Device Tracking via Linux's New TCP Source Port Selection Algorithm

Paper by Moshe Kol, Amit Klein and Yossi Gilad[1]

Seminar

Presenter:
Jürgen Mattheis
(juergmatth@gmail.com)

Supervising Professor:
Prof. Dr. Christian Schindelhauer

Supervising Assistants:
Sneha Mohanty
Saptadi Nugroho
Joan Bordoy
Wenxin Xiong

July 30, 2025

University of Freiburg, Technical Faculty, Chair for Computer Networks and Telematics

Introduction

Motivation

Meltdown and Spectre

MELTDOWN [3] SPECTRE [4]

- Exploited performance features of modern CPUs
- Side effects of out-of-order and speculative execution
- Allows to read innaccesible data from memory

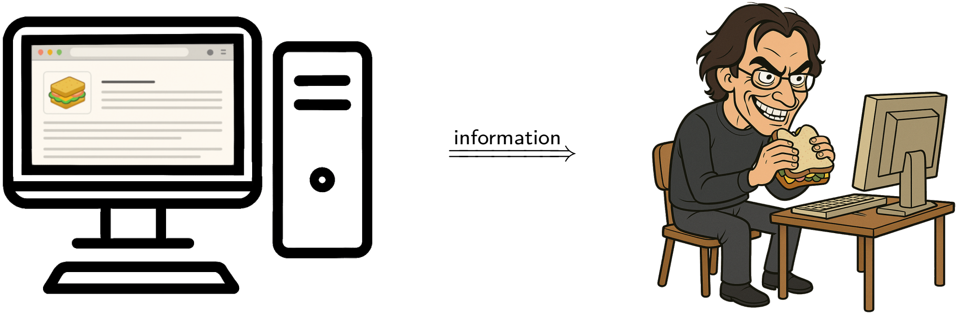
Device Tracking via Linux's New TCP Source Port Selection Algorithm (Extended Version) *

Moshe Kol, *Hebrew University of Jerusalem*
Amit Klein, *Hebrew University of Jerusalem*
Yossi Gilad, *Hebrew University of Jerusalem*

- Exploited Linux's TCP source port selection algorithm
- Side effects of hash collisions
- Allows to track devices through device ID tied to device-specific key

Browser-based device tracking

Definition and Problem



- Information collected through web browser
- Privacy risk

Browser-based device tracking

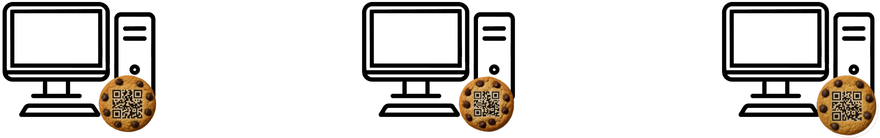
Goldene image challenge

- Golden image challenge: Distinguish identical devices with same hardware and software configuration ☑ $\Rightarrow$ device-specific key

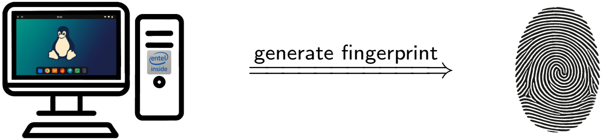
Browser-based device tracking

Types of techniques

- Tagging techniques: Insert an ID



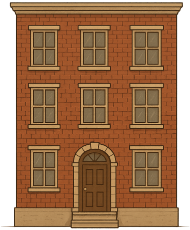
- Fingerprinting techniques: Measure characteristics



Source Port Selection

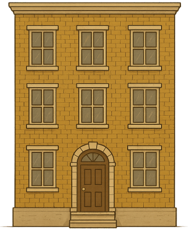
4-Tuple

- 4-tuple: $(IP_{Src}, Port_{Src}, IP_{Dest}, Port_{Dest})$



(123 Maple St., Apt. 5B) (456 Oak St., Apt. 9D)

(192.0.2.123, 55’555) (203.0.113.456, 49’999)



  - (192.0.2.123, 55’555, 203.0.113.10, 49’999)
  - (123 Maple St., Apt. 5B, 456 Oak St., Apt. 9D)

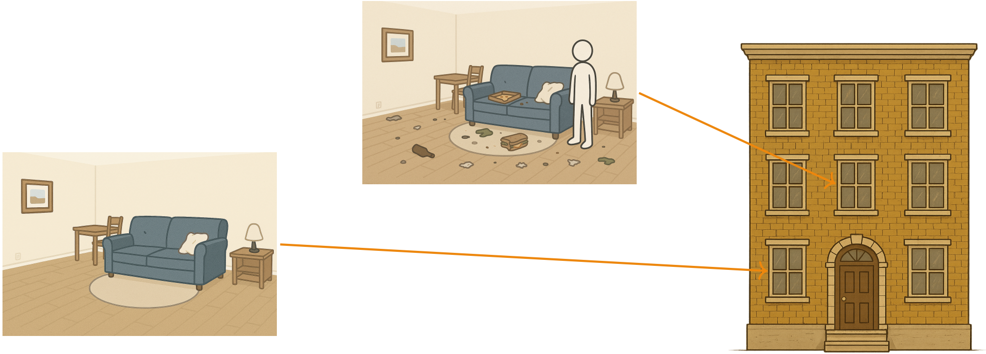
- Chooses TCP source port $Port_{Src}$ for 3-tuple: $(IP_{Src}, IP_{Dest}, Port_{Dest})$

Source Port Selection

Goals

- Functionality: Reusing soure ports can cause failures

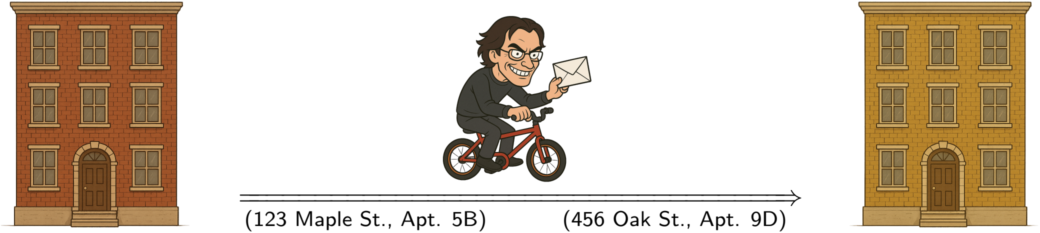
  $\Rightarrow$ Earlier connections might be active (`TIME_WAIT` state)



Source Port Selection

Goals

- Security:
  - Off-path attacks
  - Determine Device activity (number TCP connections per time)



Double-Hash Port Selection Algorithm (DHPS)

Algorithm Overview

- TCP source ports divided into ranges:

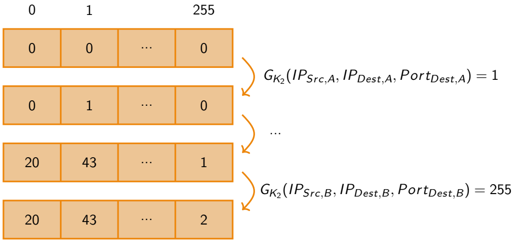
Port Range Type	Typical Range	Usage
Well-known ports	0–1023	Specific services, e.g. HTTP (80), HTTPS (443) etc.
Registered ports	1024–49151	User applications or services, not ephemeral
Ephemeral ports	49152–60999	Dynamically allocated by OS for short-lived client-side connections

Double-Hash Port Selection Algorithm (DHPS)

Algorithm Overview

- Complete 3-tuple to 4-tuple with cryptographic keyed-hash functions:
  - F_{K_1} : 3-Tuples $\xrightarrow{K_1} \{0, \ldots, 2^{32}\}$
  - G_{K_2} : 3-Tuples $\xrightarrow{K_2} \{0, \ldots, T\}$ where T is length of perturbation table

- Incrementation of port numbers separated into T different spaces
  - $\Rightarrow$ Port-reuse frequency lower[2]
  - $\Rightarrow$ Harder to analyse device activity



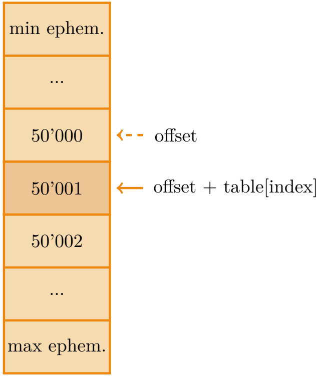
Double-Hash Port Selection Algorithm (DHPS)

Pseudocode

Algorithm 1 DHPS Source Port Selection

```
procedure SelectEphemeralPort
    num_ephemeral ← max_ephemeral − min_ephemeral + 1
    offset ← F_K1(IP_SRC, IP_DST, PORT_DST)
    index ← G_K2(IP_SRC, IP_DST, PORT_DST)
    count ← num_ephemeral
    repeat
        port ← min_ephemeral +
            ((offset + table[index]) mod num_ephemeral)
        table[index] ← table[index] + 1
        if CheckSuitablePort(port) then
            return port
        count ← count − 1
    until count = 0
    return Error
```

