

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 29, 2025

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

Introduction

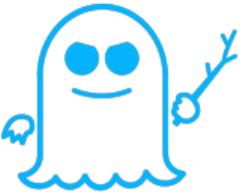
Motivation

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
→



- ▶ 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 ⇒ device-specific key

Browser-based device tracking

Types of techniques

- ▶ Tagging techniques: Insert an ID



- ▶ Fingerprinting techniques: Measure characteristics



generate fingerprint



Source Port Selection

4-Tuple

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



$$\begin{array}{ccc} (123 \text{ Maple St., Apt. 5B}) & & (456 \text{ Oak St., Apt. 9D}) \\ \hline (192.0.2.123, 55'555) & & (203.0.113.456, 49'999) \end{array} \rightarrow$$



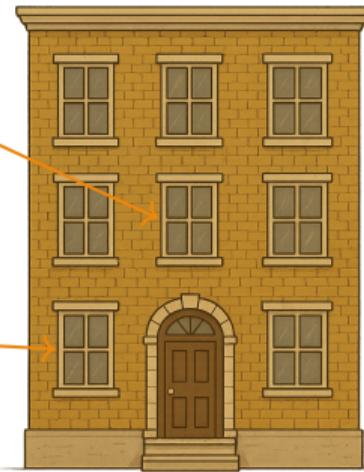
- ◆ $(192.0.2.123, 55'555, 203.0.113.10, 49'999)$
- ◆ $(123 \text{ Maple St., Apt. 5B, 456 Oak St., Apt. 9D})$

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

Source Port Selection

Goals

- **Functionality:** Reusing source ports can cause failures
 - ⇒ Earlier connections might be active (TIME_WAIT state)



Source Port Selection

Goals

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



(123 Maple St., Apt. 5B)

(456 Oak St., Apt. 9D)



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



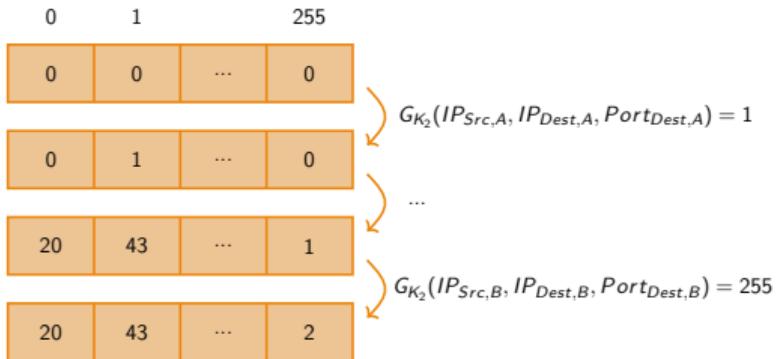
Double-Hash Port Selection Algorithm (DHPS)

Algorithm Overview

- ▶ Complete 3-tuple to 4-tuple with cryptographic keyed-hash functions:

- ◆ $F_{K_1} : 3\text{-Tuples} \xrightarrow{K_1} \{0, \dots, 2^{32}\}$
- ◆ $G_{K_2} : 3\text{-Tuples} \xrightarrow{K_2} \{0, \dots, T\}$ where T is length of perturbation table

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



Double-Hash Port Selection Algorithm (DHPS)

Pseudocode

Algorithm 1 DHPS SOURCE PORT SELECTION

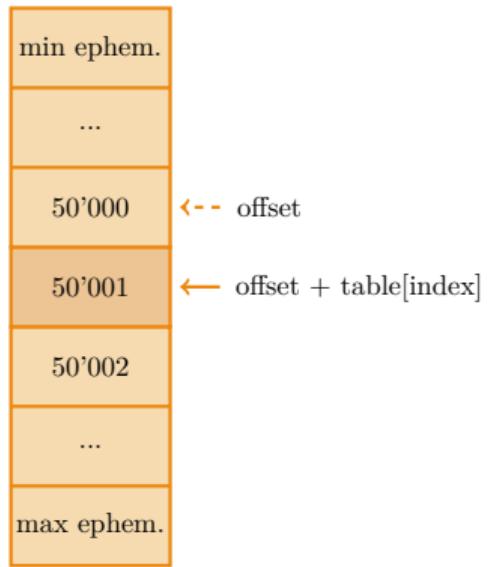
```

1 procedure SELECTEPHEMERALPORT
2   num_ephemeral  $\leftarrow$  max_ephemeral - min_ephemeral + 1
3   offset  $\leftarrow$   $F_{K_1}(IP_{SRC}, IP_{DST}, PORT_{DST})$ 
4   index  $\leftarrow$   $G_{K_2}(IP_{SRC}, IP_{DST}, PORT_{DST})$ 
5   count  $\leftarrow$  num_ephemeral
6   repeat
7     port  $\leftarrow$  min_ephemeral +
8        $((offset + table[index]) \bmod num\_ephemeral)$ 
9     table[index]  $\leftarrow$  table[index] + 1
10    if CHECKSUITABLEPORT(port) then
11      return port
12      count  $\leftarrow$  count - 1
13    until count = 0
14    return Error

```

► Source port calculation:

- ◆ $offset = 50'000$
- ◆ $index = 255$



Double-Hash Port Selection Algorithm (DHPS)

Pseudocode

Algorithm 2 DHPS SOURCE PORT SELECTION

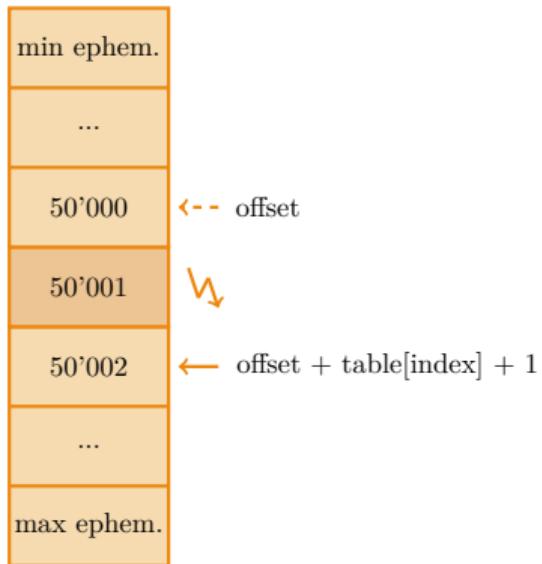
```

1 procedure SELECTEPHEMERALPORT
2     num_ephemeral  $\leftarrow$  max_ephemeral – min_ephemeral + 1
3     offset  $\leftarrow$   $F_{K_1}(IP_{SRC}, IP_{DST}, PORT_{DST})$ 
4     index  $\leftarrow$   $G_{K_2}(IP_{SRC}, IP_{DST}, PORT_{DST})$ 
5     count  $\leftarrow$  num_ephemeral
6     repeat
7         port  $\leftarrow$  min_ephemeral +
8              $((offset + table[index]) \bmod num\_ephemeral)$ 
9         table[index]  $\leftarrow$  table[index] + 1
10        if CHECKSUITABLEPORT(port) then
11            return port
12        count  $\leftarrow$  count – 1
13    until count = 0
14    return Error

```

► Source port calculation:

- ◆ $offset = 50'000$
- ◆ $index = 255$



Double-Hash Port Selection Algorithm (DHPS)

Pseudocode

Algorithm 3 DHPS SOURCE PORT SELECTION

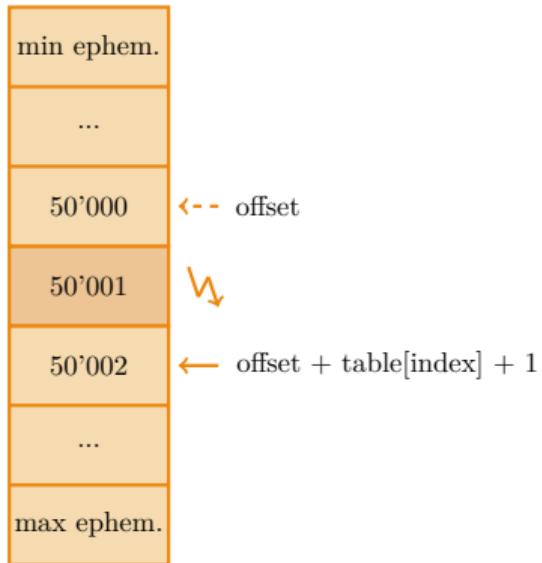
```

1 procedure SELECTEPHEMERALPORT
2     num_ephemeral  $\leftarrow$  max_ephemeral – min_ephemeral + 1
3     offset  $\leftarrow$   $F_{K_1}(IP_{SRC}, IP_{DST}, PORT_{DST})$ 
4     index  $\leftarrow$   $G_{K_2}(IP_{SRC}, IP_{DST}, PORT_{DST})$ 
5     count  $\leftarrow$  num_ephemeral
6     repeat
7         port  $\leftarrow$  min_ephemeral +
8              $((offset + table[index]) \bmod num\_ephemeral)$ 
9         table[index]  $\leftarrow$  table[index] + 1
10        if CHECKSUITABLEPORT(port) then
11            return port
12            count  $\leftarrow$  count - 1
13        until count = 0
14        return Error

```

► Source port calculation:

- ◆ $offset = 50'000$
- ◆ $index = 255$



Double-Hash Port Selection Algorithm (DHPS)

Pseudocode

Algorithm 4 DHPS SOURCE PORT SELECTION

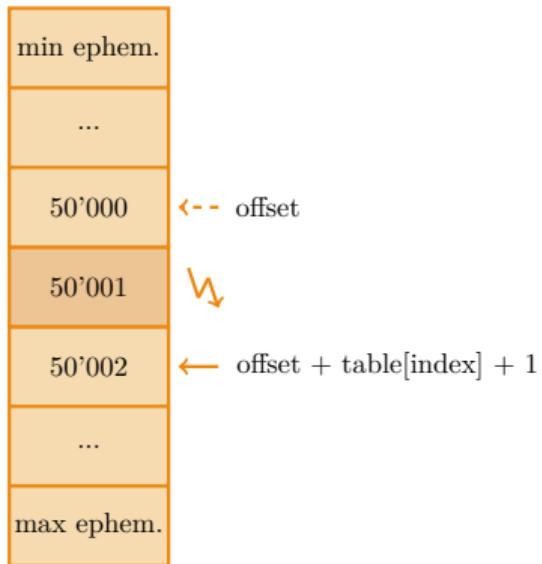
```

1 procedure SELECTEPHEMERALPORT
2     num_ephemeral  $\leftarrow$  max_ephemeral – min_ephemeral + 1
3     offset  $\leftarrow$   $F_{K_1}(IP_{SRC}, IP_{DST}, PORT_{DST})$ 
4     index  $\leftarrow$   $G_{K_2}(IP_{SRC}, IP_{DST}, PORT_{DST})$ 
5     count  $\leftarrow$  num_ephemeral
6     repeat
7         port  $\leftarrow$  min_ephemeral +
8              $((offset + table[index]) \bmod num\_ephemeral)$ 
9         table[index]  $\leftarrow$  table[index] + 1
10        if CHECKSUITABLEPORT(port) then
11            return port
12            count  $\leftarrow$  count – 1
13        until count = 0
14    return Error

