Covert channels in TCP/IP: attack and defence

The creation and detection of TCP/IP steganography for covert channels and device fingerprinting

Steven J. Murdoch and Stephen Lewis

http://www.cl.cam.ac.uk/users/{sjm217, srl32}

Computer Laboratory
University of Cambridge

Security Group seminar series
31 January 2006
Scenario

Alice

Bob

Walter
Threat model

• Walter is a *passive warden*, trying to detect unauthorised communication from Alice to Bob
  • To break this policy, Alice uses a *covert channel*
• Walter knows which OS Alice is running
• Alice sends message hidden in *cover-text*
• The cover-text must be received intact
• Alice requires *indistinguishability*
• Subject to these constraints, Alice would like to maximise the available *bandwidth*
• Techniques to achieve these goals are known as *steganography*
Outline

- Covert channel opportunities in TCP/IP
- Detecting Nushu
- Undetectable steganography
- Extensions to fingerprinting
Protocol stack

Ethernet

IP

TCP

GET /congress/2005/ HTTP/1.0
...
Seq: 3622491521, Ack: 1380426457
From:  port 37633, To: port 80, ...
Type: TCP, ToS: 0, Flags: 4, ID:6801,
From: 128.232.0.20, To: 213.73.91.29, ...
Type: IP, From: 00:11:11:0e:08:ea,
To: 00:00:0c:07:ac:01

 Seq: 3622491521, Ack: 1380426457
 From: port 37633, To: port 80, ...
 GET /congress/2005/ HTTP/1.0
 ...
Why TCP/IP

• Lower levels (Ethernet) will not reach Bob
• Alice might not be able to control which applications she runs
• So higher level protocols might not be available
• Almost all network applications use TCP/IP
• So Alice can use this without raising suspicion
<table>
<thead>
<tr>
<th>Field</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>4</td>
</tr>
<tr>
<td>IHL</td>
<td>4</td>
</tr>
<tr>
<td>Type of Service</td>
<td>8</td>
</tr>
<tr>
<td>Total Length</td>
<td>16</td>
</tr>
<tr>
<td>Identification</td>
<td>16</td>
</tr>
<tr>
<td>Flags</td>
<td>16</td>
</tr>
<tr>
<td>Fragment Offset</td>
<td>16</td>
</tr>
<tr>
<td>Time to Live</td>
<td>8</td>
</tr>
<tr>
<td>Protocol</td>
<td>8</td>
</tr>
<tr>
<td>Header Checksum</td>
<td>16</td>
</tr>
<tr>
<td>Source Address</td>
<td>32</td>
</tr>
<tr>
<td>Destination Address</td>
<td>32</td>
</tr>
<tr>
<td>Options</td>
<td>128</td>
</tr>
<tr>
<td>Padding</td>
<td>128</td>
</tr>
</tbody>
</table>
Fragmentation

- If IP packets are too large to fit into the lower layer, they can be fragmented
- Data could be encoded by changing
  - The size of fragments
  - The order of fragments
- IP gives no guarantees of in-order delivery
  - So IP packets can be re-ordered
- All these are predictable, so while the cover-text will get through, Walter can see the steganography
Seldom used IP options

- **ToS**: Used for altering quality of service
  - Almost never used, so easily detectable
- **Flags**: Used to signal fragmentation
  - Predictable based on context, so easily detectable
- **IP options** (different from TCP options)
  - Seldom used now, so easily detectable
IP ID

- Unique value associated with each IP packet
- Used to re-assemble fragments
- Commonly implemented (e.g. Linux) as a per-destination counter
  - This is to prevent *idle-scanning*
  - Linked to TCP (details later)
- Violating this would result in easy detection
- Respecting this dramatically reduces bandwidth
### TCP Header

<table>
<thead>
<tr>
<th>Field</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source Port</td>
<td>Destination Port</td>
</tr>
<tr>
<td><strong>Sequence Number</strong></td>
<td></td>
</tr>
<tr>
<td>Acknowledgement Number</td>
<td></td>
</tr>
<tr>
<td>Offset</td>
<td>Reserved</td>
</tr>
<tr>
<td>Flags</td>
<td>Window</td>
</tr>
<tr>
<td>Checksum</td>
<td>Urgent Pointer</td>
</tr>
<tr>
<td>Options (including timestamp)</td>
<td>Padding</td>
</tr>
</tbody>
</table>