- Source port calculation:
  - $offset = 50'000$
  - $index = 255$



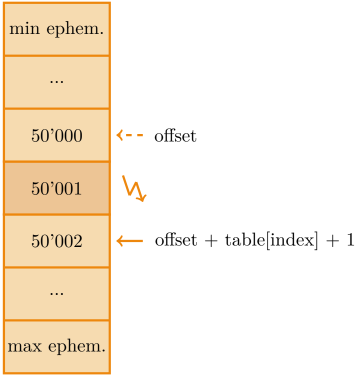
Double-Hash Port Selection Algorithm (DHPS)

Pseudocode

Algorithm 2 DHPS SOURCE PORT SELECTION

```
procedure SELECTEPHEMERALPORT
    num_ephemeral ← max_ephemeral − min_ephemeral + 1
    offset ← F_K1(IP_SRC,IP_DST,PORT_DST)
    index ← G_K2(IP_SRC,IP_DST,PORT_DST)
    count ← num_ephemeral
    repeat
        port ← min_ephemeral +
            ((offset + table[index]) mod num_ephemeral)
        table[index] ← table[index] + 1
        if CHECKSUITABLEPORT(port) then
            return port
        count←count − 1
    until count = 0
    return Error
```

- Source port calculation:
  - $offset = 50'000$
  - $index = 255$



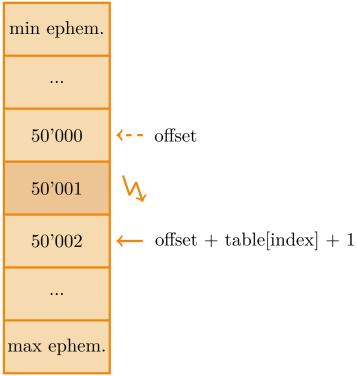
Double-Hash Port Selection Algorithm (DHPS)

Pseudocode

Algorithm 3 DHPS Source Port Selection

```
procedure SelectEphemeralPort
    num_ephemeral ← max_ephemeral − min_ephemeral + 1
    offset ← F_K1(IP_SRC,IP_DST,PORT_DST)
    index ← G_K2(IP_SRC,IP_DST,PORT_DST)
    count ← num_ephemeral
    repeat
        port ← min_ephemeral +
            ((offset + table[index]) mod num_ephemeral)
        table[index] ← table[index] + 1
        if CheckSuitablePort(port) then
            return port
        count←count − 1
    until count = 0
    return Error
```

- Source port calculation:
  - $offset = 50'000$
  - $index = 255$



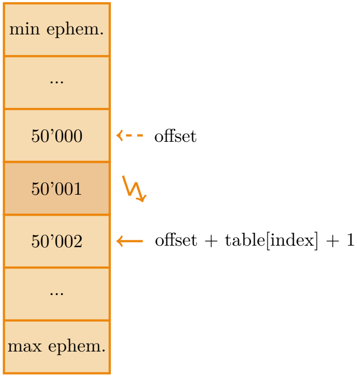
Double-Hash Port Selection Algorithm (DHPS)

Pseudocode

Algorithm 4 DHPS SOURCE PORT SELECTION

```
procedure SelectEphemeralPort
    num_ephemeral ← max_ephemeral − min_ephemeral + 1
    offset ← F_K1(IP_SRC,IP_DST,PORT_DST)
    index ← G_K2(IP_SRC,IP_DST,PORT_DST)
    count ← num_ephemeral
    repeat
        port ← min_ephemeral +
            ((offset + table[index]) mod num_ephemeral)
        table[index] ← table[index] + 1
        if CheckSuitablePort(port) then
            return port
        count←count − 1
    until count = 0
    return Error
```

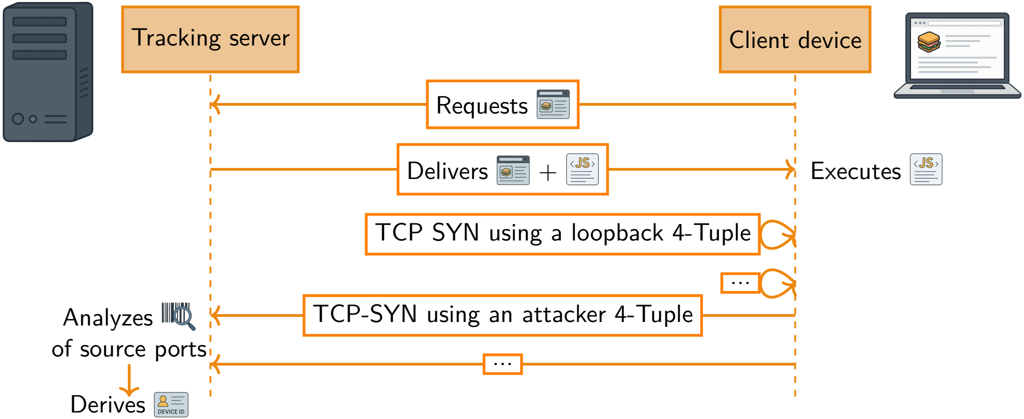
- Source port calculation:
  - $offset = 50'000$
  - $index = 255$



Attack

Attack Overview

Parties



Attack Overview

2 Types of 3-tuples

- Attacker 3-tuples:

$$DHPS(IP_{Src} = 192.0.2.123, IP_{Dst} = 203.0.113.10, Port_{Dst} = 456)$$

$=$

$$(192.0.2.123, 55'555, 203.0.113.10, 456)$$

$$(IP_{Src}, Port_{Src}, IP_{Dest}, Port_{Dest})$$

- Loopback 3-tuples:

$$DHPS(IP_{Src} = 127.0.0.1, IP_{Dst} = 127.0.0.3, Port_{Dst} = 11'111)$$

$=$

$$(127.0.0.1, 49'999, 127.0.0.3, 11'111)$$

$$(IP_{Src}, Port_{Src}, IP_{Dest}, Port_{Dest})$$

Device ID

Collisions

- Structure formed by collisions of loopback 3-tuples via G_{K_2}:

0	1	···	T-2	T-1
20	43	···	9+1+1	2

$G_{K_2}(IP_{Src}, IP_{Dst}, Port_{Dst}) = index_1$

$G_{\boxed{K_2}}(127.0.0.1,\ 127.0.0.3,\ x) = \text{T-2}$

$G_{\boxed{K_2}}(127.0.0.1,\ 127.0.0.3,\ y) = \text{T-2}$

$G_{K_2}(IP_{Src}, IP_{Dst}, Port_{Dst}) = index_2$

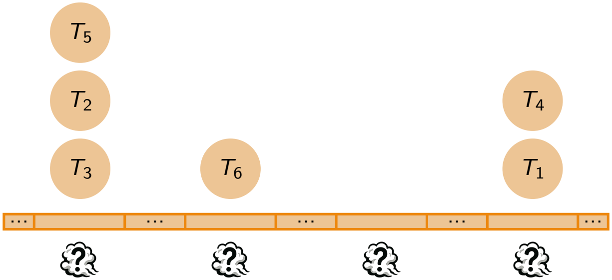
- Collisions depend on K_2
- K_2 randomly generated at
  - $\Rightarrow$ Persists across and
  - $\Rightarrow$ Also persists

Device ID

Independent pairs

- Consists of set of all such colliding independent pairs:

$\{(T_3, T_2),$
$(T_3, T_5)\}$ $\{(T_1, T_4)\}$



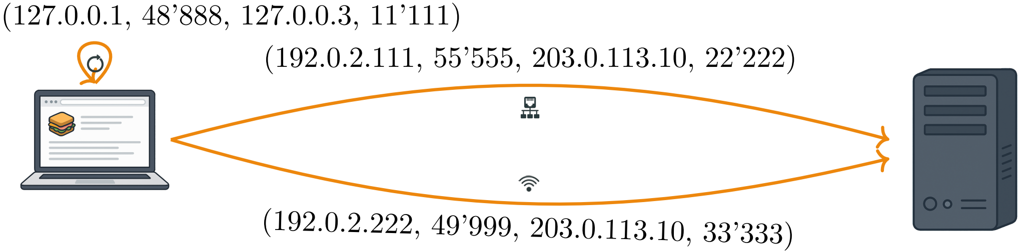
- : 6 = Device ID
- : 3, evidence how K_2 maps into perturbation table cells

- Why?
  - $\binom{3}{2}$ pairs of $\{T_3, T_2, T_5\}$
    - $3 - 1$ are independent
  - $\{(T_3, T_2), (T_3, T_5)\}$ imply (T_2, T_5)

Device ID

Why loopback 3-tuples?