```

► Source port calculation:

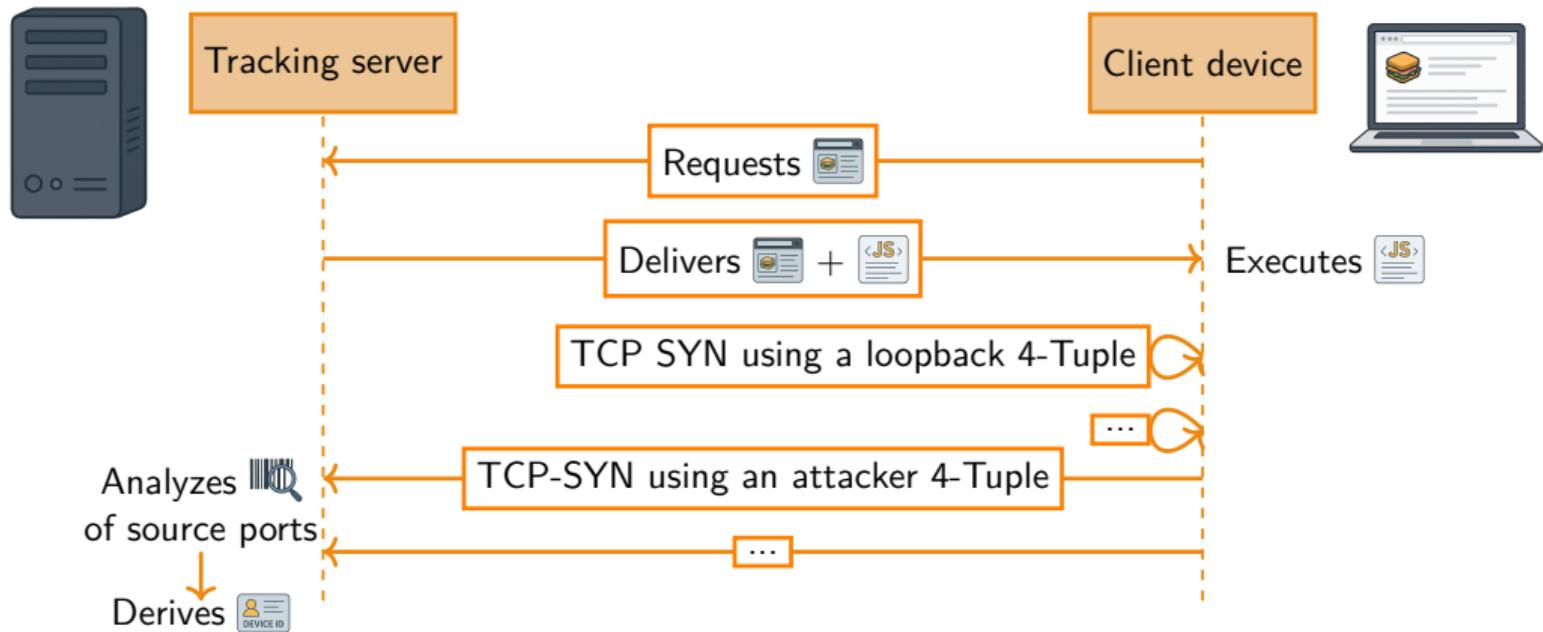
- ◆ $offset = 50'000$
- ◆ $index = 255$



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:

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


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

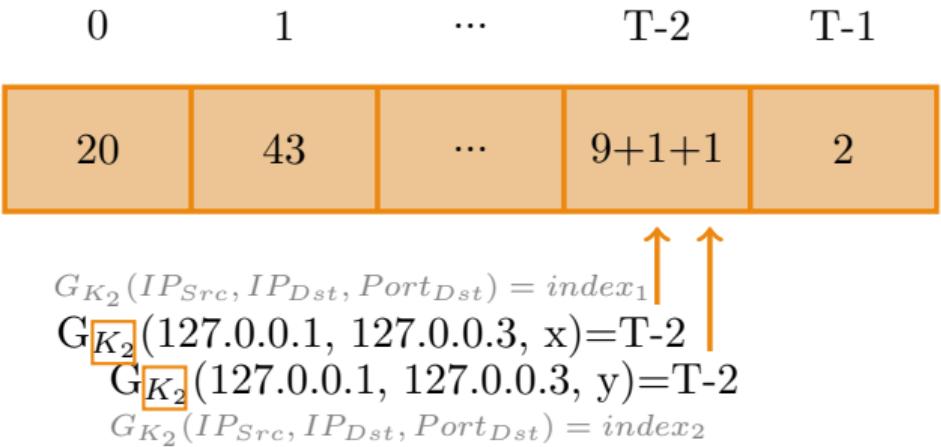
=

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

Device ID

Collisions

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



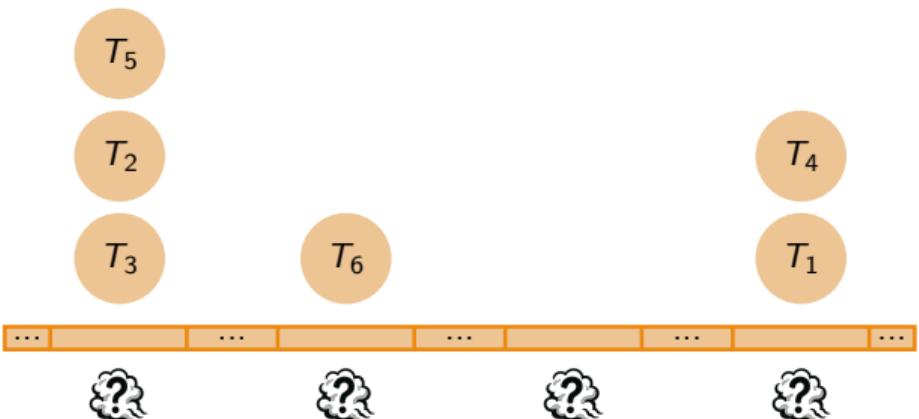
- Collisions depend on K_2
- K_2 randomly generated at 
- ⇒ Persists across  and 
- ⇒ Also  persists

Devide ID

Independant pairs

- ▶ Consists of set of all such colliding **independant** pairs:

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



- ▶ $\boxed{\#i: 6}$ = Device ID
- ▶ $\blacktriangleleft: 3$, evidence how K_2 maps into **perturbation table** cells

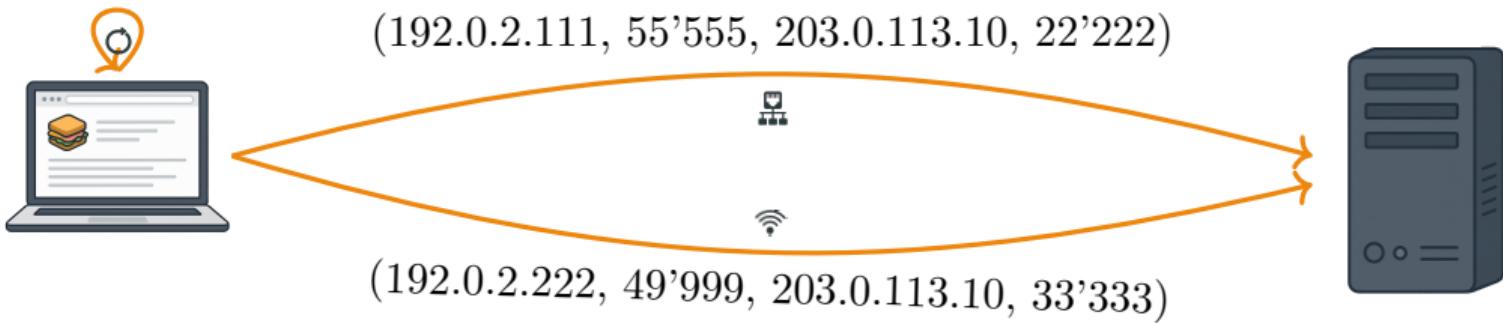
- ▶ Why?
 - ◆ $\binom{3}{2}$ pairs of $\{T_3, T_2, T_5\}$
 - ◆ $3 - 1$ are independant
 - ◆ $\{(T_3, T_2), (T_3, T_5)\}$ imply (T_2, T_5)

Device ID

Why loopback 3-tuples?

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

(127.0.0.1, 48'888, 127.0.0.3, 11'111)

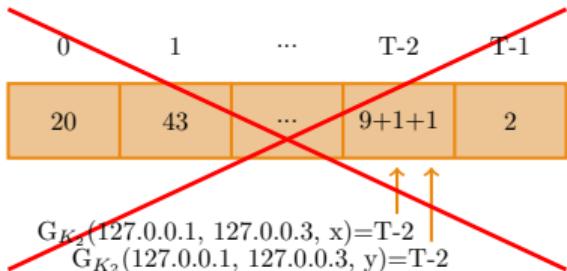
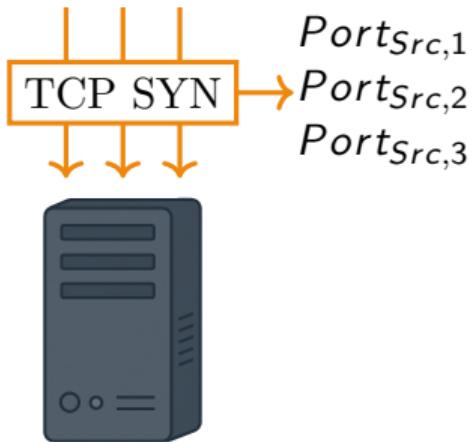


⇒ Loopback 3-tuples used for generating 

Sandwiching Technique

Cannot directly observe collisions

- no access to kernel internals



- side effect: predictable increment

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

- analyse source port increments via
- ⇒ derive

Sandwiching Technique

General principle

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

Sandwiching Technique

General principle

Step	Calculation
Wrapper	$i = G_{K_2}(IP_{Src,X}, IP_{Dst,X}, Port_{Dst,X})$ $offset_X = F_{K_1}(IP_{Src,X}, IP_{Dst,X}, Port_{Dst,X})$
Filling	$j = G_{K_2}(IP_{Src,Y}, IP_{Dst,Y}, Port_{Dst,Y})$ $offset_Y = F_{K_1}(IP_{Src,Y}, IP_{Dst,Y}, Port_{Dst,Y})$
Wrapper	$i = G_{K_2}(IP_{Src,X}, IP_{Dst,X}, Port_{Dst,X})$ $offset_X = F_{K_1}(IP_{Src,X}, IP_{Dst,X}, Port_{Dst,X})$