The TCP header contains fields for source and destination ports, sequence and acknowledgement numbers, flags, window, checksum, urgent pointer, options, and padding.
TCP timestamp

- Option available in TCP packets which allows hosts to measure round-trip-time
- Available in most modern operating systems, but off by default in Windows
- Stores the time packet was sent, according to a 1 Hz–1 kHz clock
- Predictable, but packets can be delayed to force this value to be odd or even, allowing 1 bit per packet to be sent
- With high-bandwidth connections, where many packets with the same timestamp are normally sent out, this scheme can be detected
TCP initial sequence number

- When TCP connection is first built, each side picks an initial sequence number (ISN), used for reliability and flow control.
- To prevent IP address spoofing, this number should be hard to guess.
- While there have been problems in the past, all modern operating systems now do this.
- It is large (32 bits), and because it is unpredictable to outsiders, including Walter, this field is the most useful for steganography.
- However using it properly is far from simple.
Nushu

- Presented by Joanna Rutkowska at 21C3
- Steganographic covert channel implemented for Linux
- Also includes error recovery
- Uses clever kernel tricks to hide from local detection (outside the scope of this talk)
- Replaces ISN with encrypted message (so should look random)
Catching Nushu

Unmodified Linux

Nushu

Current ISN

Next ISN

9.20e+08
9.30e+08
906800000
906900000

9.30e+08
9.20e+08
906900000
906800000

2e+09
3e+09
4.292e+09
15840000
2e+09
3e+09
4.28e+09
11490000
Nushu encryption

Source

<table>
<thead>
<tr>
<th>Port</th>
<th>Address</th>
</tr>
</thead>
</table>

\[ \oplus \]

Destination

<table>
<thead>
<tr>
<th>Port</th>
<th>Address</th>
</tr>
</thead>
</table>

"NU"

\[ DES \]

\[ \oplus \]

Message

New ISN
Nushu encryption

Frequent duplications

Source
Port Address
⊕

Destination
Port Address

"NU"

DES

Key

Message

New ISN
Attacking the cryptography

- There will be frequent duplications of this
  - Source IP is fixed; destination IP will not vary much
  - Destination port will not vary much and source port does not use anywhere near all of the $2^{16}$ possibilities
-Whenever there is a duplication, the output of DES will be the same
- $X = (M_1 \oplus K) \oplus (M_2 \oplus K) = M_1 \oplus M_2$
- If $M_1 = M_2$ then $X = 0$
- Even if $M_1 \neq M_2$, $X$ will still show patterns
Nushu revealed

Unmodified Linux

- Next ISN with same IV
- Current ISN

Nushu

- Current ISN
Linux ISN generation

<table>
<thead>
<tr>
<th>Source IP</th>
<th>Dest. IP</th>
<th>S. Port</th>
<th>D. Port</th>
</tr>
</thead>
</table>

Concatenate 32 random bits

R-MD4 block: 256 random bits

Take bits 32–63

c
replace top byte with rekey counter

and add 32-bit time ($\mu$s) + T
**Linux ISN generation**

<table>
<thead>
<tr>
<th>Source IP</th>
<th>Dest. IP</th>
<th>S. Port</th>
<th>D. Port</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>R</td>
<td></td>
</tr>
</tbody>
</table>

Concatenate 32 random bits

---

| R-MD4 block: 256 random bits | Take bits 32–63 | replace top byte with rekey counter | \( + T \) |
## Linux ISN generation

<table>
<thead>
<tr>
<th>Source IP</th>
<th>Dest. IP</th>
<th>S. Port</th>
<th>D. Port</th>
</tr>
</thead>
</table>

- Concatenate 32 random bits
- Replace top byte with rekey counter
- And add 32-bit time (µs) + T