- Attacker 3-tuples not consistent across
- Loopback 3-tuples don't have this problem

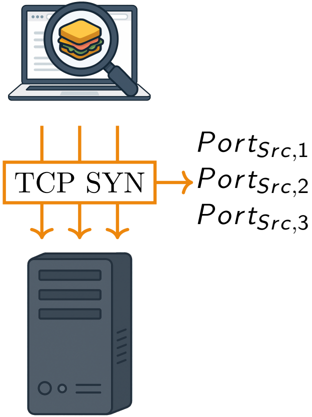


$\Rightarrow$ Loopback 3-tuples used for generating DEVICE ID

Sandwiching Technique

Cannot directly observe collisions

- no access to kernel internals



0	1	...	T-2	T-1
20	43	...	9+1+1	2

G_{K_2}(127.0.0.1, 127.0.0.3, x)=T-2

G_{K_2}(127.0.0.1, 127.0.0.3, y)=T-2

- side effect: predictable increment

$$\texttt{table[index]} \leftarrow \texttt{table[index]} + 1$$

- analyse source port increments via 

$\Rightarrow$ derive 

Sandwiching Technique

General principle

Step	Calculation
Wrapper	$i = G_{K_2}(\boxed{\mathrm{IP}_{\mathrm{Src},X}, \mathrm{IP}_{\mathrm{Dst},X}, \mathrm{Port}_{\mathrm{Dst},X}})$
Filling	$j = G_{K_2}(\mathrm{IP}_{\mathrm{Src},Y}, \mathrm{IP}_{\mathrm{Dst},Y}, \mathrm{Port}_{\mathrm{Dst},Y})$
Wrapper	$i = G_{K_2}(\boxed{\mathrm{IP}_{\mathrm{Src},X}, \mathrm{IP}_{\mathrm{Dst},X}, \mathrm{Port}_{\mathrm{Dst},X}})$

Sandwiching Technique

General principle

Step	Calculation
Wrapper	$i = G_{K_2}(\mathrm{IP}_{\mathrm{Src},X}, \mathrm{IP}_{\mathrm{Dst},X}, \mathrm{Port}_{\mathrm{Dst},X})$ $\mathrm{offset}_X = F_{K_1}(\mathrm{IP}_{\mathrm{Src},X}, \mathrm{IP}_{\mathrm{Dst},X}, \mathrm{Port}_{\mathrm{Dst},X})$
Filling	$j = G_{K_2}(\mathrm{IP}_{\mathrm{Src},Y}, \mathrm{IP}_{\mathrm{Dst},Y}, \mathrm{Port}_{\mathrm{Dst},Y})$ $\mathrm{offset}_Y = F_{K_1}(\mathrm{IP}_{\mathrm{Src},Y}, \mathrm{IP}_{\mathrm{Dst},Y}, \mathrm{Port}_{\mathrm{Dst},Y})$
Wrapper	$i = G_{K_2}(\mathrm{IP}_{\mathrm{Src},X}, \mathrm{IP}_{\mathrm{Dst},X}, \mathrm{Port}_{\mathrm{Dst},X})$ $\mathrm{offset}_X = F_{K_1}(\mathrm{IP}_{\mathrm{Src},X}, \mathrm{IP}_{\mathrm{Dst},X}, \mathrm{Port}_{\mathrm{Dst},X})$

Sandwiching Technique

General principle

Step	Calculation
Wrapper	$i = G_{K_2}(\mathrm{IP}_{\mathrm{Src},X}, \mathrm{IP}_{\mathrm{Dst},X}, \mathrm{Port}_{\mathrm{Dst},X})$ $\mathrm{offset}_X = F_{K_1}(\mathrm{IP}_{\mathrm{Src},X}, \mathrm{IP}_{\mathrm{Dst},X}, \mathrm{Port}_{\mathrm{Dst},X})$ $\mathrm{Port}_{\mathrm{Src},1} = \mathrm{min\_ephemeral} + (\mathrm{offset}_X + \mathrm{table}[i]) \bmod \mathrm{num\_ephemeral}$
Filling	$j = G_{K_2}(\mathrm{IP}_{\mathrm{Src},Y}, \mathrm{IP}_{\mathrm{Dst},Y}, \mathrm{Port}_{\mathrm{Dst},Y})$ $\mathrm{offset}_Y = F_{K_1}(\mathrm{IP}_{\mathrm{Src},Y}, \mathrm{IP}_{\mathrm{Dst},Y}, \mathrm{Port}_{\mathrm{Dst},Y})$ $\mathrm{Port}_{\mathrm{Src},2} = \mathrm{min\_ephemeral} + (\mathrm{offset}_Y + \mathrm{table}[j]) \bmod \mathrm{num\_ephemeral}$
Wrapper	$i = G_{K_2}(\mathrm{IP}_{\mathrm{Src},X}, \mathrm{IP}_{\mathrm{Dst},X}, \mathrm{Port}_{\mathrm{Dst},X})$ $\mathrm{offset}_X = F_{K_1}(\mathrm{IP}_{\mathrm{Src},X}, \mathrm{IP}_{\mathrm{Dst},X}, \mathrm{Port}_{\mathrm{Dst},X})$ $\mathrm{Port}_{\mathrm{Src},3} = \mathrm{min\_ephemeral} + (\mathrm{offset}_X + \mathrm{table}[i]) \bmod \mathrm{num\_ephemeral}$

Sandwiching Technique

General principle

Step	Calculation
Wrapper	$i = G_{K_2}(\mathrm{IP}_{\mathrm{Src},X}, \mathrm{IP}_{\mathrm{Dst},X}, \mathrm{Port}_{\mathrm{Dst},X})$ $\mathrm{offset}_X = F_{K_1}(\mathrm{IP}_{\mathrm{Src},X}, \mathrm{IP}_{\mathrm{Dst},X}, \mathrm{Port}_{\mathrm{Dst},X})$ $\mathrm{Port}_{\mathrm{Src},1} = \mathrm{min\_ephemeral} + (\mathrm{offset}_X + \mathrm{table}[i]) \bmod \mathrm{num\_ephemeral}$ $\mathrm{table}[i] = \mathrm{table}[i] + 1$
Filling	$j = G_{K_2}(\mathrm{IP}_{\mathrm{Src},Y}, \mathrm{IP}_{\mathrm{Dst},Y}, \mathrm{Port}_{\mathrm{Dst},Y})$ $\mathrm{offset}_Y = F_{K_1}(\mathrm{IP}_{\mathrm{Src},Y}, \mathrm{IP}_{\mathrm{Dst},Y}, \mathrm{Port}_{\mathrm{Dst},Y})$ $\mathrm{Port}_{\mathrm{Src},2} = \mathrm{min\_ephemeral} + (\mathrm{offset}_Y + \mathrm{table}[j]) \bmod \mathrm{num\_ephemeral}$ $\mathrm{table}[j] = \mathrm{table}[j] + 1$
Wrapper	$i = G_{K_2}(\mathrm{IP}_{\mathrm{Src},X}, \mathrm{IP}_{\mathrm{Dst},X}, \mathrm{Port}_{\mathrm{Dst},X})$ $\mathrm{offset}_X = F_{K_1}(\mathrm{IP}_{\mathrm{Src},X}, \mathrm{IP}_{\mathrm{Dst},X}, \mathrm{Port}_{\mathrm{Dst},X})$ $\mathrm{Port}_{\mathrm{Src},3} = \mathrm{min\_ephemeral} + (\mathrm{offset}_X + \mathrm{table}[i]) \bmod \mathrm{num\_ephemeral}$ $\mathrm{table}[i] = \mathrm{table}[i] + 1$

Sandwiching Technique

General principle

$$table[i] = \begin{cases} rand(i) + 1, & \text{if } i \neq j, \text{ meaning table cell } i \text{ was not used filling step} \\ rand(i) + 2, & \text{if } i = j, \text{ meaning table cell } i \text{ was used in filling step} \end{cases}$$

Step	Calculation
Wrapper	$i = G_{K_2}(\mathrm{IP}_{\mathrm{Src},X}, \mathrm{IP}_{\mathrm{Dst},X}, \mathrm{Port}_{\mathrm{Dst},X})$ $\mathrm{offset}_X = F_{K_1}(\mathrm{IP}_{\mathrm{Src},X}, \mathrm{IP}_{\mathrm{Dst},X}, \mathrm{Port}_{\mathrm{Dst},X})$ $\mathrm{Port}_{\mathrm{Src},1} = \mathrm{min\_ephemeral} + (\mathrm{offset}_X + \mathrm{table}[i]) \bmod \mathrm{num\_ephemeral}$ $\mathrm{table}[i] = \mathrm{table}[i] + 1$
Filling	$j = G_{K_2}(\mathrm{IP}_{\mathrm{Src},Y}, \mathrm{IP}_{\mathrm{Dst},Y}, \mathrm{Port}_{\mathrm{Dst},Y})$ $\mathrm{offset}_Y = F_{K_1}(\mathrm{IP}_{\mathrm{Src},Y}, \mathrm{IP}_{\mathrm{Dst},Y}, \mathrm{Port}_{\mathrm{Dst},Y})$ $\mathrm{Port}_{\mathrm{Src},2} = \mathrm{min\_ephemeral} + (\mathrm{offset}_Y + \mathrm{table}[j]) \bmod \mathrm{num\_ephemeral}$ $\mathrm{table}[j] = \mathrm{table}[j] + 1$
Wrapper	$i = G_{K_2}(\mathrm{IP}_{\mathrm{Src},X}, \mathrm{IP}_{\mathrm{Dst},X}, \mathrm{Port}_{\mathrm{Dst},X})$ $\mathrm{offset}_X = F_{K_1}(\mathrm{IP}_{\mathrm{Src},X}, \mathrm{IP}_{\mathrm{Dst},X}, \mathrm{Port}_{\mathrm{Dst},X})$ $\mathrm{Port}_{\mathrm{Src},3} = \mathrm{min\_ephemeral} + (\mathrm{offset}_X + \boxed{\mathrm{table}[i]}) \bmod \mathrm{num\_ephemeral}$ $\mathrm{table}[i] = \mathrm{table}[i] + 1$