Sandwiching Technique

General principle

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

Sandwiching Technique

General principle

Step	Calculation
Wrapper	$table[j] = \begin{cases} rand(j) + 0, & \text{if } i \neq j, \text{ meaning perturbation table cell } i \text{ was not used,} \\ rand(j) + 1, & \text{if } i = j, \text{ meaning perturbation table cell } i \text{ was used} \end{cases}$ $\text{offset}_Y = F_{K_1}(IP_{Src,Y}, IP_{Dst,Y}, Port_{Dst,Y})$ $table[i] = \begin{cases} rand(i) + 1, & \text{if } i \neq j, \text{ meaning perturbation table cell } i \text{ was not used,} \\ rand(i) + 2, & \text{if } i = j, \text{ meaning perturbation table cell } i \text{ was used} \end{cases}$ $j = G_{K_2}(IP_{Src,Y}, IP_{Dst,Y}, Port_{Dst,Y})$ $\text{offset}_Y = F_{K_1}(IP_{Src,Y}, IP_{Dst,Y}, Port_{Dst,Y})$ $\text{Port}_{Src,2} = \text{min_ephemeral} + (\text{offset}_Y + \boxed{table[j]}) \bmod \text{num_ephemeral}$ $table[j] = table[j] + 1$
Filling	$i = G_{K_2}(IP_{Src,X}, IP_{Dst,X}, Port_{Dst,X})$ $\text{offset}_X = F_{K_1}(IP_{Src,X}, IP_{Dst,X}, Port_{Dst,X})$ $\text{Port}_{Src,3} = \text{min_ephemeral} + (\text{offset}_X + \boxed{table[i]}) \bmod \text{num_ephemeral}$ $table[i] = table[i] + 1$
Wrapper	$i = G_{K_2}(IP_{Src,X}, IP_{Dst,X}, Port_{Dst,X})$ $\text{offset}_X = F_{K_1}(IP_{Src,X}, IP_{Dst,X}, Port_{Dst,X})$ $\text{Port}_{Src,3} = \text{min_ephemeral} + (\text{offset}_X + \boxed{table[i]}) \bmod \text{num_ephemeral}$ $table[i] = table[i] + 1$

Sandwiching Technique

General principle

Step	Calculation
Wrapper	$i = G_{K_2}(\text{IP}_{\text{Src},X}, \text{IP}_{\text{Dst},X}, \text{Port}_{\text{Dst},X})$ $\text{offset}_X = F_{K_1}(\text{IP}_{\text{Src},X}, \text{IP}_{\text{Dst},X}, \text{Port}_{\text{Dst},X})$ $\text{Port}_{\text{Src},1} = \text{min_ephemeral} + (\text{offset}_X + \text{table}[i]) \bmod \text{num_ephemeral}$ $\text{table}[i] = \text{table}[i] + 1$

Collision between 3-tuples X and Y

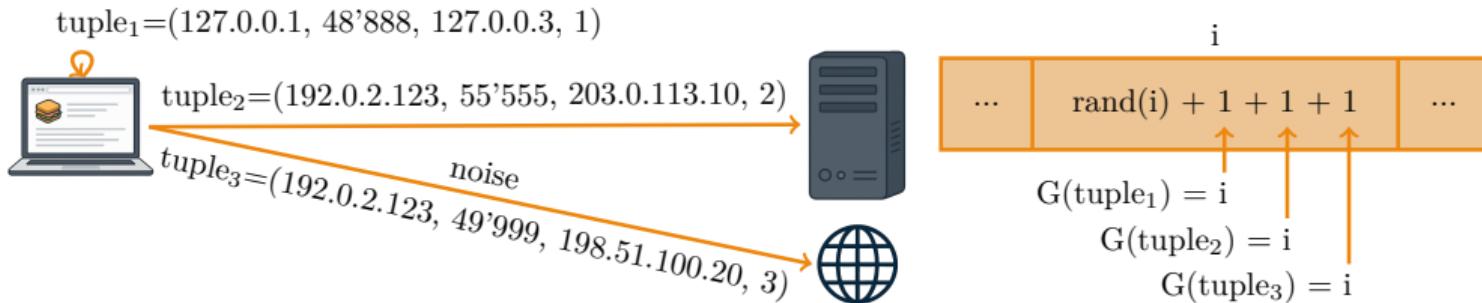
- ↔ G_{K_2} maps 3-tuples X and Y to same perturbation table cell
- ↔ $\text{Port}_{\text{Src},3}$ is greater than $\text{Port}_{\text{Src},1} + 1$

Step	Calculation
Wrapper	$i = G_{K_2}(\text{IP}_{\text{Src},X}, \text{IP}_{\text{Dst},X}, \text{Port}_{\text{Dst},X})$ $\text{offset}_X = F_{K_1}(\text{IP}_{\text{Src},X}, \text{IP}_{\text{Dst},X}, \text{Port}_{\text{Dst},X})$ $\text{Port}_{\text{Src},3} = \text{min_ephemeral} + (\text{offset}_X + \text{table}[i]) \bmod \text{num_ephemeral}$ $\text{table}[i] = \text{table}[i] + 1$

Sandwiching Technique

To consider in practise

- ▶ Background noise



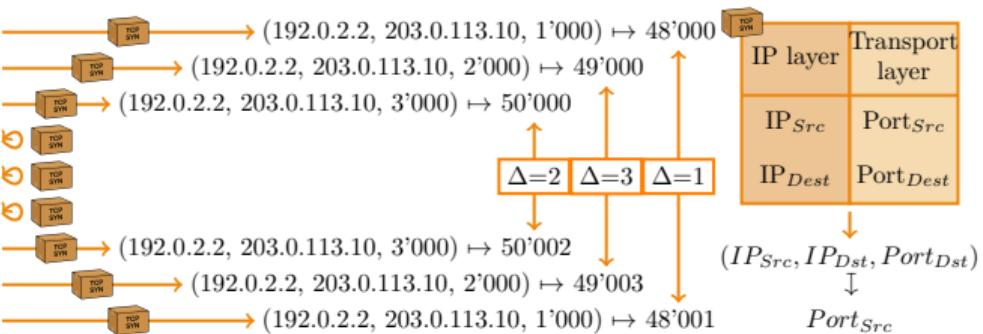
- ▶ Multiple 3-tuples used in steps
- ▶ Limitations for loopback 3-tuples

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)

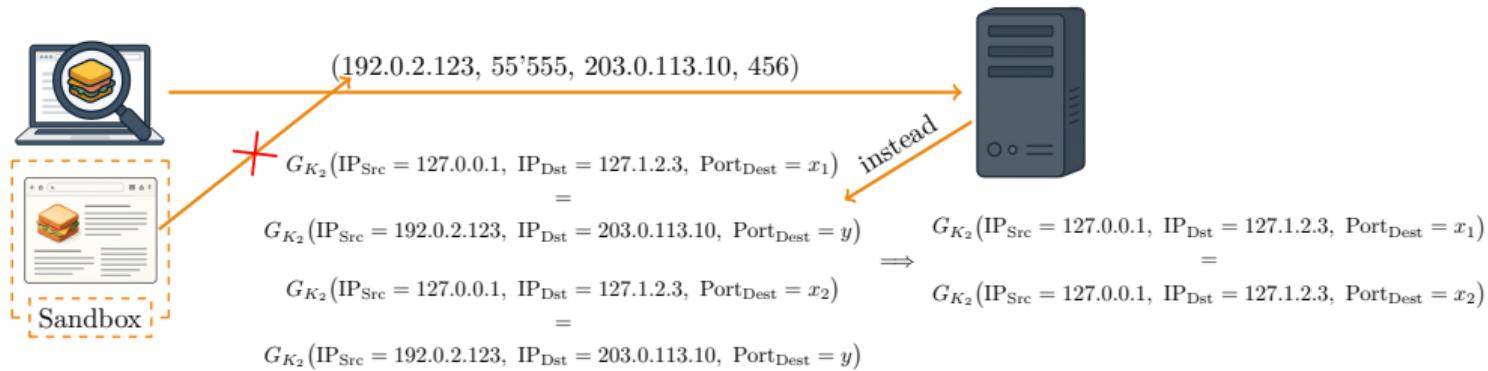


- ▶ Limitations for loopback 3-tuples

Sandwiching Technique

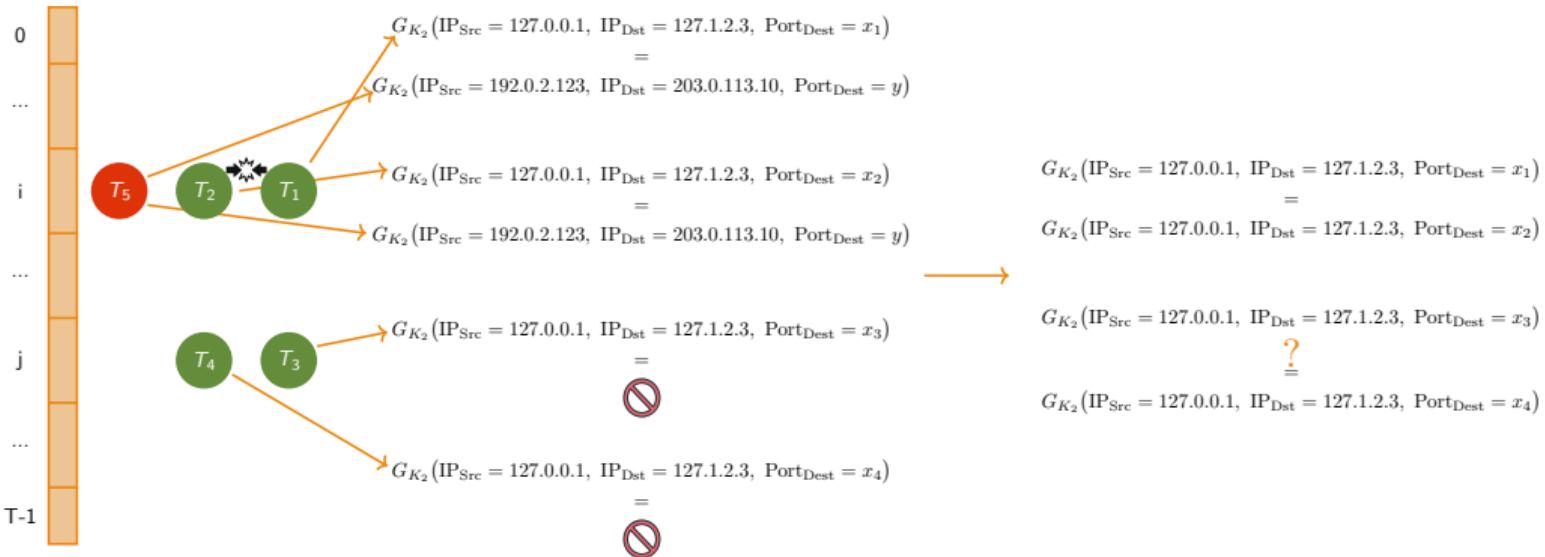
To consider in practise

- ▶ Background noise
- ▶ Multiple 3-tuples used in steps
- ▶ Limitations for loopback 3-tuples



