Covert Messaging Through TCP Timestamps

to be published in the proceedings of the Privacy Enhancing Technologies Workshop, April 14-15 2002

John Giffin
Rachel Greenstadt
Peter Litwack
Richard Tibbetts
Overview

• Covert Channels: What and Why

• Protocol

• Implementation details and status

• Future work and conclusions
Covert Channel - a communications channel which allows information to be transferred in a way that violates a security policy.

It can provide:

- Unobservable communication
- Censorship resistance
Covert Channels – Why?

• Provide privacy when cryptography is not allowed
  – China Totalistan
  – Increasingly subject to monitoring in the US (carnivore, etc)

• Hide the fact that secret communication is going on
  – Prisoner’s Problem
Properties Required of Effective Covert Channels

• Plausibility

• Randomness

• Indispensibility
Why TCP timestamps?

• TCP is standard and ubiquitous and the timestamp option is nearly always employed.

• Low order bits of timestamps will look random for slow connections, so encrypted data will be indistinguishable from unmodified timestamps.

• Other parts of the tcp headers don’t have the appearance of randomness so if an adversary is looking, they will notice.
Adversary Model

The system is broken when the adversary

- Detects that data is being sent over the covert channel

- Can differentiate between a normal TCP connection and one sending covert data

- Can deny service to the users of the covert channel while still allowing service to the users of unmodified TCP connections.
The Adversary can:

- Passively observe all TCP connections over the network

- Modify and delete some number of packets

However, the Adversary must allow most traffic to pass through.
Related Work

• Covert Channels in TCP
  – Craig Rowland “Covert Channels in the TCP/IP Protocol Suite”
  – UC Davis Denial of Service Project
  – IP Checksum Covert Channels and Selected Hash Collision

• Using Timing Information in Covert Channels
  – Abad, timing in hardware channels
  – Miller, Moscowitz, “Simple Timing Channels”
General Covert Channels Background

- Simmons - Initial Work

- McHugh - Covert Channel Analysis
Design Goals

- Send data

- Be undetectable

- Require as little as possible from the channel
Channel Characteristics

Worst Case Scenario - bitwise lossy, unacknowledged, bits sent required (by adversary) to pass certain statistical tests
We assume:

- statistical test is randomness

- each bit has a nonce which arrives intact

Additional Assumptions

- sender and receiver have access to such a channel and a shared secret

- best effort datagram service is sufficient
Protocol

- Send one bit of our message block per bit of the channel

- Choose bit to send based on a keyed cryptographic hash of the nonce

- Derive an independent key bit $k$ from the cryptographic hash of the nonce and send the xor of $k$ and the plaintext message bit.

- The sender assumes a block has been transmitted after it has sent each bit $o$ times, and the receiver computes a checksum to determine if it has received a valid block.
Using TCP

- By imposing slight delays on the processing of selected TCP packets, we can modify the low order bits of their timestamps.

- On most connections, the low order bits are randomly distributed.

- The rest of the TCP headers provide a nonce which is nearly free from repetition.
TCP headers

| 00  | 01  | 02  | 03  | 04  | 05  | 06  | 07  | 08  | 09  | 10  | 11  | 12  | 13  | 14  | 15  | 16  | 17  | 18  | 19  | 20  | 21  | 22  | 23  | 24  | 25  | 26  | 27  | 28  | 29  | 30  | 31 |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| Source Port | Destination Port |
| Sequence Number |
| Acknowledgement Number |
| Data Offset | Reserved | ECN | Control Bits | Window |
| Checksum | Urgent Pointer |
| Options andPadding | Kind = 8 | Length = 10 |
| Timestamp Value |
| Timestamp Echo Reply |
| Data |
Implementation Problems

- The channel delivers our bits out of order and can drop individual bits randomly.
  - Many TCP connections are short (HTTP GETs etc.).
  - Data is acknowledged not packets.

- Merely hiding the data is insufficient. It must be encrypted. Otherwise an adversary that knew our protocol could detect covert data.

- We can never lower the value of a timestamp. Otherwise, an adversary could detect this based upon a conflict with an application layer timestamp.
Sending Data

Packet Header | Secret Key

SHA1

Hash of Headers and Key

bits 0-7
Index

bits 8
KeyBit

Current Message Block

Plain Text Bit

Cipher Text Bit
Receiving Data
Rewriting the Timestamp

Start

LSB of timestamp = cipher text bit?

YES

Done

NO

Increment timestamp

Did the high order bits change?

YES

Recompute cipher text bit

NO

NO

YES

Start

LSB of timestamp = cipher text bit?

YES

Done

NO

Increment timestamp

Did the high order bits change?

YES

Recompute cipher text bit

NO
Security Evaluation

• Given a random oracle model for the hash function used by the sender, the key bit will be a random number if the packet headers do not collide.

• Odds of such collisions are low

• If the key bit is random, plaintext bit xor key bit is random.

• If headers collide, one bit of information is revealed about the two bits of plaintext in those two packets, the secret key is not compromised.

• The connection must be sufficiently slow so that the low order bit of the timestamp is random.
Implementation Specifics

- Linux 2.4.9

- Export tcp_rewrite_timestamp function pointer so we can work from a module.

- Use a module to speed testing, ease deployment.

- Ioctl in key and destination, and write 32 byte blocks to the device.

- Slowing down tcp streams (not yet implemented).

- Receiver uses libpcap (from tcpdump). Runs on many platforms.
Performance Analysis

• Some errors in the paper, this section is being worked on

• Linux timestamps measured in 10 ms jiffies, expected delay time to send a bit covertly is 3.75 ms.

• As a result, can send $n$ bits in $3.75 \times n$ ms.

• Expected number of packets to send 256 bits (once) is 1300.
Conclusion

- Protocol applicable to low bandwidth, lossy covert channels

- Implementation in progress