Sandwiching Technique

General principle

Step	Calculation
Wrapper	$i = G_{K_2}(\mathrm{IP}_{\mathrm{Src},X}, \mathrm{IP}_{\mathrm{Dst},X}, \mathrm{Port}_{\mathrm{Dst},X})$ $\mathrm{offset}_X = F_{K_1}(\mathrm{IP}_{\mathrm{Src},X}, \mathrm{IP}_{\mathrm{Dst},X}, \mathrm{Port}_{\mathrm{Dst},X})$ $\boxed{\mathrm{Port}_{\mathrm{Src},1}} = \mathrm{min\_ephemeral} + (\mathrm{offset}_X + \mathrm{table}[i]) \bmod \mathrm{num\_ephemeral}$ $\mathrm{table}[i] = \mathrm{table}[i] + 1$
Filling	$i = G_{K_2}(\mathrm{IP}_{\mathrm{Src},Y}, \mathrm{IP}_{\mathrm{Dst},Y}, \mathrm{Port}_{\mathrm{Dst},Y})$ $\mathrm{offset}_Y = F_{K_1}(\mathrm{IP}_{\mathrm{Src},Y}, \mathrm{IP}_{\mathrm{Dst},Y}, \mathrm{Port}_{\mathrm{Dst},Y})$ $\mathrm{Port}_{\mathrm{Src},2} = \mathrm{min\_ephemeral} + (\mathrm{offset}_Y + \mathrm{table}[i]) \bmod \mathrm{num\_ephemeral}$ $\mathrm{table}[i] = \mathrm{table}[i] + 1$
Wrapper	$i = G_{K_2}(\mathrm{IP}_{\mathrm{Src},X}, \mathrm{IP}_{\mathrm{Dst},X}, \mathrm{Port}_{\mathrm{Dst},X})$ $\mathrm{offset}_X = F_{K_1}(\mathrm{IP}_{\mathrm{Src},X}, \mathrm{IP}_{\mathrm{Dst},X}, \mathrm{Port}_{\mathrm{Dst},X})$ $\boxed{\mathrm{Port}_{\mathrm{Src},3}} = \mathrm{min\_ephemeral} + (\mathrm{offset}_X + \mathrm{table}[i]) \bmod \mathrm{num\_ephemeral}$ $\mathrm{table}[i] = \mathrm{table}[i] + 1$

Collision between 3-tuples X and Y

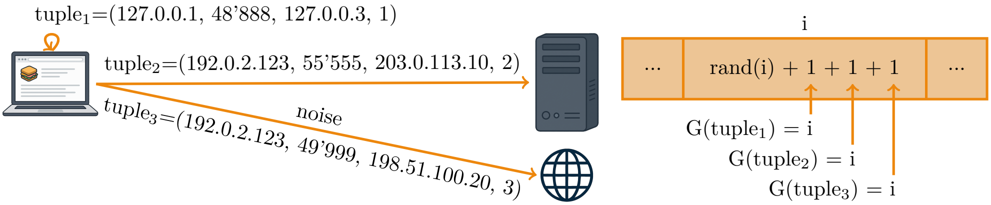
$\Leftrightarrow$ G_{K_2} maps 3-tuples X and Y to same perturbation table cell

$\Leftrightarrow$ $\mathrm{Port}_{\mathrm{Src},3}$ is greater than $\mathrm{Port}_{\mathrm{Src},1} + 1$

Sandwiching Technique

To consider in practise

- Background noise



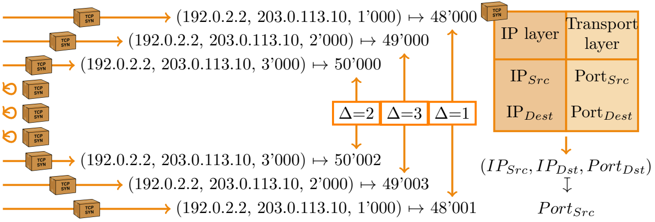
- Multiple 3-tuples used in steps
- Limitations of client device

Sandwiching Technique

To consider in practise

- Background noise
- Multiple 3-tuples used in steps

Wrapper	(192.0.2.2, 203.0.113.10, 1'000)
	(192.0.2.2, 203.0.113.10, 2'000)
	(192.0.2.2, 203.0.113.10, 3'000)
Filling	(127.0.0.1, 127.0.0.3, 4'000)
	(127.0.0.1, 127.0.0.3, 5'000)
	(127.0.0.1, 127.0.0.3, 6'000)
Wrapper	(192.0.2.2, 203.0.113.10, 3'000)
	(192.0.2.2, 203.0.113.10, 2'000)
	(192.0.2.2, 203.0.113.10, 1'000)

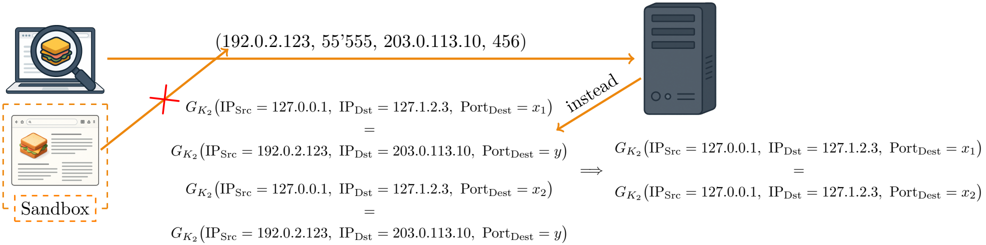


- Limitations of client device

Sandwiching Technique

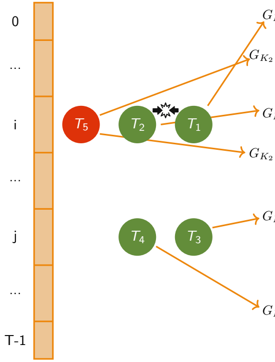
To consider in practise

- Background noise
- Multiple 3-tuples used in steps
- Limitations of client device



Phase 1

Reason for Phase 1



$$G_{K_2}(\mathrm{IP}_{\mathrm{Src}} = 127.0.0.1,\ \mathrm{IP}_{\mathrm{Dst}} = 127.1.2.3,\ \mathrm{Port}_{\mathrm{Dest}} = x_1) = G_{K_2}(\mathrm{IP}_{\mathrm{Src}} = 192.0.2.123,\ \mathrm{IP}_{\mathrm{Dst}} = 203.0.113.10,\ \mathrm{Port}_{\mathrm{Dest}} = y)$$

$$G_{K_2}(\mathrm{IP}_{\mathrm{Src}} = 127.0.0.1,\ \mathrm{IP}_{\mathrm{Dst}} = 127.1.2.3,\ \mathrm{Port}_{\mathrm{Dest}} = x_2) = G_{K_2}(\mathrm{IP}_{\mathrm{Src}} = 192.0.2.123,\ \mathrm{IP}_{\mathrm{Dst}} = 203.0.113.10,\ \mathrm{Port}_{\mathrm{Dest}} = y)$$

$$G_{K_2}(\mathrm{IP}_{\mathrm{Src}} = 127.0.0.1,\ \mathrm{IP}_{\mathrm{Dst}} = 127.1.2.3,\ \mathrm{Port}_{\mathrm{Dest}} = x_3) = 🚫$$

$$G_{K_2}(\mathrm{IP}_{\mathrm{Src}} = 127.0.0.1,\ \mathrm{IP}_{\mathrm{Dst}} = 127.1.2.3,\ \mathrm{Port}_{\mathrm{Dest}} = x_4) = 🚫$$

$$\longrightarrow$$

$$G_{K_2}(\mathrm{IP}_{\mathrm{Src}} = 127.0.0.1,\ \mathrm{IP}_{\mathrm{Dst}} = 127.1.2.3,\ \mathrm{Port}_{\mathrm{Dest}} = x_1) = G_{K_2}(\mathrm{IP}_{\mathrm{Src}} = 127.0.0.1,\ \mathrm{IP}_{\mathrm{Dst}} = 127.1.2.3,\ \mathrm{Port}_{\mathrm{Dest}} = x_2)$$

$$G_{K_2}(\mathrm{IP}_{\mathrm{Src}} = 127.0.0.1,\ \mathrm{IP}_{\mathrm{Dst}} = 127.1.2.3,\ \mathrm{Port}_{\mathrm{Dest}} = x_3) \stackrel{?}{=} G_{K_2}(\mathrm{IP}_{\mathrm{Src}} = 127.0.0.1,\ \mathrm{IP}_{\mathrm{Dst}} = 127.1.2.3,\ \mathrm{Port}_{\mathrm{Dest}} = x_4)$$