Phase 1

Reason for Phase 1



- ▶ Every cell should be testable for collisions
- ⇒ Set of unique attacker 3-tuples that collectively cover all cells

Phase 1

Pseudocode

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

```

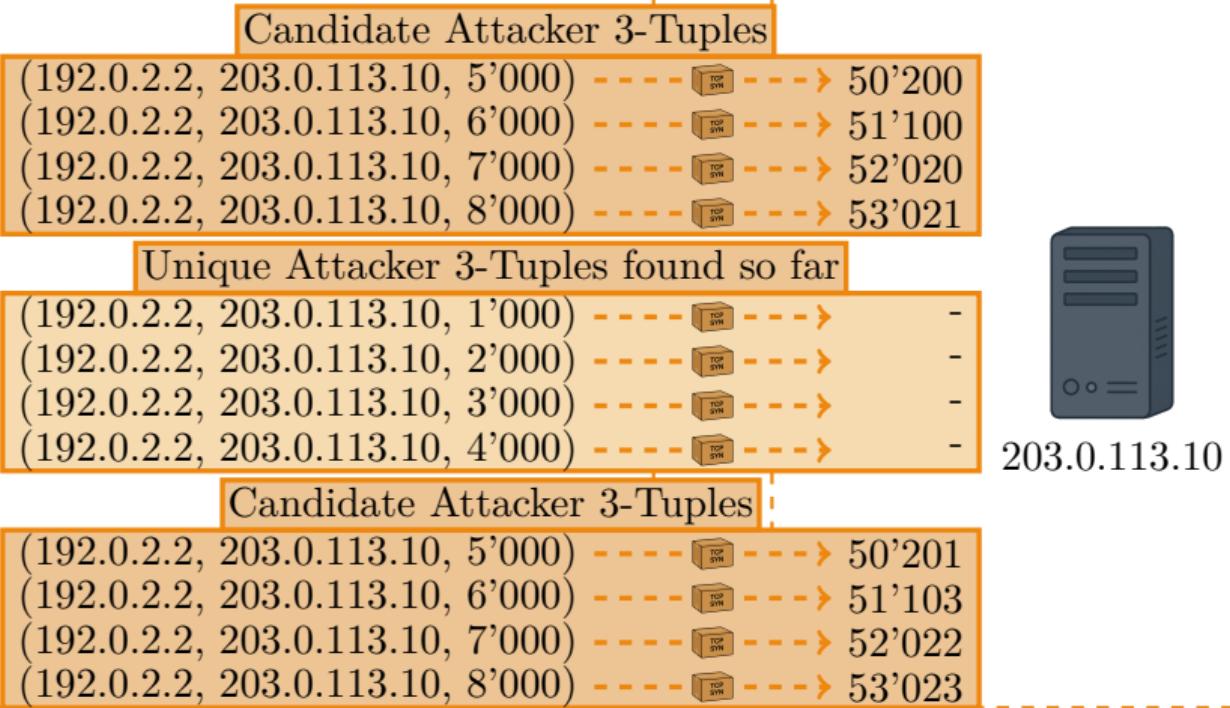
1 procedure SENDBURST(x)
2   for all x ∈ X do
3     ATTEMPTCONNECTTCP(x)
4 procedure GETSOURCEPORTS(u)
5   SENDBURST(u)
6   R ← RECEIVEATTACKERTUPLETOPORTMAP()
7   return R
8 procedure PHASE1
9   S' ← ∅
10  while |S'| < T do
11    S ← GETNEWEXTERNALDESTINATIONS()
12    P ← GETSOURCEPORTS(S) // 1st burst
13    SENDBURST(S') // 2nd burst
14    P' ← GETSOURCEPORTS(S) // 3rd burst
15    S' ← S' ∪ {x | P'(x) - P(x) = 1}
16  return S'

```

- ▶ S : New candidate attacker 3-tuples
- ▶ S' : Unique attacker 3-tuples found so far (\emptyset at the beginning)
- ▶ P, P' : Function for all new candidate attacker 3-tuples in 1st (P) and 2nd wrapper (P') a TCP-SYN mapping $(IP_{Src}, IP_{Dest}, Port_{Dest}) \mapsto Port_{Src}$
- ▶ x : Candidate attacker 3-tuple

Phase 1

Overview



Phase 1

Pseudocode

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

```

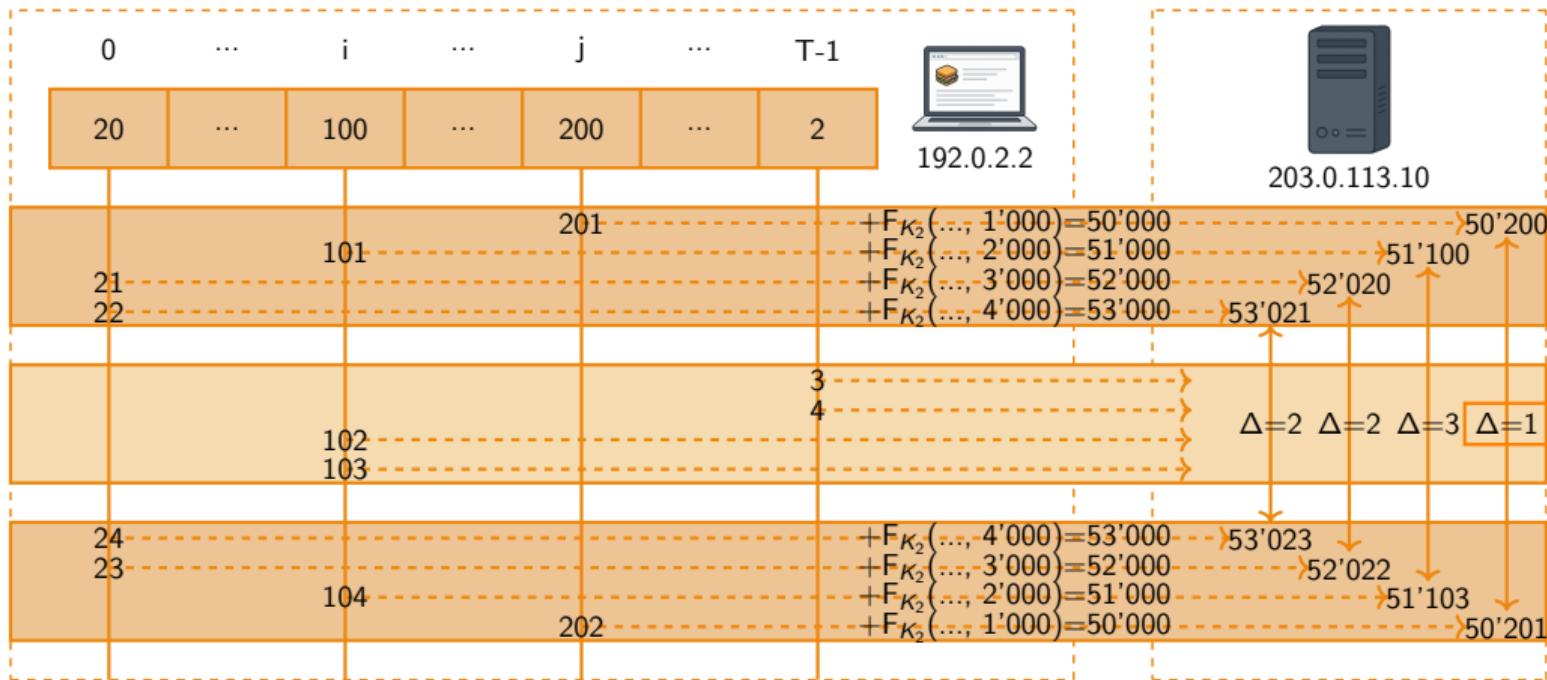
1 procedure SENDBURST(x)
2   for all x ∈ X do
3     ATTEMPTCONNECTTCP(x)
4 procedure GETSOURCEPORTS(u)
5   SENDBURST(u)
6   R ← RECEIVEATTACKERTUPLETOPORTMAP()
7   return R
8 procedure PHASE1
9   S' ← ∅
10  while |S'| < T do
11    S ← GETNEWEXTERNALDESTINATIONS()
12    P ← GETSOURCEPORTS(S) // 1st burst
13    SENDBURST(S') // 2nd burst
14    P' ← GETSOURCEPORTS(S) // 3rd burst
15    S' ← S' ∪ {x | P'(x) - P(x) = 1}
16  return S'

```

- ▶ S : New candidate attacker 3-tuples
- ▶ S' : Unique attacker 3-tuples found so far (\emptyset at the beginning)
- ▶ P, P' : Function for all new candidate attacker 3-tuples in 1st (P) and 2nd wrapper (P') a TCP-SYN mapping $(IP_{Src}, IP_{Dest}, Port_{Dest}) \mapsto Port_{Src}$
- ▶ x : Candidate attacker 3-tuple
- ▶ 3-tuples only incremented by 1
 ⇒ Does not collide with unique attacker 3-tuples found so far, else increment > 1
 ⇒ First to map to a certain table cell

