Prefix- and Lexicographical-order-preserving IP Address Anonymization

Matúš Harvan

International University Bremen
Bremen, Germany

2006 IEEE/IFIP Network Operations and Management Symposium
Why do we need SNMP measurements?

- many speculations on how SNMP is being used in real world production networks and how it performs
- no systematic measurements have been performed and published so far
- comparative studies based on assumed usage, lacking experimental evidence
- important to understand impact on network and devices while
  - improvements to SNMP
  - designing new management protocols
SNMP Questions

- usage of protocol operations
- message size distributions
- response times distributions
- periodic vs. aperiodic traffic
- trap-directed polling
- usage of obsolete objects (e.g. ipRouteTable)
- more questions in draft-schoenw-nrmg-snmp-measure-01.txt
Why do we need (IP) anonymization?

- required to obtain and analyze SNMP traces from several production networks
- necessary to analyze SNMP payload, not just headers
- traces need to be anonymized (management traffic naturally contains sensitive data)
- traces need to retain enough information after anonymization
- IP address prefix-relationships important (routing)
- SNMP imposes an additional constraint of preserving the lexicographical ordering
Prefix-preserving IP address anonymization

prefix-preserving anonymization solved by Crypto-PAn[1, 2]

- canonical form for all prefix-preserving anonymization functions
- using cryptography (AES) for anonymization
- working implementation
Prefix-preserving Anonymization[1]

Definition

Two IP addresses \(a = a_1 a_2 \ldots a_n\) and \(b = b_1 b_2 \ldots b_n\) share a \(k\)-bit prefix (\(0 \leq k \leq n\)) if \(a_1 a_2 \ldots a_k = b_1 b_2 \ldots b_k\) and \(a_{k+1} \neq b_{k+1}\) when \(k < n\).

Definition

An anonymization function \(F\) is defined as one-to-one function from \(\{0, 1\}^n\) to \(\{0, 1\}^n\).

Definition

An anonymization function \(F\) is prefix-preserving if given two IP addresses \(a\) and \(b\) that share a \(k\)-bit prefix, \(F(a)\) and \(F(b)\) share a \(k\)-bit prefix as well.
### Prefix-preserving Anonymization Example

<table>
<thead>
<tr>
<th>Original IP addresses</th>
<th>Anonymized IP addresses</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>IP1:</strong> 10.12.3.5</td>
<td><strong>F(IP1):</strong> 117.16.14.250</td>
</tr>
<tr>
<td>(00001010.0001100.0000011.00000101)</td>
<td>(01110101.00010000.00011110.11111010)</td>
</tr>
<tr>
<td><strong>IP2:</strong> 10.16.220.3</td>
<td><strong>F(IP2):</strong> 117.0.92.115</td>
</tr>
<tr>
<td>(00001010.0001000.1101100.0000011)</td>
<td>(01110101.00000000.01011100.01110011)</td>
</tr>
</tbody>
</table>
### Prefix-preserving Anonymization Example

<table>
<thead>
<tr>
<th>Original IP addresses</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>IP1: 10.12.3.5</td>
<td>(00001010.00001100.00000011.000000101)</td>
</tr>
<tr>
<td>IP2: 10.16.220.3</td>
<td>(00001010.00010000.11011100.00000011)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Anonymized IP addresses</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>( F(\text{IP1}) ): 117.16.14.250</td>
<td>(01110101.00010000.00001110.11111010)</td>
</tr>
<tr>
<td>( F(\text{IP2}) ): 117.0.92.115</td>
<td>(01110101.00000000.01011100.01110011)</td>
</tr>
</tbody>
</table>
Theorem

Let $f_i$ be a function from $\{0, 1\}^i$ to $\{0, 1\}$ for $i = 1, 2, \ldots, n - 1$ and $f_0$ be a constant function. Let $F$ be a function from $\{0, 1\}^n$ to $\{0, 1\}^n$ defined as follows. Given $a = a_1 a_2 \ldots a_n$, let

$$F(a) := a'_1 a'_2 \ldots a'_n$$

where $a'_i = a_i \oplus f_{i-1}(a_1, a_2, \ldots, a_{i-1})$ and $\oplus$ is the exclusive-or operation, for $i = 1, 2, \ldots, n$. Then $F$ is a prefix-preserving anonymization function and every prefix-preserving anonymization function necessarily takes this form.
**Address Tree**