- Every cell should be testable for collisions

$\Rightarrow$ Set of unique attacker 3-tuples that collectively cover all cells

Phase 1

Pseudocode

Algorithm 5 FINDING ATTACKER 3-TUPLE PER CELL (PHASE 1)

```
procedure SendBurst(X)
    for all x ∈ X do
        AttemptConnectTCP(x)
procedure GetSourcePorts(U)
    SendBurst(U)
    R ← ReceiveAttackerTupleToPortMap()
    return R
procedure Phase1
    S' ← ∅
    while |S'| < T do
        S ← GetNewExternalDestinations()
        C ← GetSourcePorts(S) // 1st burst
        SendBurst(S') // 2nd burst
        C' ← GetSourcePorts(S) // 3rd burst
        S' ← S' ∪ {x | C'(x) − C(x) = 1}
    return S'
```

- S: New candidate attacker 3-tuples
- S': Unique attacker 3-tuples found so far
- P, P': Functions of $(IP_{Src}, IP_{Dest}, Port_{Dest}) \mapsto Port_{Src}$ for new candiate attacker 3-tuples
- x: Candidate attacker 3-tuple

Phase 1

Overview

Candidate Attacker 3-Tuples

(192.0.2.2, 203.0.113.10, 5’000)	TCP SYN	50’200
(192.0.2.2, 203.0.113.10, 6’000)	TCP SYN	51’100
(192.0.2.2, 203.0.113.10, 7’000)	TCP SYN	52’020
(192.0.2.2, 203.0.113.10, 8’000)	TCP SYN	53’021

Unique Attacker 3-Tuples found so far

(192.0.2.2, 203.0.113.10, 1’000)	TCP SYN	-
(192.0.2.2, 203.0.113.10, 2’000)	TCP SYN	-
(192.0.2.2, 203.0.113.10, 3’000)	TCP SYN	-
(192.0.2.2, 203.0.113.10, 4’000)	TCP SYN	-

Candidate Attacker 3-Tuples

(192.0.2.2, 203.0.113.10, 5’000)	TCP SYN	50’201
(192.0.2.2, 203.0.113.10, 6’000)	TCP SYN	51’103
(192.0.2.2, 203.0.113.10, 7’000)	TCP SYN	52’022
(192.0.2.2, 203.0.113.10, 8’000)	TCP SYN	53’023

192.0.2.2

203.0.113.10

Phase 1

Pseudocode

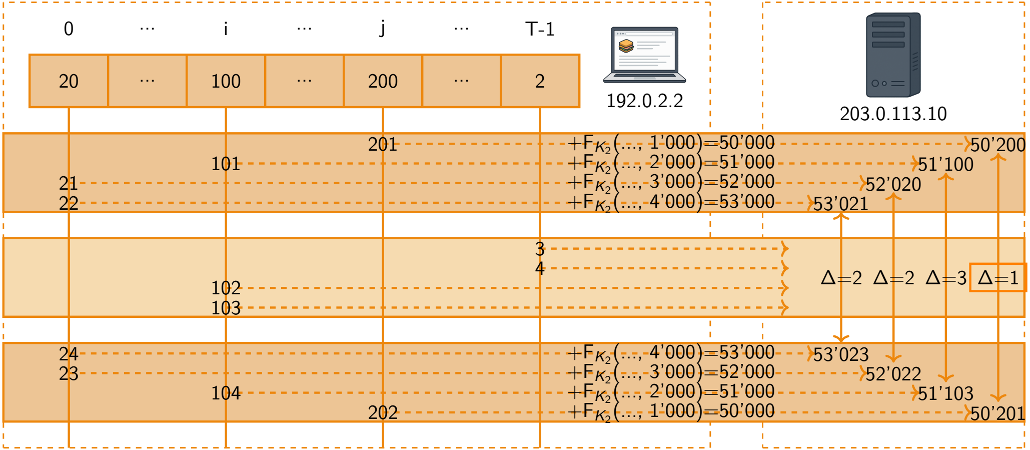
Algorithm 6 FINDING ATTACKER 3-TUPLE PER CELL (PHASE 1)

```
procedure SENDBURST(X)
    for all x ∈ X do
        ATTEMPTCONNECTTCP(x)
procedure GETSOURCEPORTS(U)
    SENDBURST(U)
    R ← RECEIVEATTACKERTUPLETOPORTMAP()
    return R
procedure PHASE1
    S' ← ∅
    while |S'| < T do
        S ← GETNEWEXTERNALDESTINATIONS()
        C ← GETSOURCEPORTS(S) // 1st burst
        SENDBURST(S') // 2nd burst
        C' ← GETSOURCEPORTS(S) // 3rd burst
        S' ← S' ∪ {x | C'(x) − C(x) = 1}
    return S'
```

- S: New candidate attacker 3-tuples
- S': Unique attacker 3-tuples found so far
- P, P': Functions of $(IP_{Src}, IP_{Dest}, Port_{Dest}) \mapsto Port_{Src}$ for new candiate attacker 3-tuples
- x: Candidate attacker 3-tuple

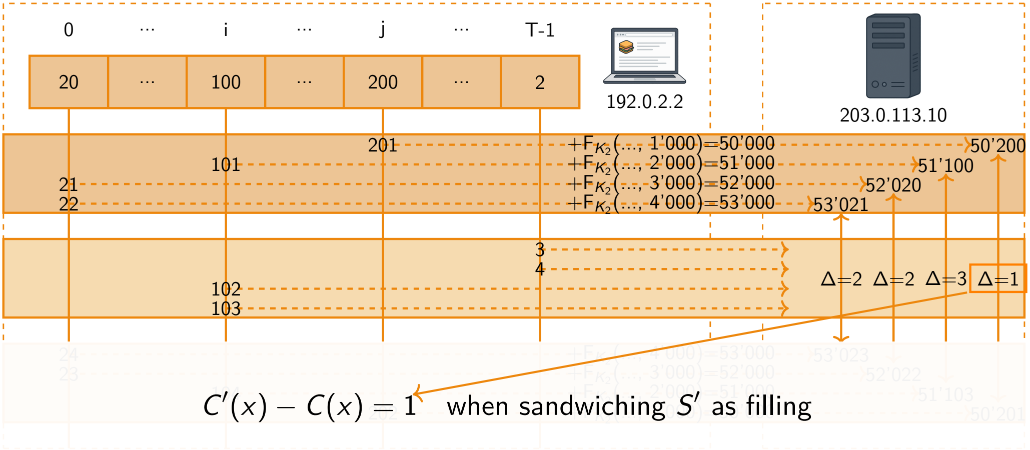
Phase 1

Condition for unique attacker 3-tuple



Phase 1

Condition for unique attacker 3-tuple



$C'(x) - C(x) = 1$ when sandwiching S' as filling

Phase 1

Pseudocode

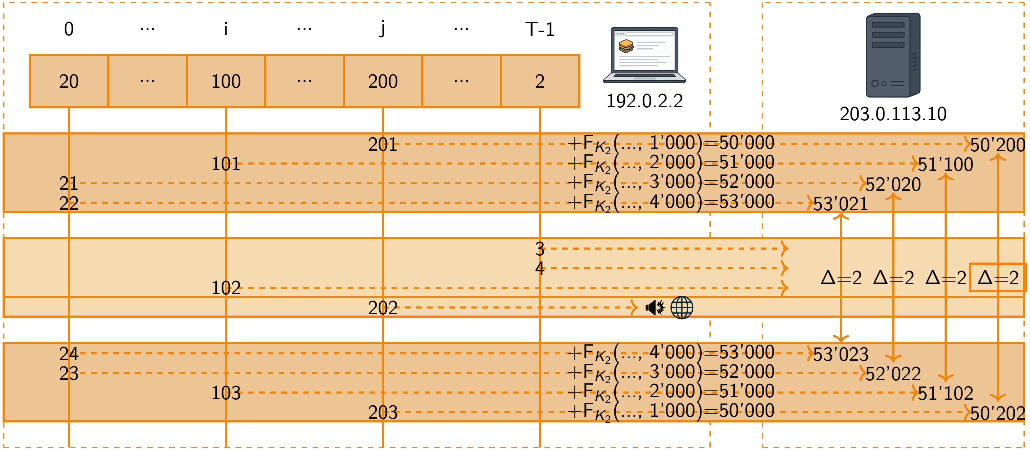
Algorithm 7 Finding Attacker 3-Tuple per Cell (Phase 1)

```
procedure SendBurst(X)
    for all x ∈ X do
        AttemptConnectTCP(x)
procedure GetSourcePorts(U)
    SendBurst(U)
    R ← ReceiveAttackerTupleToPortMap()
    return R
procedure Phase1
    S' ← ∅
    while |S'| < T do
        S ← GetNewExternalDestinations()
        C ← GetSourcePorts(S) // 1st burst
        SendBurst(S') // 2nd burst
        C' ← GetSourcePorts(S) // 3rd burst
        S' ← S' ∪ {x | C'(x) − C(x) = 1}
    return S'
```

- S: New candidate attacker 3-tuples
- S': Unique attacker 3-tuples found so far
- P, P': Functions of $(IP_{Src}, IP_{Dest}, Port_{Dest}) \mapsto Port_{Src}$ for new candiate attacker 3-tuples
- x: Candidate attacker 3-tuple

