NetHide: Secure and Practical Network Topology Obfuscation

Roland Meier(1), Petar Tsankov(1), Vincent Lenders(2), Laurent Vanbever(1), Martin Vechev(1)

nethide.ethz.ch

USENIX Security 2018
Link-flooding attacks (LFAs) target the infrastructure

Low-rate, legitimate flows spread over many endpoints

Botnet

Public servers
Link-flooding attacks (LFAs) require knowing the topology.
$ traceroute X
1
$ traceroute X
1

UDP
dst = X
TTL = 1

Botnet
Public servers
$ traceroute X
1  A  1.755 ms

UDP
dst = X
TTL = 1

ICMP
TTL Exceeded
src = A

do
$ traceroute X
1  -A-  1.755 ms
2

UDP
dst = X
TTL = 2

Botnet

Public servers
$ traceroute X
  1  -A-  1.755 ms
  2

UDP
dst = X
TTL = 2

UDP
dst = X
TTL = 1

Botnet

Public servers
$ traceroute X
1  -A-  1.755 ms
2  -B-  1.062 ms

UDP
dst = X
TTL = 2

UDP
dst = X
TTL = 1

ICMP
TTL Exceeded
src = B

Botnet

Public servers
$ traceroute X
1  -A-  1.755 ms
2  -B-  1.062 ms
3  -C-  0.880 ms

UDP
dst = X
TTL = 3

ICMP
TTL Exceeded
src = C

Public servers
Botnet
$ traceroute X
1    -A-  1.755 ms
2    -B-  1.062 ms
3    -C-  0.880 ms
4    -D-  0.929 ms

UDP
dst = X
TTL = 4

ICMP
TTL Exceeded
src = D

device

Botnet

Public servers
$ traceroute X
1 — A — 1.755 ms
2 — B — 1.062 ms
3 — C — 0.880 ms
4 — D — 0.929 ms
5 — E — 0.827 ms

UDP
dst = X
TTL = 5

ICMP
TTL Exceeded
src = E
$ traceroute X
1 -A- 1.755 ms
2 -B- 1.062 ms
3 -C- 0.880 ms
4 -D- 0.929 ms
5 -E- 0.827 ms
6 -X- 0.819 ms

UDP
dst = X
TTL = 6

ICMP
TTL Exceeded
src = X
So the solution is to hide the topology?

yes, but...
traceroute is an essential debugging tool
So the solution is to hide the topology?
So the solution is to hide the topology? Which parts? How?
NetHide: Secure and Practical
Network Topology Obfuscation

NetHide deploys the virtual topology using programmable networks

NetHide computes a secure virtual topology that is similar to the physical topology

NetHide works for realistic topologies and maintains the utility of debugging tools
NetHide: Secure and Practical Network Topology Obfuscation

NetHide computes a secure virtual topology that is similar to the physical topology.

NetHide deploys the virtual topology using programmable networks.

NetHide works for realistic topologies and maintains the utility of debugging tools.
Topology obfuscation
as an optimization problem

Given the **physical topology** $P$, compute a **virtual topology** $V$, such that

- $V$ is robust against link-flooding attacks
- $V$ has maximal practicality
NetHide: Secure and Practical
Network Topology Obfuscation

NetHide computes a secure virtual topology that is similar to the physical topology.

NetHide deploys the virtual topology using programmable networks.

NetHide works for realistic topologies and maintains the utility of debugging tools.
Utility-preserving topology deployment

Deploy the virtual topology V, such that

- debugging tools still work
- network performance is not impacted
- it scales to large networks
Utility-preserving topology deployment

Deploy the virtual topology $V$, such that

- debugging tools still work
- network performance is not impacted
- it scales to large networks
Maintaining the utility of debugging tools requires sending packets through the actual network

- Answer from a central controller
- Answer at the edge
- Answer in a virtual clone of the network
- Answer from the correct device that appears on the path
Utility-preserving topology deployment

Deploy the *virtual topology V*, such that

- debugging tools still work
- network performance is not impacted
- it scales to large networks
Programmable network devices allow modifying tracing packets at line rate

- Read & modify packet headers
e.g. the TTL value

- Basic operations
e.g. hash functions and checksums

- Add or remove custom headers
to store information
Programmable network devices allow modifying tracing packets at line rate.

- Packets with a small TTL value expire in the network.
- Packets with different path lengths in P and V need to increase or decrease TTL.
Programmable network devices are configured through match+action tables.

If I receive a packet to X with TTL = i, I should send it to Y with TTL = j.
Programmable network devices are configured through match+action tables.

<table>
<thead>
<tr>
<th></th>
<th>original</th>
<th></th>
<th>new</th>
</tr>
</thead>
<tbody>
<tr>
<td>dst</td>
<td>TTL</td>
<td>dst</td>
<td>TTL</td>
</tr>
<tr>
<td>X</td>
<td>1</td>
<td>Y</td>
<td>2</td>
</tr>
</tbody>
</table>

physical path

virtual path
Deploy the virtual topology $V$, such that

- debugging tools still work
- network performance is not impacted
- it scales to large networks
Encoding state in packets instead of storing it in devices
NetHide: Secure and Practical Network Topology Obfuscation

NetHide computes a secure virtual topology that is similar to the physical topology.

NetHide deploys the virtual topology using programmable networks.

NetHide works for realistic topologies and maintains the utility of debugging tools.
NetHide: Secure and Practical
Network Topology Obfuscation

NetHide computes a secure virtual topology that is similar to the physical topology.

NetHide deploys the virtual topology using programmable networks.

NetHide works for realistic topologies and maintains the utility of debugging tools.