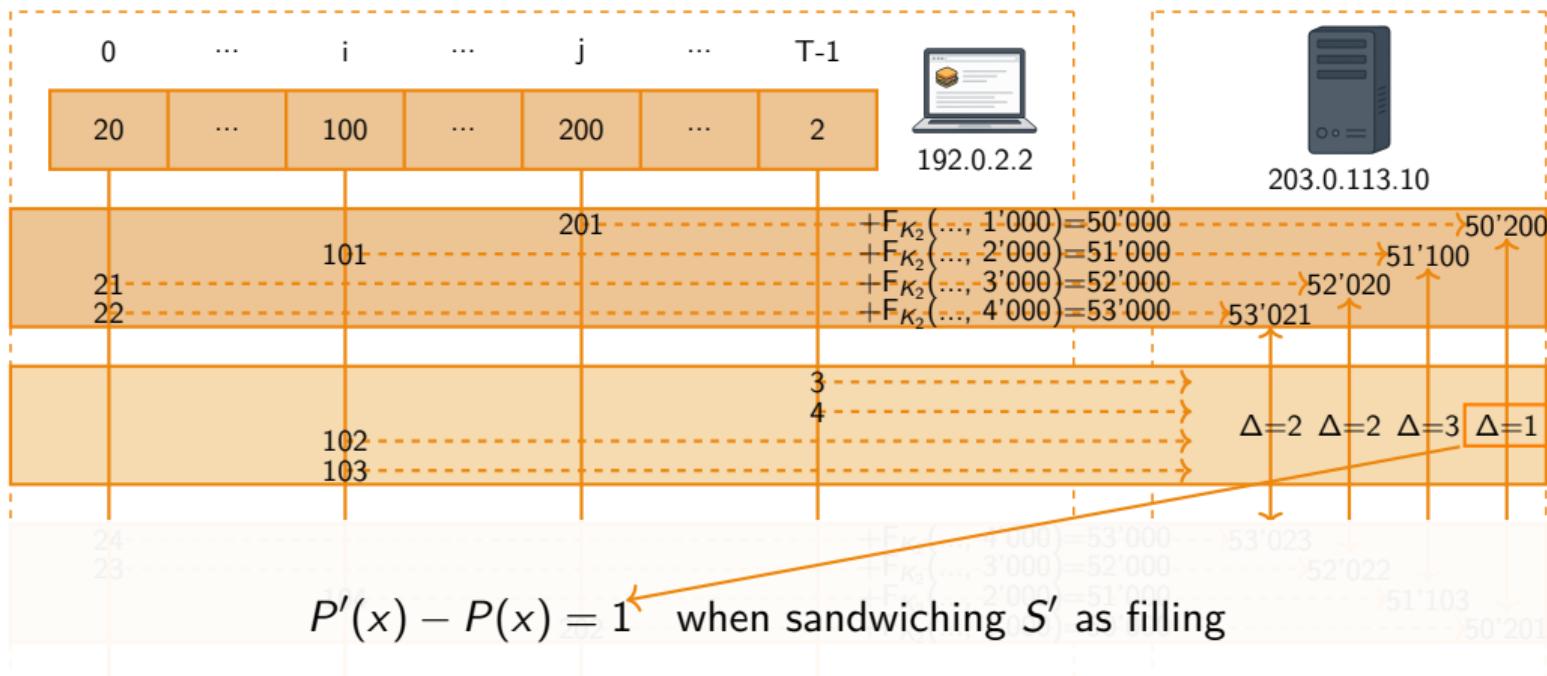
Phase 1

Condition for unique attacker 3-tuple



Phase 1

Condition for unique attacker 3-tuple



Phase 1

Pseudocode

Algorithm 7 FINDING ATTACKER 3-TUPLE PER CELL (PHASE 1)

```

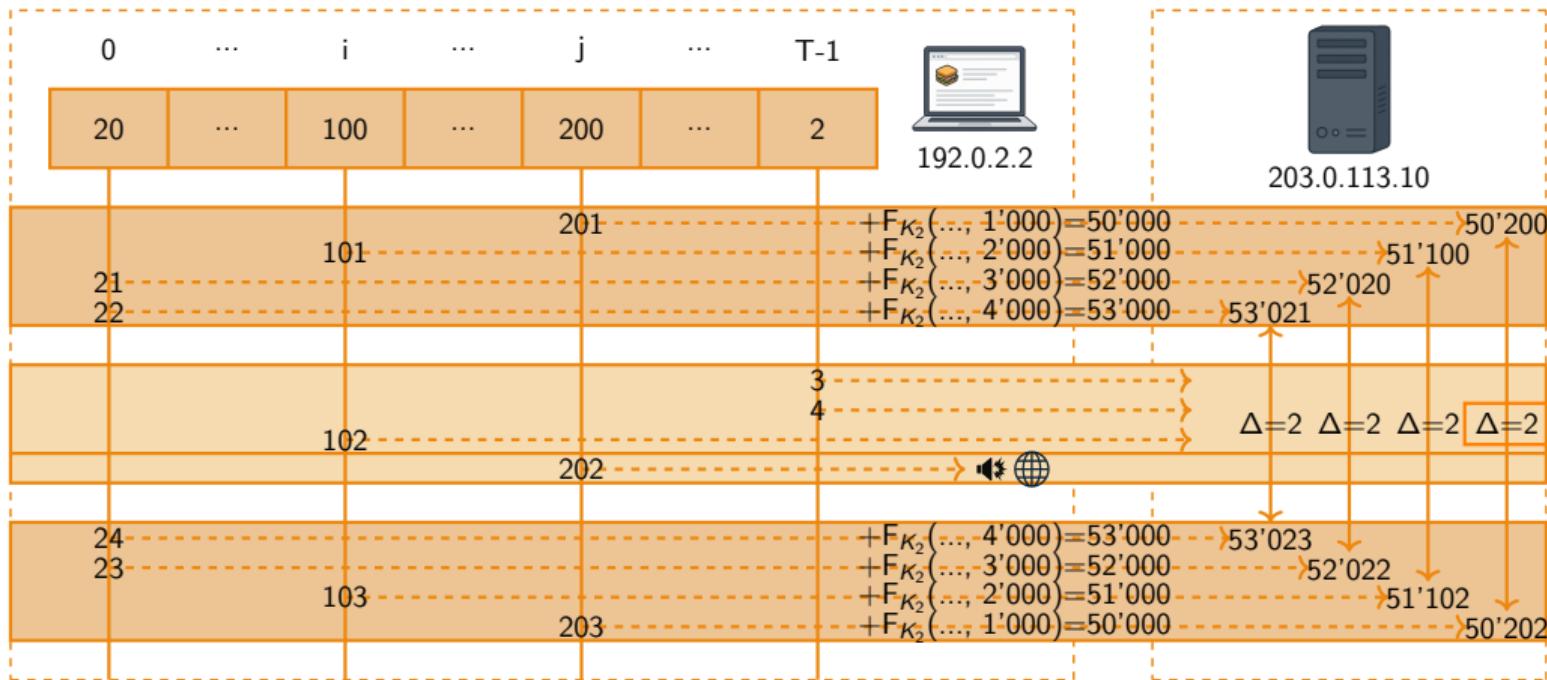
1 procedure SENDBURST(x)
2   for all x ∈ X do
3     ATTEMPTCONNECTTCP(x)
4 procedure GETSOURCEPORTS(u)
5   SENDBURST(u)
6   R ← RECEIVEATTACKERTUPLETOPORTMAP()
7   return R
8 procedure PHASE1
9   S' ← ∅
10  while |S'| < T do
11    S ← GETNEWEXTERNALDESTINATIONS()
12    P ← GETSOURCEPORTS(S) // 1st burst
13    SENDBURST(S') // 2nd burst
14    P' ← GETSOURCEPORTS(S) // 3rd burst
15    S' ← S' ∪ {x | P'(x) - P(x) = 1}
16  return S'

```

- ▶ S : New candidate attacker 3-tuples
- ▶ S' : Unique attacker 3-tuples found so far (\emptyset at the beginning)
- ▶ P, P' : Function for all new candidate attacker 3-tuples in 1st (P) and 2nd wrapper (P') a TCP-SYN mapping $(IP_{Src}, IP_{Dest}, Port_{Dest}) \mapsto Port_{Src}$
- ▶ x : Candidate attacker 3-tuple
- ▶ 3-tuples only incremented by 1
 ⇒ Does not collide with unique attacker 3-tuples found so far, else increment > 1
 ⇒ First to map to a certain table cell