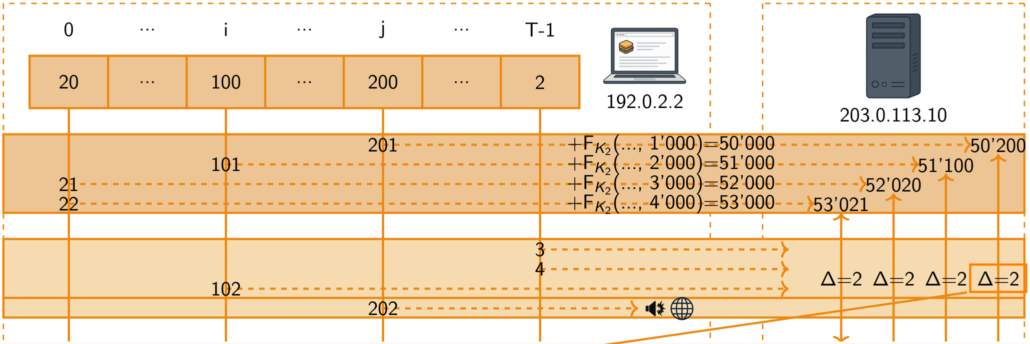
Phase 1

Influence of noise



Phase 1

Influence of noise



- Noise can hide a unique attacker 3-tuple, but it can't fabricate one

Phase 2

Pseudocode

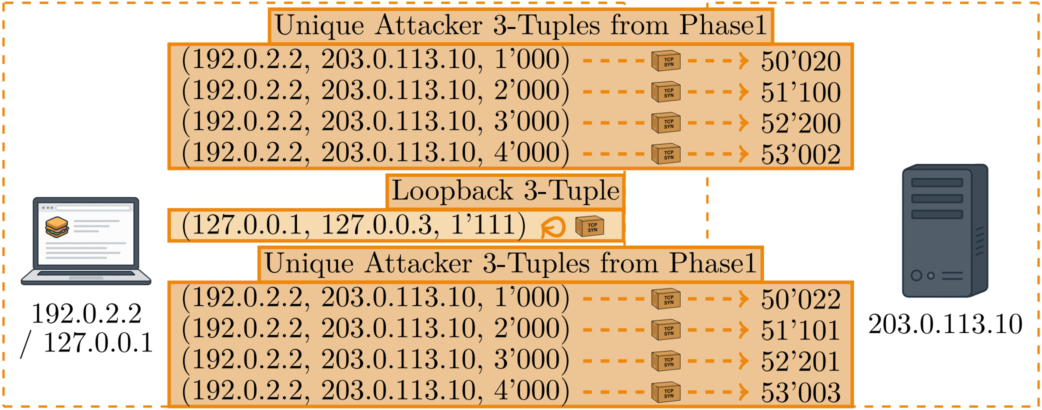
Algorithm 8 Finding a Device ID (Phase 2)

procedure Phase2
  $\texttt{C} \leftarrow \emptyset; \texttt{n} \leftarrow 0; \texttt{i} \leftarrow 0$
  repeat
    $\boxed{\texttt{i} \leftarrow \texttt{i} + 1}$
    $\texttt{C} \leftarrow$ GetSourcePorts(S') // 1st burst
    AttemptConnectTCP($\{L_i\}$) // 2nd burst
    $\texttt{C'} \leftarrow$ GetSourcePorts(S') // 3rd burst
    $\texttt{w} \leftarrow \iota w(\texttt{C'}(w) - \texttt{C}(w) > \varepsilon)$
    if Defined(B[w]) **then** // Collision was found
      $\texttt{p} \leftarrow p \cup \{(L_i, \texttt{B}[w])\}$
      $\texttt{n} \leftarrow n + 1$
    else
      $\texttt{B}[w] \leftarrow L_i$
  until $\texttt{n} \geq \texttt{n}^*[i]$ // This is equiv. to $P_T(i, n) \leq p^*$
  return (p,i)

- i: Number of iterations
- n: Number of independent pairs / collisions of loopback 3-tuples
- L_i: Single loopback 3-tuple
- S': Unique attacker 3-tuples from Phase 1
- w: The one unique attacker 3-tuple

Phase 2

Overview



Phase 2

Pseudocode

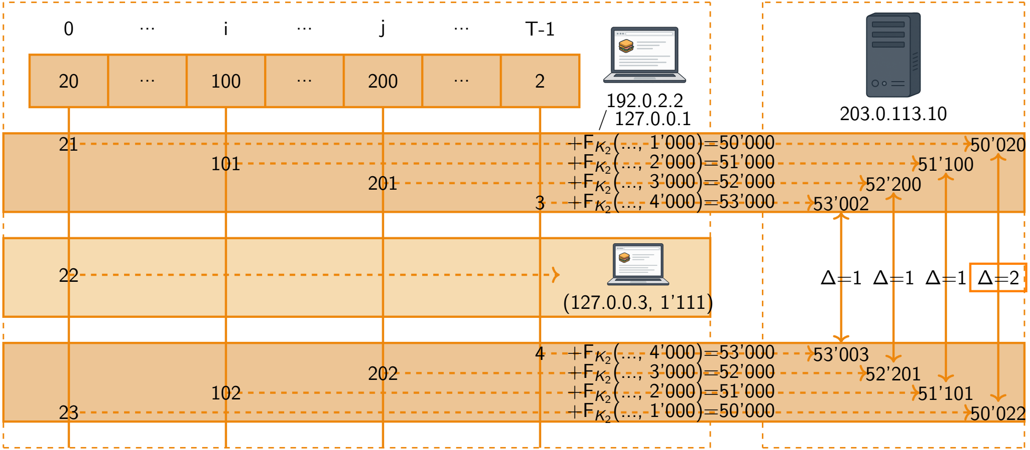
Algorithm 9 FINDING A DEVICE ID (PHASE 2)

```
procedure PHASE2
    C ← ∅; n ← 0; i ← 0
    repeat
        i ← i + 1
        C ← GETSOURCEPORTS(S')          // 1st burst
        ATTEMPTCONNECTTCP({L_i})         // 2nd burst
        C' ← GETSOURCEPORTS(S')         // 3rd burst
        w ← ιw(C'(w) − C(w) > ε)
        if DEFINED(B[w]) then            // Collision was found
            p ← p ∪ {(L_i, B[w])}
            n ← n + 1
        else
            B[w] ← L_i
    until n ≥ n*[i]                      // This is equiv. to P_T(i, n) ≤ p*
    return (p,i)
```

- i: Number of iterations
- n: Number of independent pairs / collisions of loopback 3-tuples
- L_i: Single loopback 3-tuple
- S': Unique attacker 3-tuples from Phase 1
- w: The one unique attacker 3-tuple
- ιw: Definite Description Operator, "the unique w" attacker 3-tuple

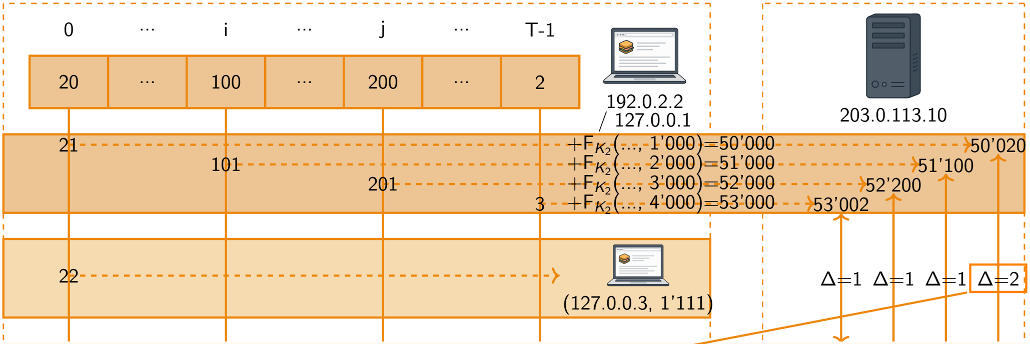
Phase 2

Condition for collision between attacker 3-tuple and loopback 3-tuple



Phase 2

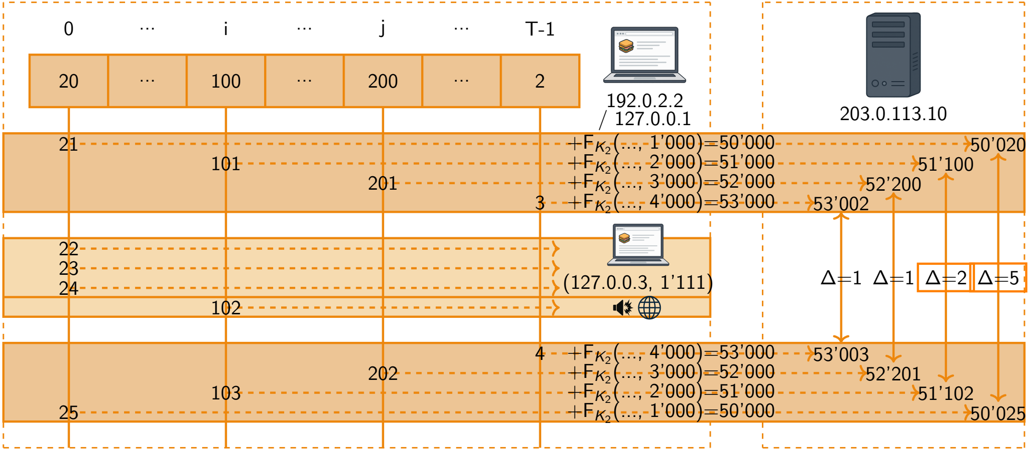
Condition for collision between attacker 3-tuple and loopback 3-tuple