Figure: Original address tree

Figure: Anonymization function

Figure: Anonymized address tree
Lexicographical order on IP addresses

Definition

Let \( a = a_1a_2 \ldots a_n \) and \( b = b_1b_2 \ldots b_n \) be two IP addresses (of the same length) where \( a_i \)'s and \( b_i \)'s are bits. Then a lexicographic ordering \(<^l\) is defined by

\[
a <^l b \iff a_1a_2 \ldots a_n <^l b_1b_2 \ldots b_n
\]

\[
\iff (\exists m > 0)(\forall i < m)(a_i = b_i) \land (a_m < b_m)
\]

Definition

An anonymization function \( F \) is lexicographical-order-preserving if given two IP addresses \( a \) and \( b \) we have

\[
a <^l b \Rightarrow F(a) <^l F(b)
\]
### Lexicographical-order-preserving Anonymization Example

<table>
<thead>
<tr>
<th>Original IP addresses</th>
<th>Anonymized IP addresses</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>IP1:</strong> 10.12.3.5</td>
<td><strong>F(IP1):</strong> 117.0.14.250</td>
</tr>
<tr>
<td><strong>IP2:</strong> 10.16.220.3</td>
<td><strong>F(IP2):</strong> 117.16.92.115</td>
</tr>
</tbody>
</table>

Original IP addresses:
- **IP1:** 10.12.3.5 (00001010.00010000.00000011.00000101)
- **IP2:** 10.16.220.3 (00001010.00010000.11011100.00000011)

Anonymized IP addresses:
- **F(IP1):** 117.0.14.250 (01110101.00000000.00001110.11111010)
- **F(IP2):** 117.16.92.115 (01110101.00010000.01011100.01110011)
Lexicographical-order-preserving Anonymization Example

<table>
<thead>
<tr>
<th>Original IP addresses</th>
<th>Anonymized IP addresses</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>IP1:</strong> 10.12.3.5</td>
<td>(F(\text{IP1}): 117.0.14.250)</td>
</tr>
<tr>
<td><strong>IP2:</strong> 10.16.220.3</td>
<td>(F(\text{IP2}): 117.16.92.115)</td>
</tr>
</tbody>
</table>
Theorem

Let $f_i$, $f'_i$ be functions from $\{0, 1\}^i$ to $\{0, 1\}$ for $i = 1, 2, \ldots, n - 1$ and $f_0, f'_0$ be constant functions. Let $F$ be a function from $\{0, 1\}^n$ to $\{0, 1\}^n$ defined as follows. Given $a = a_1 a_2 \ldots a_n$, let

$$F(a) := a'_1 a'_2 \ldots a'_n$$

$$a'_i = a_i \oplus f'_{i-1}(a_1, a_2, \ldots, a_{i-1})$$

$$f'_i(a_1, a_2, \ldots, a_i) = f_i(a_1, a_2, \ldots, a_i)$$

$$\land \neg (\text{used}_{i+1}(a_1, a_2, \ldots, a_i, 0) \land \text{used}_{i+1}(a_1, a_2, \ldots, a_i, 1))$$

for $i = 1, 2, \ldots, n$. Then we claim $F$ is a prefix-preserving and lexicographical-order-preserving anonymization function.
Idea
determines if any IP addresses in the subtree below the $a_i$ bit are used

Definition
Let $\text{used}_i$ be a function from $\{0, 1\}^i$ to $\{0, 1\}$ for $i = 1, 2, \ldots, n$. $\text{used}_i$ is defined recursively as

$$
\text{used}_i(a_1a_2\ldots a_i) = \text{used}_{i+1}(a_1a_2\ldots a_i0) \lor \text{used}_{i+1}(a_1a_2\ldots a_i1)
$$

$\text{used}_n(a_1a_2\ldots a_n)$ is true if the IP address $a_1a_2\ldots a_i$ is in the traffic trace and false otherwise.
Figure: Prefix-preserving only anonymization function ($f_i$)

Figure: Bits that can be flipped

Figure: Prefix- and lexicographical-order-preserving anonymization function ($f'_i$)
implemented as a C library *libanon*

works for both IPv4 and IPv6

lexicographical-order-preserving anonymization of other data types as well

- MAC addresses
- strings
- integers

being integrated with snmpdump package (conversion of snmp traces from pcap format to xml)
• runtime generally acceptable for offline analysis as long as the binary tree data structure fits into main memory
• memory consumption increases significantly faster for IPv6
## IPv4 - Memory