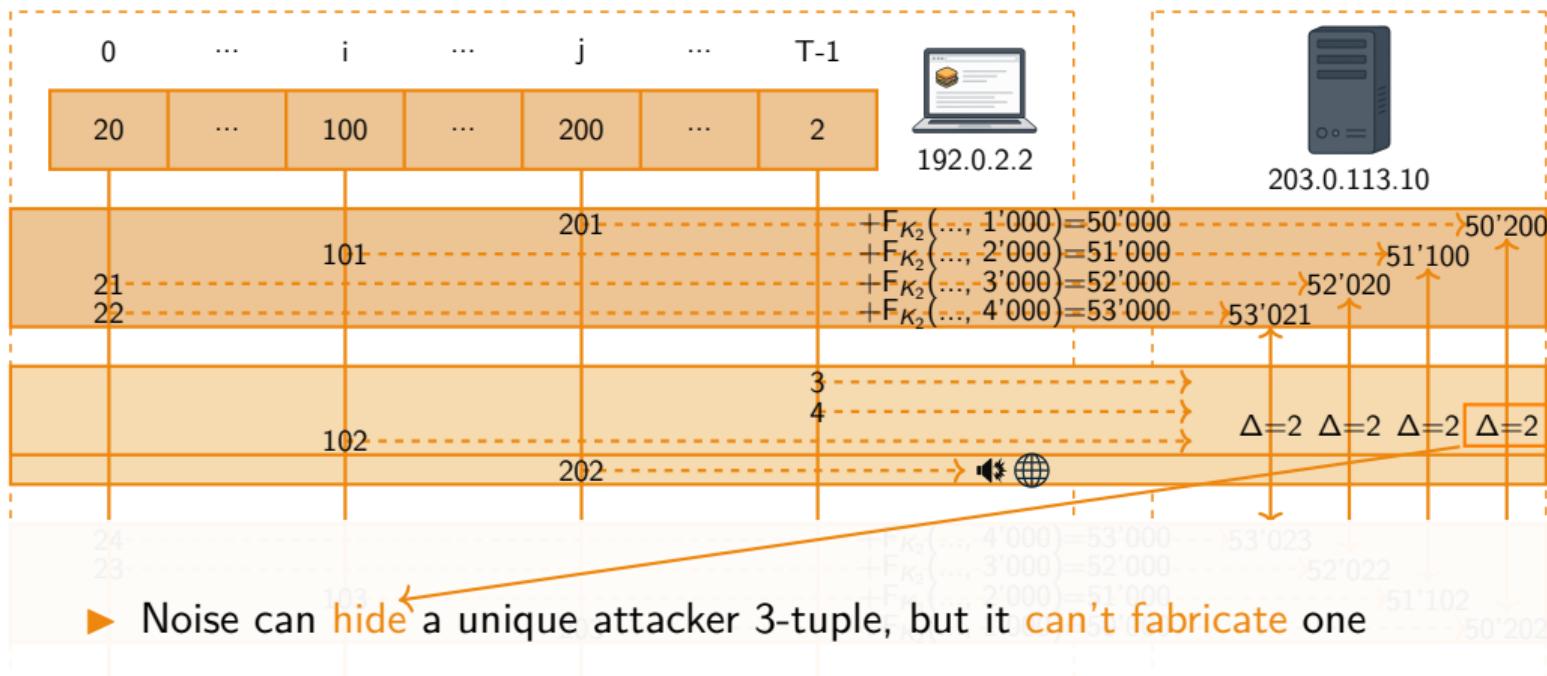
Phase 1

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)

```

1 procedure PHASE2
2   C  $\leftarrow \emptyset$ ; n  $\leftarrow 0$ ; i  $\leftarrow 0$ 
3   repeat
4     i  $\leftarrow i + 1$ 
5     P  $\leftarrow$  GETSOURCEPORTS(s') // 1st burst
6     ATTEMPTCONNECTTCP({Li}) // 2nd burst
7     P'  $\leftarrow$  GETSOURCEPORTS(s') // 3rd burst
8     w  $\leftarrow \iota w (P'(w) - P(w) > \varepsilon)$ 
9     if DEFINED(Bw) then // A collision was found
10      C  $\leftarrow C \cup \{(L_i, B[w])\}$ 
11      n  $\leftarrow n + 1$ 
12    else
13      | B[w]  $\leftarrow L_i$ 
14    until n  $\geq n^*[i]$  // This is equiv. to PDi(n)  $\leq p^*$ 
15    return (C, 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



192.0.2.2
/ 127.0.0.1



203.0.113.10

Unique Attacker 3-Tuples from Phase1

- (192.0.2.2, 203.0.113.10, 1'000) 50'020
- (192.0.2.2, 203.0.113.10, 2'000) 51'100
- (192.0.2.2, 203.0.113.10, 3'000) 52'200
- (192.0.2.2, 203.0.113.10, 4'000) 53'002

Loopback 3-Tuple

- (127.0.0.1, 127.0.0.3, 1'111) |

Unique Attacker 3-Tuples from Phase1

- (192.0.2.2, 203.0.113.10, 1'000) 50'022
- (192.0.2.2, 203.0.113.10, 2'000) 51'101
- (192.0.2.2, 203.0.113.10, 3'000) 52'201
- (192.0.2.2, 203.0.113.10, 4'000) 53'003

Phase 2

Pseudocode

Algorithm 9 FINDING A DEVICE ID (PHASE 2)

```

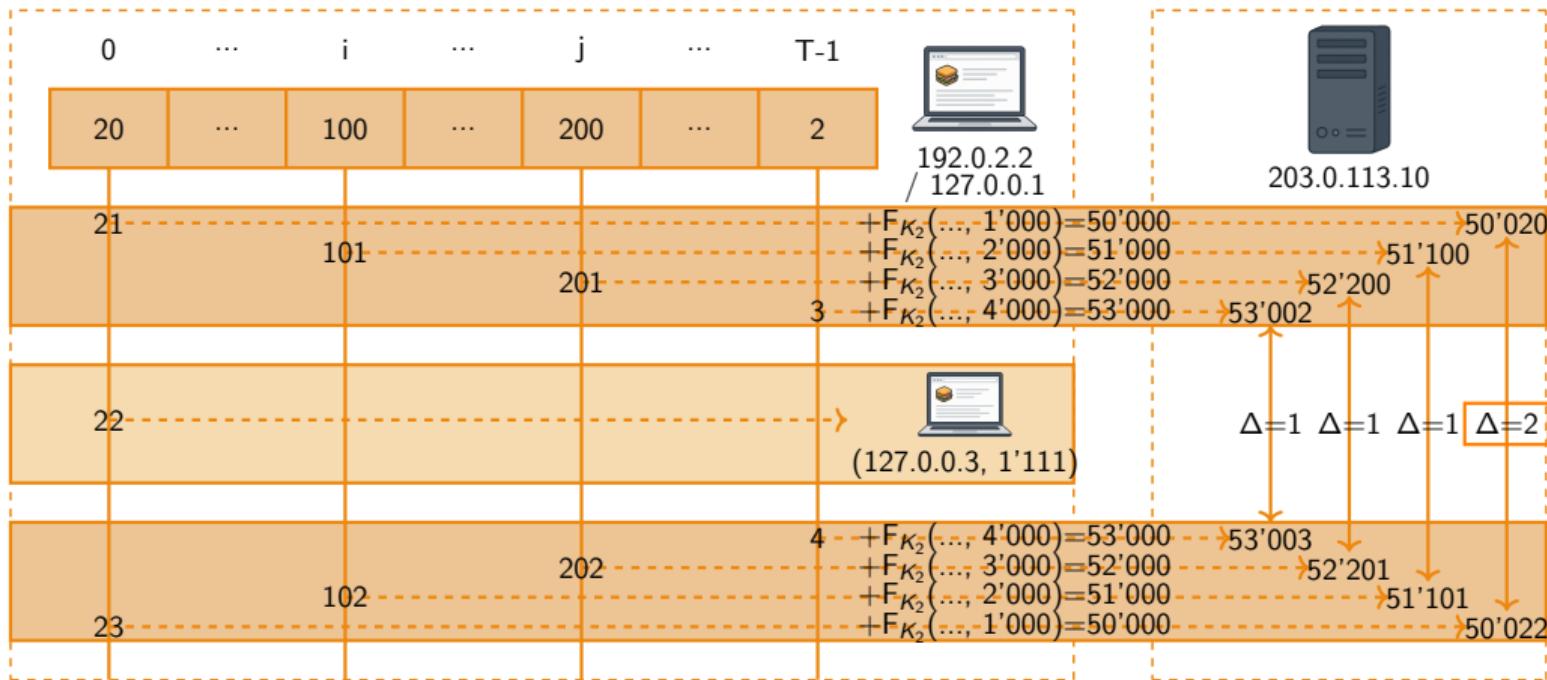
1 procedure PHASE2
2   C ← ∅; n ← 0; i ← 0
3   repeat
4     i ← i + 1
5     P ← GETSOURCEPORTS(s') // 1st burst
6     ATTEMPTCONNECTTCP({Li}) // 2nd burst
7     P' ← GETSOURCEPORTS(s') // 3rd burst
8     w ←  $\iota w (P'(w) - P(w) > \varepsilon)$ 
9     if DEFINED(Bw) then // A collision was found
10      C ← C ∪ {(Li, B[w])}
11      n ← n + 1
12    else
13      B[w] ← Li
14    until n ≥ n*[i] // This is equiv. to  $P_D^i(n) \leq p^*$ 
15    return (C, 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**, means "the unique w" attacker 3-tuple
- ▶ Only one loopback 3-tuple send repeatedly
 - ⇒ Only one attacker 3-tuple has significant increment
 - ⇒ Collision between attacker 3-tuple and loopback 3-tuple
- ▶ Fforeach table cell there's an attacker 3-tuple
 - ⇒ Always collision with attacker 3-tuple