$C'(w) - C(w) > 1$ when sandwiching L_i as filling

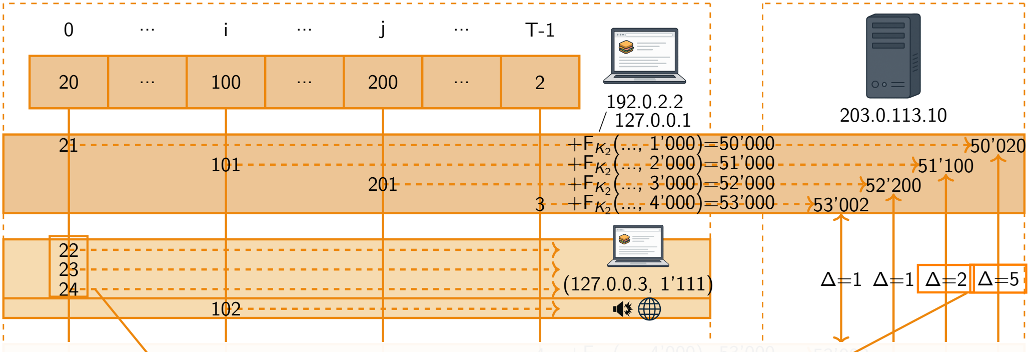
Phase 2

Influence of noise



Phase 2

Influence of noise

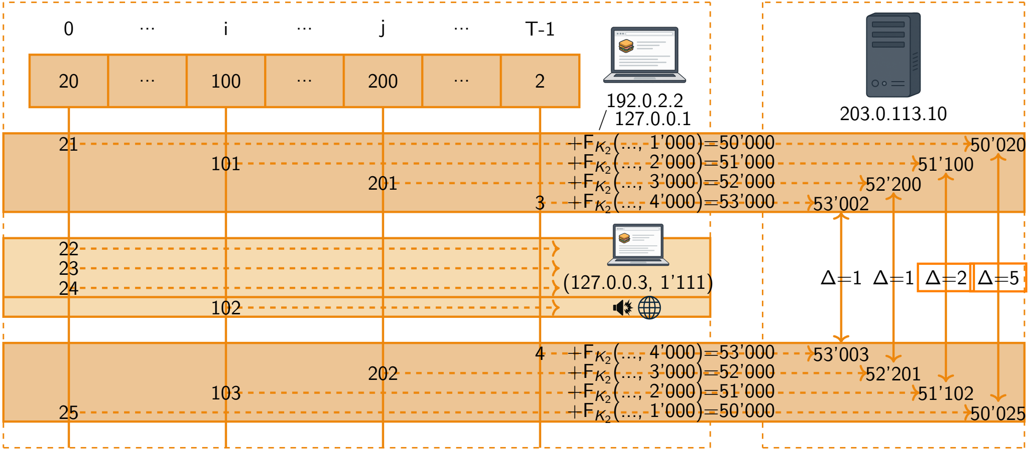


- Repeated connection attempts with same loopback 3-tuple

$\Rightarrow$ Any increments caused by noise are outweighed

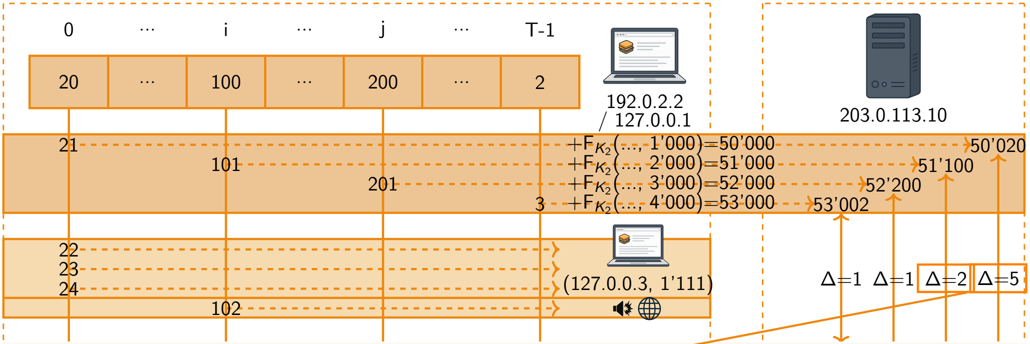
Phase 2

Condition for collision between attacker 3-tuple and loopback 3-tuple



Phase 2

Condition for collision between attacker 3-tuple and loopback 3-tuple



$C'(w) - C(w) > \varepsilon$ when sandwiching L_i as filling

Phase 2

Pseudocode

Algorithm 10 FINDING A DEVICE ID (PHASE 2)

```
procedure PHASE2
    C ← ∅; n ← 0; i ← 0
    repeat
        i ← i + 1
        C ← GETSOURCEPORTS(S')          // 1st burst
        ATTEMPTCONNECTTCP({L_i})        // 2nd burst
        C' ← GETSOURCEPORTS(S')         // 3rd burst
        w ← ιw(C'(w) − C(w) > ε)
        if DEFINED(B[w]) then           // Collision was found
            p ← p ∪ {(L_i, B[w])}
            n ← n + 1
        else
            B[w] ← L_i
    until n ≥ n*[i]                     // This is equiv. to P_T(i, n) ≤ p*
    return (p,i)
```

- i: Number of iterations
- n: Number of independent pairs / collisions of loopback 3-tuples
- L_i: Single loopback 3-tuple
- S': Unique attacker 3-tuples from Phase 1
- w: The one unique attacker 3-tuple
- ιw: Definite Description Operator, "the unique w" attacker 3-tuple
- $B[w]$: Dictionary, keys = unique attacker 3-tuples, values = first loopback 3-tuples
- $\{T_3, T_2, T_5\} \Rightarrow \{(T_3, T_2), (T_3, T_5)\}$

Phase 2

Pseudocode

Algorithm 11 FINDING A DEVICE ID (PHASE 2)

```
procedure PHASE2
    C ← ∅; n ← 0; i ← 0
    repeat
        i ← i + 1
        C ← GETSOURCEPORTS(S')          // 1st burst
        ATTEMPTCONNECTTCP({L_i})        // 2nd burst
        C' ← GETSOURCEPORTS(S')         // 3rd burst
        w ← ιw(C'(w) − C(w) > ε)
        if DEFINED(B[w]) then           // Collision was found
            p ← p ∪ {(L_i, B[w])}
            n ← n + 1
        else
            B[w] ← L_i
    until n ≥ n*[i]                     // This is equiv. to P_T(i, n) ≤ p*
    return (p,i)
```

- i: Number of iterations
- n: Number of independent pairs / collisions of loopback 3-tuples
- L_i: Single loopback 3-tuple
- S': Unique attacker 3-tuples from Phase 1
- w: The one unique attacker 3-tuple
- ιw: Definite Description Operator, "the unique w" attacker 3-tuple

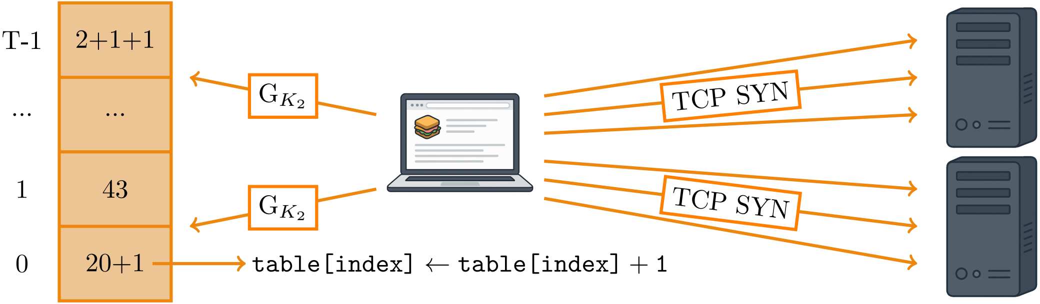
---

- $B[w]$: Dictionary, keys = unique attacker 3-tuples, values = first loopback 3-tuples
- $\{T_3, T_2, T_5\} \Rightarrow \{(T_3, T_2), (T_3, T_5)\}$

Countermeasures

Countermeasures Overview

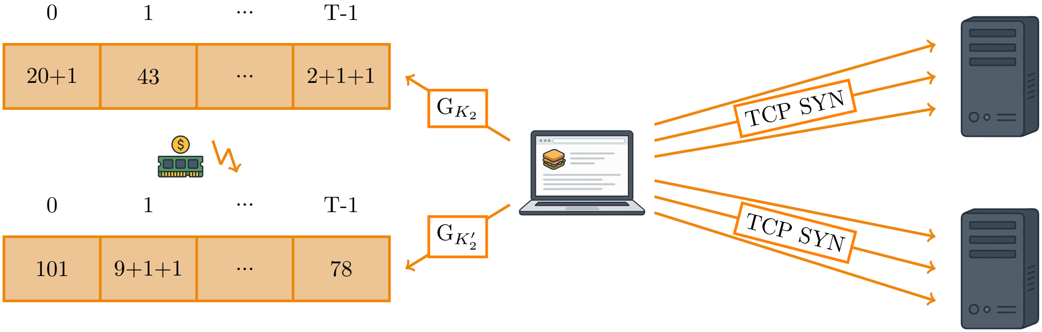
Root cause



- shared perturbation table ⇒ device-specific hash collisions
- predictable increment