**Diagram:**

- R-MD4 block: 256 random bits
- Concatenate 32 random bits
### Linux ISN generation

**Table:**

<table>
<thead>
<tr>
<th>Source IP</th>
<th>Dest. IP</th>
<th>S. Port</th>
<th>D. Port</th>
</tr>
</thead>
</table>

**Process:**

1. Concatenate 32 random bits
2. R-MD4 block: 256 random bits
3. Take bits 32–63

**Details:**

- Replace top byte with rekey counter
- Add 32-bit time (µs) + T
Linux ISN generation

<table>
<thead>
<tr>
<th>Source IP</th>
<th>Dest. IP</th>
<th>S. Port</th>
<th>D. Port</th>
</tr>
</thead>
</table>

1. Concatenate 32 random bits

2. R-MD4 block: 256 random bits

3. Take bits 32–63

4. Replace top byte with rekey counter...
## Linux ISN generation

### Table

<table>
<thead>
<tr>
<th>Source IP</th>
<th>Dest. IP</th>
<th>S. Port</th>
<th>D. Port</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Diagram

1. **Concatenate 32 random bits**
2. **R-MD4** (block: 256 random bits)
3. **Take bits 32–63**
4. **c + T**

   - ...and add 32-bit time ($\mu s$)
Patterns with the Linux ISN

- Within a rekey period, multiple connections with the same source/destination, IP address/port number will have the same input to MD4
- The difference between the ISNs for two connections will be the time difference in microseconds
- The Nushu problem could be prevented by not repeating IVs
  - Use more randomness (hash as much of the header as possible)
  - In case there is a collision, use a 32-bit block cipher
  - Never send the same plaintext with the same IV
- This would hide any patterns, but Linux introduces patterns of its own
Better cryptography doesn’t help

Unmodified Linux

Random ISN
Steganography for Linux

- Replace the lower 3 bytes with our data
- Restore rekey counter
- Add one to rekey counter if carry bit is needed
  - Subtract current time in microseconds (mod $2^{32}$) from our data
  - If this is negative, add one to they rekey counter, otherwise leave it alone
- Patch up the checksum and IP ID (depends on ISN)
- Can use the ACK from the remote host to get a good idea of whether the SYN packet was lost
- Freshness is achieved by xoring the plaintext with a hash of Source/Destination IP and Source/Destination Port
- One bit is reserved to cope with potential collisions
OpenBSD

Counter

Block Cipher 1024 bit random key

RC4 pseudorandom
OpenBSD

Counter

Block Cipher

1024 bit random key

RC4 pseudorandom

$r$

0
Steganography with OpenBSD

- Can code directly into the bottom 15 bits of the ISN (pre-shared key, with hash of other header fields for freshness)
- Need to code arbitrary data (with redundancy) onto a pseudorandom sequence of integers between 0 and $2^{15}$
- For reliability, Bob needs to be able to cope with the loss of some of the elements in the sequence
- Elements of the sequence are encrypted using a block cipher before transmission, and thus appear exactly as a pseudorandom sequence to the warden
Clock skew (TCP timestamps)
Timing information from ISNs

- All computers have a clock crystal to measure time, but imperfections cause some to run faster or slower than they should.
- This error is very stable over time, and so can acts as an identity for a computer.
- Even if the computer changes IP address or moves, its clock skew will stay the same and allows the computer to be tracked.
- There are several ways to extract clock skew information remotely.
  - TCP timestamp clocks run at 100 Hz or 1 kHz on Linux.
  - ICMP timestamp clocks run at 1 kHz.
  - On Linux, the TCP ISN clock runs at 1 MHz.
Clock skew (ICMP timestamps)
Timing patterns in the ISN

Unmodified Linux

Random ISN
Clock skew (TCP ISN)
• Many proposed steganography schemes are detectable
• The common flaw is to assume that fields that can be random are random
• In fact, many fields in protocols are not random – sometimes for good reason, sometimes just through chance
• To build undetectable steganography schemes, you must examine exactly how fields are generated, before you can modify it safely
• If physical device fingerprinting is a concern, there are sources of time information which you might not expect