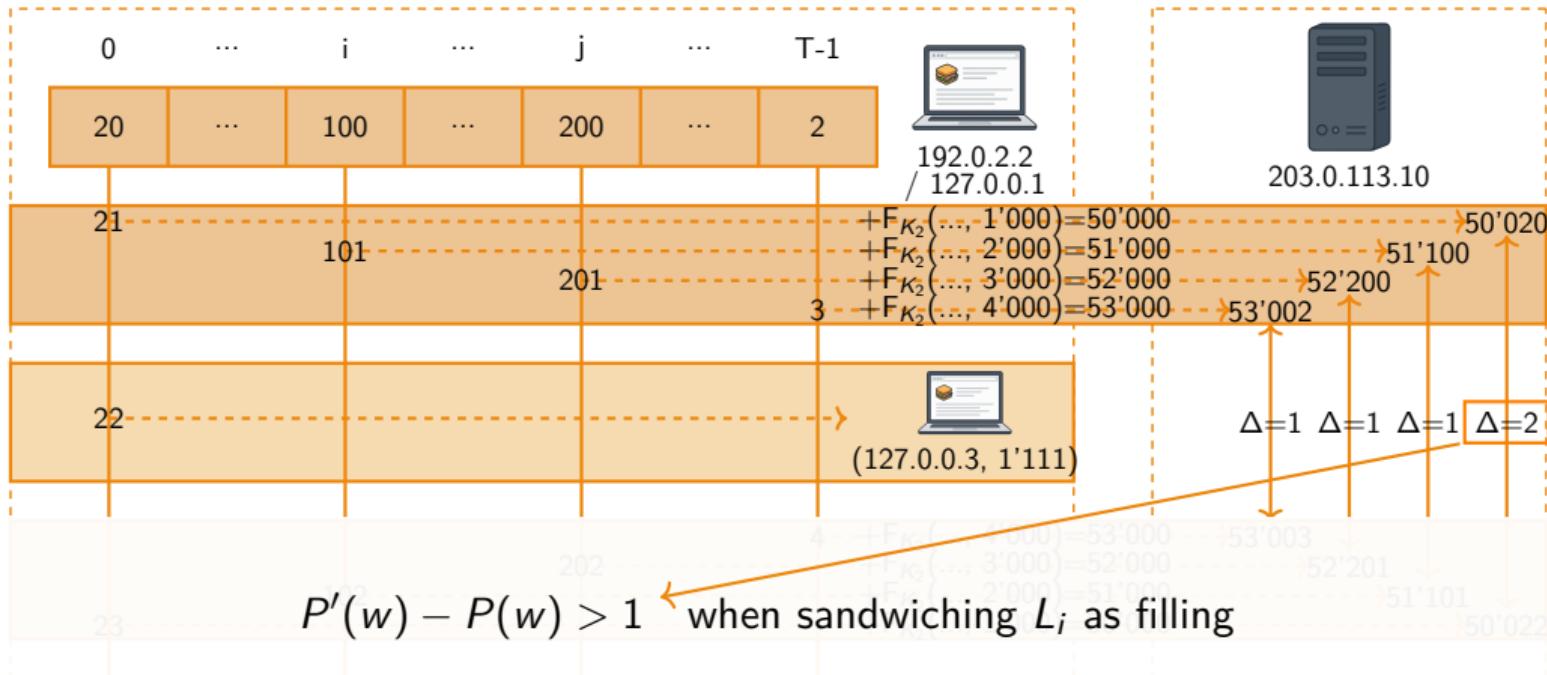
Phase 2

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



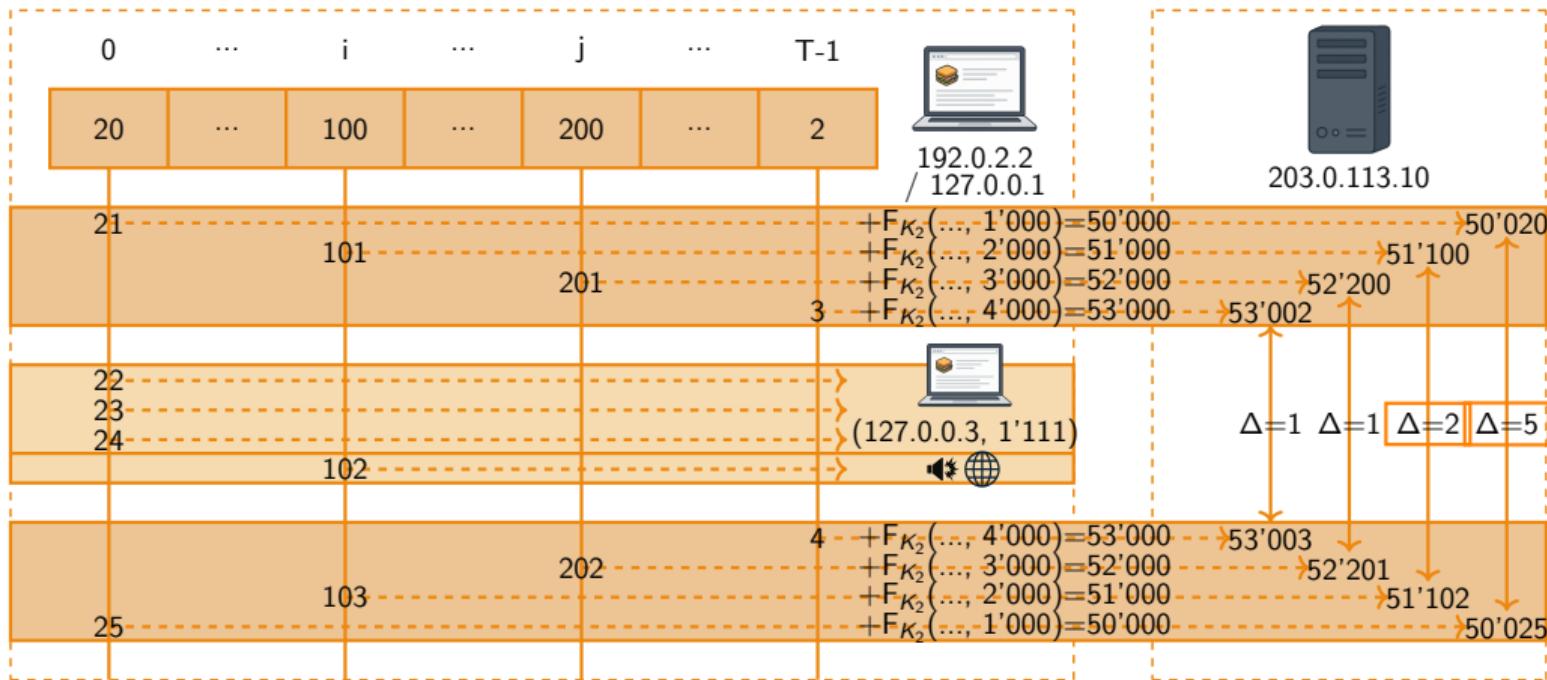
Phase 2

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



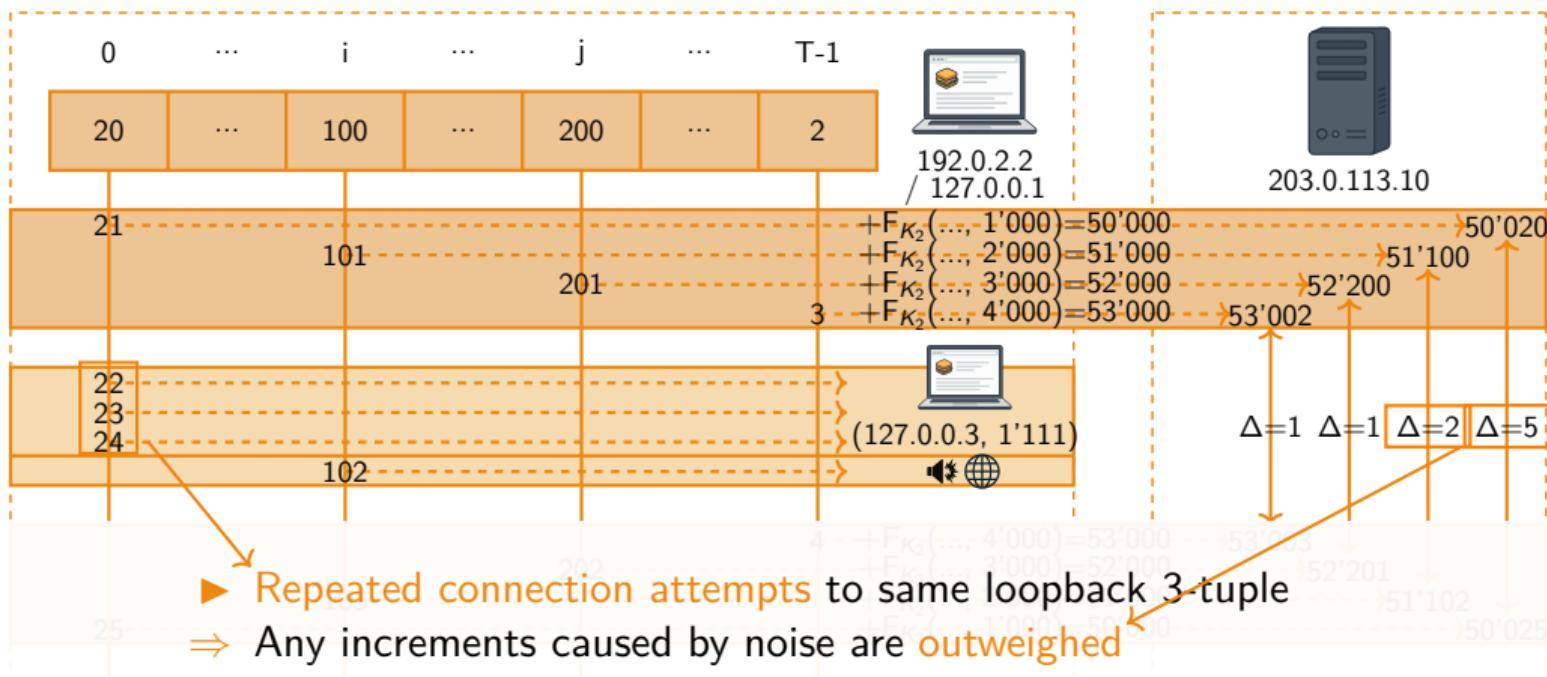
Phase 2

Influence of noise



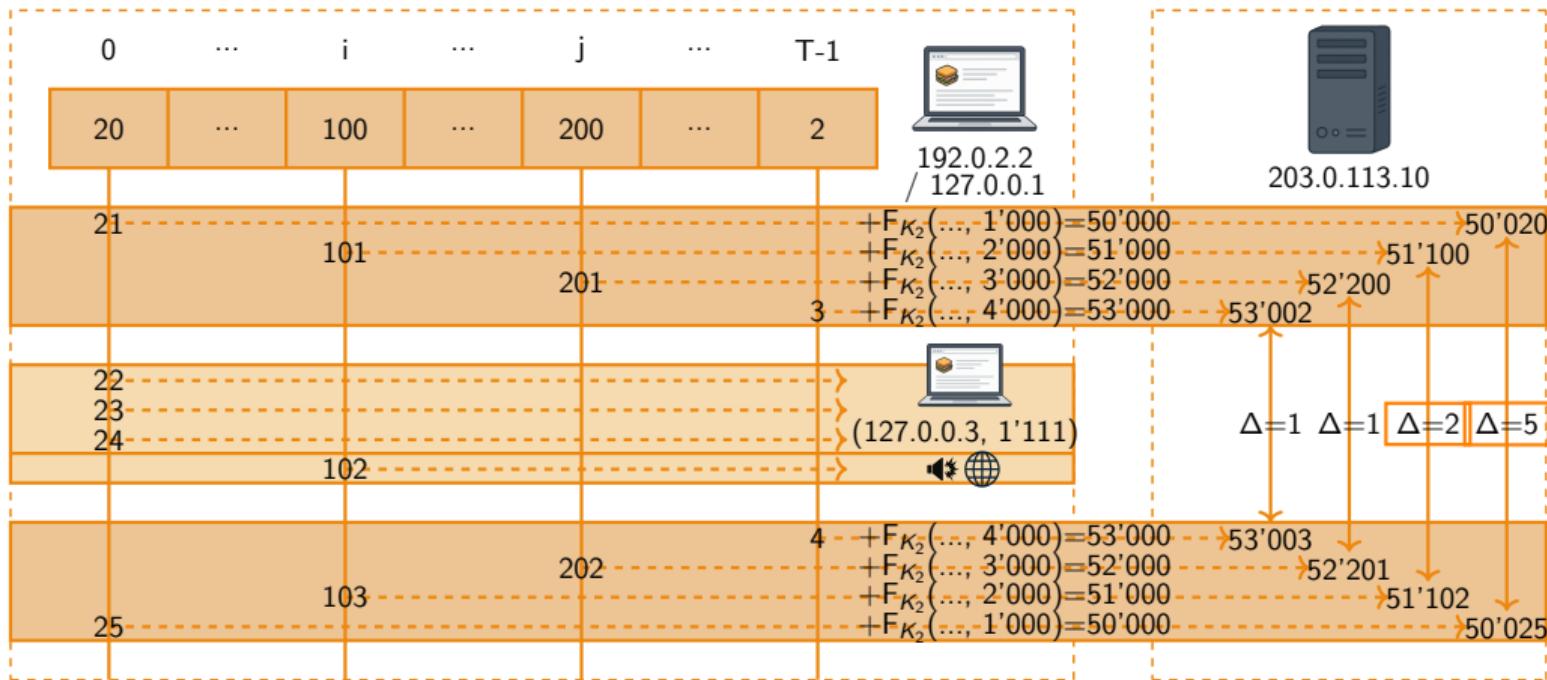
Phase 2

Influence of noise



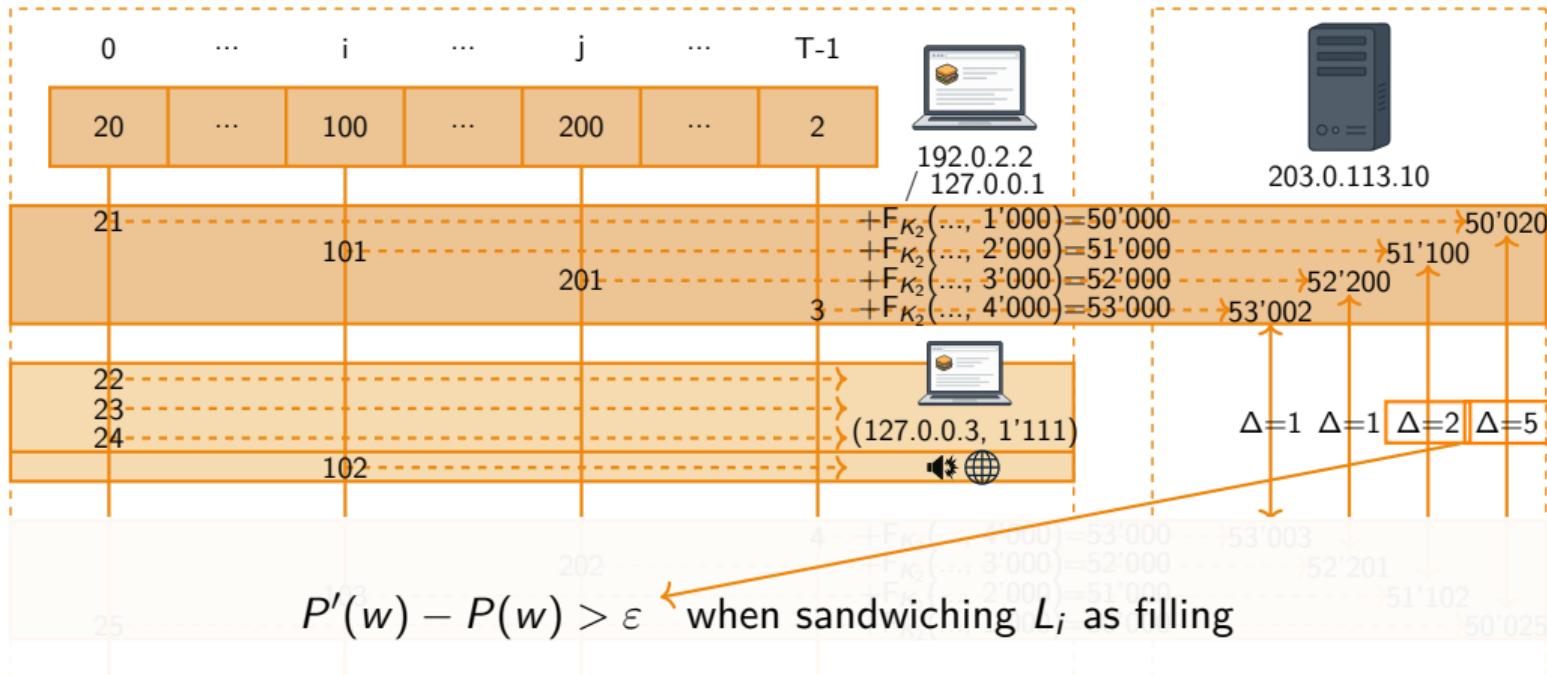
Phase 2

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



Phase 2

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



Phase 2

Pseudocode

Algorithm 10 FINDING A DEVICE ID (PHASE 2)

```

1 procedure PHASE2
2   C ← ∅; n ← 0; i ← 0
3   repeat
4     i ← i + 1
5     P ← GETSOURCEPORTS(s') // 1st burst
6     ATTEMPTCONNECTTCP({Li}) // 2nd burst
7     P' ← GETSOURCEPORTS(s') // 3rd burst
8     w ←  $\iota w(P'(w) - P(w) > \varepsilon)$ 
9     if DEFINED(Bw) then // A collision was found
10      C ← C ∪ {(Li, B[w])}
11      n ← n + 1
12    else
13      B[w] ← Li
14  until n ≥ n*[i] // This is equiv. to  $P_D^i(n) \leq p$ 
15  return (C, 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**, means "the unique w" attacker 3-tuple
- ▶ Only one loopback 3-tuple send repeatedly
 - ⇒ Only one attacker 3-tuple has significant increment
 - ⇒ Collision between attacker 3-tuple and loopback 3-tuple
- ▶ Fforeach table cell there's an attacker 3-tuple
 - ⇒ Always collision with attacker 3-tuple
- ▶ B[w]: **dictionary**, keys = unique attacker 3-tuples, values = first loopback 3-tuple
- ▶ {T₃, T₂, T₅} ⇒ {(T₃, T₂), (T₃, T₅)}


Phase 2

Pseudocode

Algorithm 11 FINDING A DEVICE ID (PHASE 2)

```

1 procedure PHASE2
2   C ← ∅; n ← 0; i ← 0
3   repeat
4     i ← i + 1
5     P ← GETSOURCEPORTS(s') // 1st burst
6     ATTEMPTCONNECTTCP({Li}) // 2nd burst
7     P' ← GETSOURCEPORTS(s') // 3rd burst
8     w ←  $\iota w (P'(w) - P(w) > \varepsilon)$ 
9     if DEFINED(Bw) then // A collision was found
10      C ← C ∪ {(Li, B[w])}
11      n ← n + 1
12    else
13      B[w] ← Li