Countermeasures Overview

Ideal solution



- separate perturbation table for each network context

⇒ high memory cost

Countermeasures Overview

General countermeasures

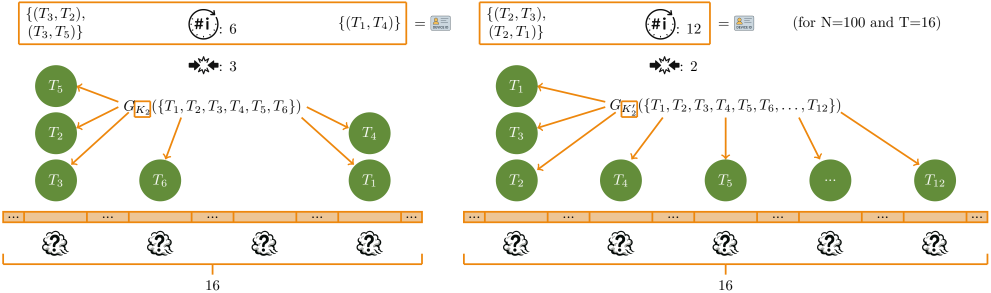
- make detecting hash collisions and
  - make it more time intensive:

  - limit time:

Countermeasures Overview

Implemented changes

- Periodic re-keying: Every 10 seconds

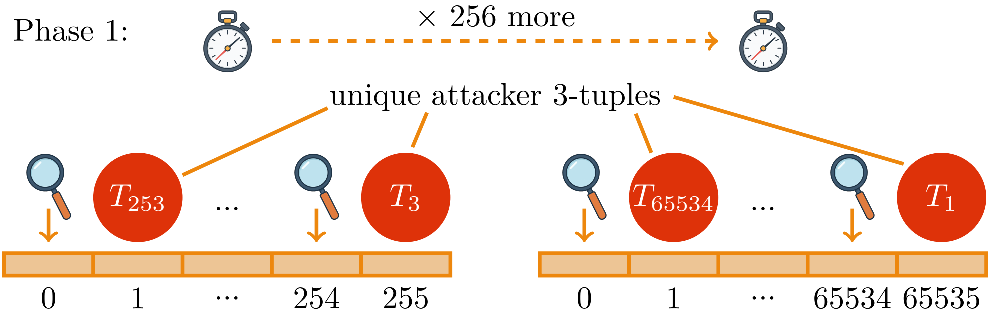


- Increase table size T: From 256 to 65536
- Introduce more noise: Each increment randomly between 1 and 8

Countermeasures Overview

Implemented changes

- Periodic re-keying: Every 10 seconds
- Increase table size T: From 256 to 65536



- Introduce more noise: Each increment randomly between 1 and 8

Countermeasures Overview

Implemented changes

- Periodic re-keying: Every 10 seconds
- Increase table size T: From 256 to 65536
- Introduce more noise: Each increment randomly between 1 and 8

$\texttt{table[index]} \leftarrow \texttt{table[index]} + 1 \longrightarrow \texttt{table[index]} \leftarrow \texttt{table[index]} + \texttt{rand(1, 8)}$

			T_4	T_3
20+1	43	⋯	11+1+1	2
0	1	⋯	T-2	T-1

larger ε necessary

			T_4	T_3
20+7	43	⋯	11+2+1	2
0	1	⋯	T-2	T-1

Countermeasures

Thanks for your attention!

Questions?

Appendix

Appendix

Phase 2 termination condition script

- https://github.com/matthejue/Seminar-Device-Tracking/blob/main/phase2_termination_condition.py

Appendix

Demonstration of the attack

- https://youtu.be/pZbfV5nCQsA?feature=shared

Phase 2

Termination condition

Probabiliy that all non-first loopback 3-tuples go into the same buckets

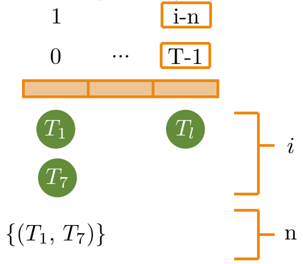
Combined probability of all first loopback 3-tuples in their buckets to match the structure

Occupied (non-empty) buckets

Correction because indices start at 0

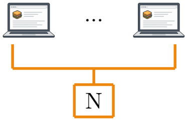
Maximum acceptable probability that any pair of devices yields same device ID

$$\underbrace{P_T(i,n) = \boxed{\frac{1}{T^n}} \cdot \boxed{\prod_{j=0}^{\boxed{i-n}-1}\left(1-\frac{j}{T}\right)}}_{\text{probability that a random device yields same device ID as device D}} \quad < \quad \underbrace{p^* = \boxed{\frac{c}{\binom{N}{2}}}}_{\text{threshold acceptance probability}}$$



$$\frac{T-j}{T} = \frac{T}{T} - \frac{j}{T} = 1 - \frac{j}{T}$$

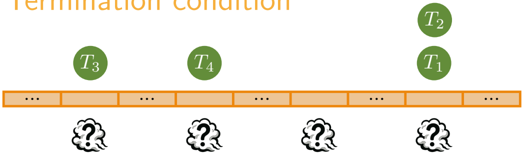
$\Rightarrow$ thus $n < i$



- T: Number of perturbation table cells
- N: Device population size
- c: Maximum average number of device ID collisions in a population
- i: Number of iterations
- n: Number of independent pairs / collisions

Phase 2

Termination condition



- Can only see which balls fall into same bucket
- Probability only depends on total number of independent collisions

- Correlations:
  - $\uparrow i \Rightarrow P_T(i,n)\downarrow \Rightarrow P_T(i,n) < p^*$ potentially closer to being satisfied

$$P_T(i,n) = \frac{1}{T^n} \cdot \prod_{j=0}^{\boxed{i}-n-1} \left(1 - \frac{j}{T}\right) \longrightarrow \text{multiply number} < 1 \text{ more often} \longrightarrow P_T(i,n)\downarrow$$

  - $\uparrow n \Rightarrow P_T(i,n)\downarrow \Rightarrow P_T(i,n) < p^*$ potentially closer to being satisfied

$$P_T(i,n) = \frac{1}{T^{\boxed{n}}} \cdot \prod_{j=0}^{i-\boxed{n}-1} \left(1 - \frac{j}{T}\right) \longrightarrow \text{multiply number} < 1 \text{ less often} \longrightarrow P_T(i,n)\uparrow$$

$$\longrightarrow \text{divide by large number} \longrightarrow P_T(i,n)\downarrow\downarrow$$

$$\longrightarrow P_T(i,n)\downarrow$$

Phase 2

Termination condition

n \ i	1	2	3	4	5	6	7	8	9	10	11	12	13	14
5	1	2	3	4	5	6	7	8	9	10	11	12	13	14
4	1	2	3	4	5	6	7	8	9	10	11	12	13	14
3	1	2	3	4	5	6	7	8	9	10	11	12	13	14
2	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

termination

20	⋯	2

N= 100   0   ⋯   15 +1=16=T

- Precompute table $n^*[i] = \min\{n \mid P_T(i,n) < p^*\}$
- $n \geq n^*[i]$ replaces $P_T(i,n) < p^*$

$\Rightarrow$ Avoid repeated computation

i-range	$n^*[i]$
4–5	4
6–11	3
12–12	2
13–13	1
14	0

Literature

Literature I

[1] Moshe Kol, Amit Klein, and Yossi Gilad. *Device Tracking via Linux's New TCP Source Port Selection Algorithm (Extended Version)*. 2022. arXiv: 2209.12993 [cs.CR]. URL: https://arxiv.org/abs/2209.12993.

[2] Michael Larsen and Fernando Gont. *Recommendations for Transport-Protocol Port Randomization*. Request for Comments RFC 6056. Internet Engineering Task Force, Jan. 2011. 29 pp. DOI: 10.17487/RFC6056. URL: https://datatracker.ietf.org/doc/rfc6056 (visited on 07/03/2025).

[3] *Meltdown (Security Vulnerability)*. In: *Wikipedia*. Dec. 26, 2024. URL: https://en.wikipedia.org/w/index.php?title=Meltdown_(security_vulnerability)&oldid=1265360095 (visited on 07/16/2025).

[4] *Spectre (Security Vulnerability)*. In: *Wikipedia*. June 16, 2025. URL: https://en.wikipedia.org/w/index.php?title=Spectre_(security_vulnerability)&oldid=1295923886 (visited on 07/16/2025).

Literature II

- All other images were generated using ChatGPT, which internally utilizes DALL-E 3
- The visualisations were created with TikZ and TikZiT