey

1st part $g^x$

2nd part $g^x$

Padding

CircID
Command
OAEP
Symmetric
K

Badalamenti C. <RollsAppleTree@gmail.com>  TOR & Omnet++
How Tor in depth works: Circuit

Send to R_1 a CREATED cell
How Tor *in depth* works: Circuit

Extract the CREATED cell's payload
How Tor in depth works: Circuit

Calculate the $g^{xy}$ value and derivative key.

$g^y$

KH

Padding
How Tor in depth works: Circuit

The circuit is extended

OP

R_1

R_2

Badalamenti C. <RollsAppleTree@gmail.com>  TOR & Omnet++
How Tor *in depth* works: Web traffic

Tor use the socks4 proxy interface.

SOCKS4 is an Internet protocol that allows client-server applications to transparently use the services of a network firewall.

So we can send thought TOR almost ALL TCP TRAFFIC!!!

We only need to “torify” the applications we would use.

Often it would mean simply set up proxy to localhost port 9050.
There's no evident problems in TOR implementation. It assure that all the tenets enunciated in the TOR design paper are respected.

But ... there are other technologies that, used together, cause

- a web browser using TOR to “Phone Home” outside the TOR network
- a web browser using TOR to “Phone home” inside the TOR network, and deliver uniquely-identifying about the client, such as the computer's hostname and IP address.
Open Problems: Phone Home

Introduction
How TOR works?
How TOR in depth works?

Phone Home

Open Problems

Badalamenti C. <RollsAppleTree@gmail.com>  TOR & Omnet++
Open Problems: Phone Home

SOLUTIONS?

Yes

Use “NoScript” plugin for Firefox...

nothing for IE (did you expect this?)
References:

- Carmelo Badalamenti - “TOR: Design e Simulazione”, Relatore prof. G. Bella, Relazione Stage

- Roger Dingledinen - “Tor: Anonymous Communications for the United States Department of Defence..and you.” July 2005

- Andrei Serjantov - “On the anonymity of anonymity systems”, October 2004

- Roger Dingledine, Nick Mathewson - “Tor Protocol Specification”, October 2006

- Roger Dingledine, Nick Mathewson, Paul Syverson - “Tor: The Second-Generation Onion Router”, November 2005

- VVAA - Wikipedia, all time

- Fortconsult – “Practical Onion hacking, Finding the real address of TOR clients”, October 2006