<table>
<thead>
<tr>
<th>number of IP addresses</th>
<th>number of nodes</th>
<th>measured memory footprint</th>
<th>theoretical memory requests</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>3 212 KB</td>
<td>16 B</td>
</tr>
<tr>
<td>1</td>
<td>33</td>
<td>3 220 KB</td>
<td>32 B</td>
</tr>
<tr>
<td>10</td>
<td>301</td>
<td>3 220 KB</td>
<td>4 KB</td>
</tr>
<tr>
<td>100</td>
<td>2 646</td>
<td>3 220 KB</td>
<td>41 KB</td>
</tr>
<tr>
<td>1 000</td>
<td>23 182</td>
<td>3 744 KB</td>
<td>362 KB</td>
</tr>
<tr>
<td>10 000</td>
<td>199 080</td>
<td>7 836 KB</td>
<td>3 110 KB</td>
</tr>
<tr>
<td>100 000</td>
<td>1 656 713</td>
<td>42 024 KB</td>
<td>25 886 KB</td>
</tr>
</tbody>
</table>

**Table:** Memory footprint for IPv4 anonymization
## IPv6 - Memory

<table>
<thead>
<tr>
<th>number of IP addresses</th>
<th>number of nodes</th>
<th>measured memory footprint</th>
<th>theoretical memory requests</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>3 212 KB</td>
<td>16 B</td>
</tr>
<tr>
<td>1</td>
<td>129</td>
<td>3 216 KB</td>
<td>2 KB</td>
</tr>
<tr>
<td>10</td>
<td>1 248</td>
<td>3 216 KB</td>
<td>19 KB</td>
</tr>
<tr>
<td>100</td>
<td>12 143</td>
<td>3 480 KB</td>
<td>189 KB</td>
</tr>
<tr>
<td>1 000</td>
<td>118 189</td>
<td>5 860 KB</td>
<td>1 846 KB</td>
</tr>
<tr>
<td>10 000</td>
<td>1 147 052</td>
<td>30 012 KB</td>
<td>17 922 KB</td>
</tr>
<tr>
<td>100 000</td>
<td>11 080 902</td>
<td>262 860 KB</td>
<td>173 139 KB</td>
</tr>
</tbody>
</table>

**Table:** Memory footprint for IPv6 anonymization
<table>
<thead>
<tr>
<th>number of IP addresses</th>
<th>runtime IPv4</th>
<th>runtime IPv6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.01 s</td>
<td>0.01 s</td>
</tr>
<tr>
<td>10</td>
<td>0.01 s</td>
<td>0.01 s</td>
</tr>
<tr>
<td>100</td>
<td>0.01 s</td>
<td>0.02 s</td>
</tr>
<tr>
<td>1 000</td>
<td>0.03 s</td>
<td>0.14 s</td>
</tr>
<tr>
<td>10 000</td>
<td>0.15 s</td>
<td>1.36 s</td>
</tr>
<tr>
<td>100 000</td>
<td>1.43 s</td>
<td>13.4 s</td>
</tr>
</tbody>
</table>

**Table:** Runtime of IPv4 and IPv6 anonymization
suitable IP address anonymization schema found, rigorously proven to be correct and implemented in the form of a C library \textit{libanon}

our contribution consists of an extension to the existing prefix-preserving cryptography-based anonymization scheme used in \textit{Crypto-PAn}

further work:

- develop a tool for anonymization of SNMP traces including complete SNMP payload
- analyze anonymized SNMP traffic traces
- improve memory consumption of the \textit{libanon} implementation


How hard is it for an attacker to recover the original addresses from an anonymized trace?

- **prefix-preserving**
  Due to the prefix-preserving property, compromising one IP address compromises a prefix of other addresses as well.

- **lexicographical-order-preserving**
  In case a complete subnet of the address space is used, host portion of the address cannot be anonymized.

- IPv6 address space larger than IPv4, so more secure anonymization possible with IPv6
Theorem

The number of times a bit cannot be flipped, i.e., the number of times \( \neg(\text{used}_i(\ldots0) \land \text{used}_i(\ldots1)) = 0 \) (white nodes in middle figure of Address Tree 2) is the number of distinct addresses in the trace \(-1\) (in case there is at least one IP address already in the trace).

Definition

\[
q = \frac{\text{number of times } \neg(\text{used}_i(\ldots0) \land \text{used}_i(\ldots1))}{2^n} = \frac{\text{number of distinct addresses} - 1}{\text{size of address space}}
\]
Theorem

The number of times a bit cannot be flipped, i.e., the number of times \( \neg(\text{used}_i(...0) \land \text{used}_i(...1)) = 0 \) (white nodes in middle figure of Address Tree 2) is the number of distinct addresses in the trace \(-1\) (in case there is at least one IP address already in the trace).

Definition

\[
q = \frac{\text{number of times} \neg(\text{used}_i(...0) \land \text{used}_i(...1))}{2^n} = \frac{\text{number of distinct addresses} - 1}{\text{size of address space}}
\]