14    until n ≥ n*[i] // This is equiv. to  $P_D^i(n) \leq p^*$ 
15    return (C, 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**, means "the unique w" attacker 3-tuple
- ▶ Only one loopback 3-tuple send repeatedly
 - ⇒ Only one attacker 3-tuple has significant increment
 - ⇒ Collision between attacker 3-tuple and loopback 3-tuple
- ▶ Fforeach table cell there's an attacker 3-tuple
 - ⇒ Always collision with attacker 3-tuple
- ▶ B[w]: **dictionary**, keys = unique attacker 3-tuples, values = first loopback 3-tuple
- ▶ {T₃, T₂, T₅} ⇒ {(T₃, T₂), (T₃, T₅)}

Phase 2

Termination condition

Probability that all non-first loopback 3-tuples go into the same buckets as in D

Combined probability of all first loopback 3-tuples in their buckets to match D's structure

Occupied (non-empty) buckets
Correction because indices start at 0

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

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

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

\Rightarrow thus $n < i$

1 $i-n$
0 ...
... $T-1$

T_1 T_l

$\{(T_1, T_7)\}$

i

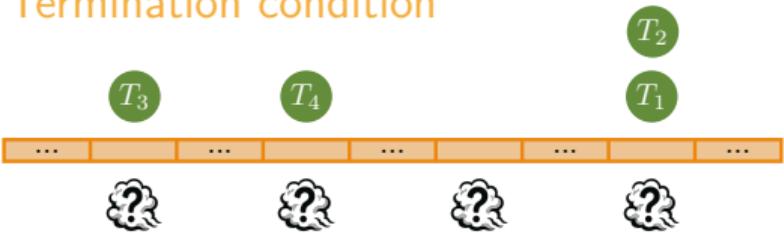
n

N

- ▷ 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:

◆ $\uparrow i \Rightarrow P_D^i(n) \downarrow \Rightarrow P_D^i(n) < p^*$ potentially closer to being satisfied

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

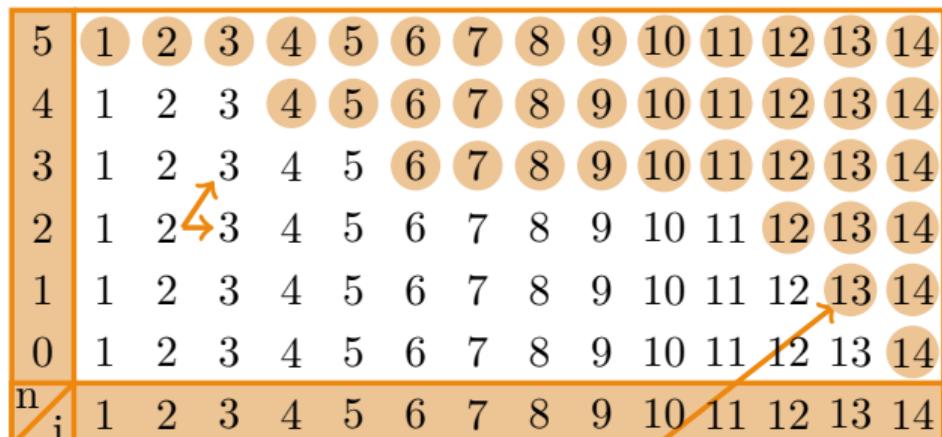
◆ $\uparrow n \Rightarrow P_D^i(n) \downarrow \Rightarrow P_D^i(n) < p^*$ potentially closer to being satisfied

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

\longrightarrow divide by large number $\longrightarrow P_D^i(n) \downarrow$

Phase 2

Termination condition



...



termination



N=100

0

...

15+1=16=T

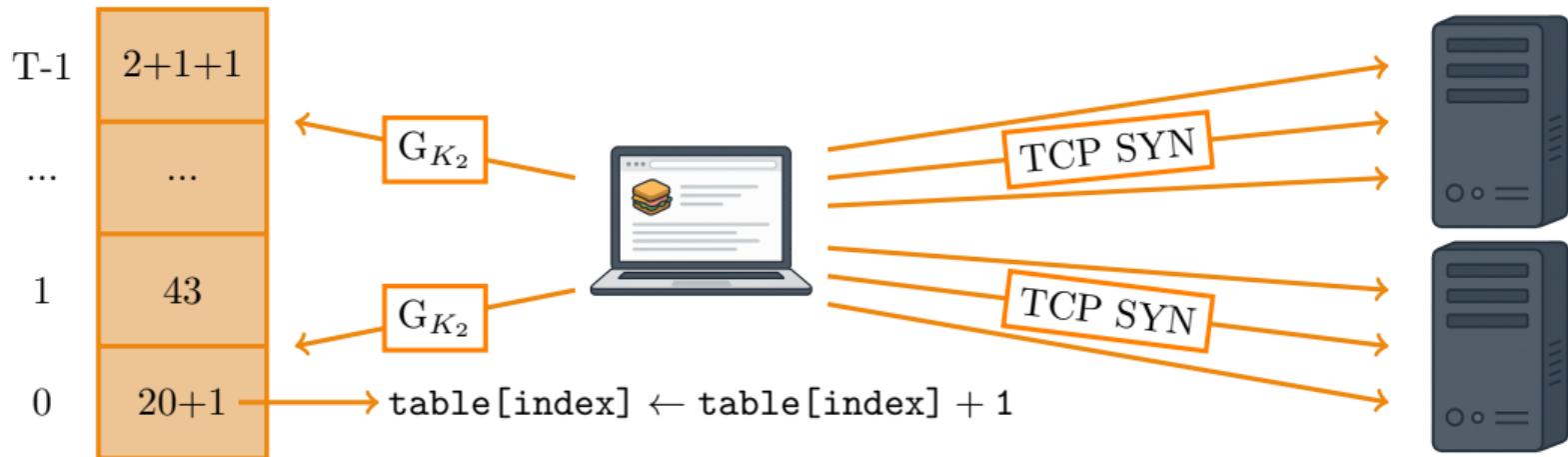
- ▶ Precompute table
 $n^*[i] = \min \{ n \mid P_D^i(n) < p^* \}$
- ▶ $n \geq n^*[i]$ replaces $P_D^i(n) < p^*$
- ⇒ Avoid repeated computation

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

Countermeasures

Countermeasures Overview

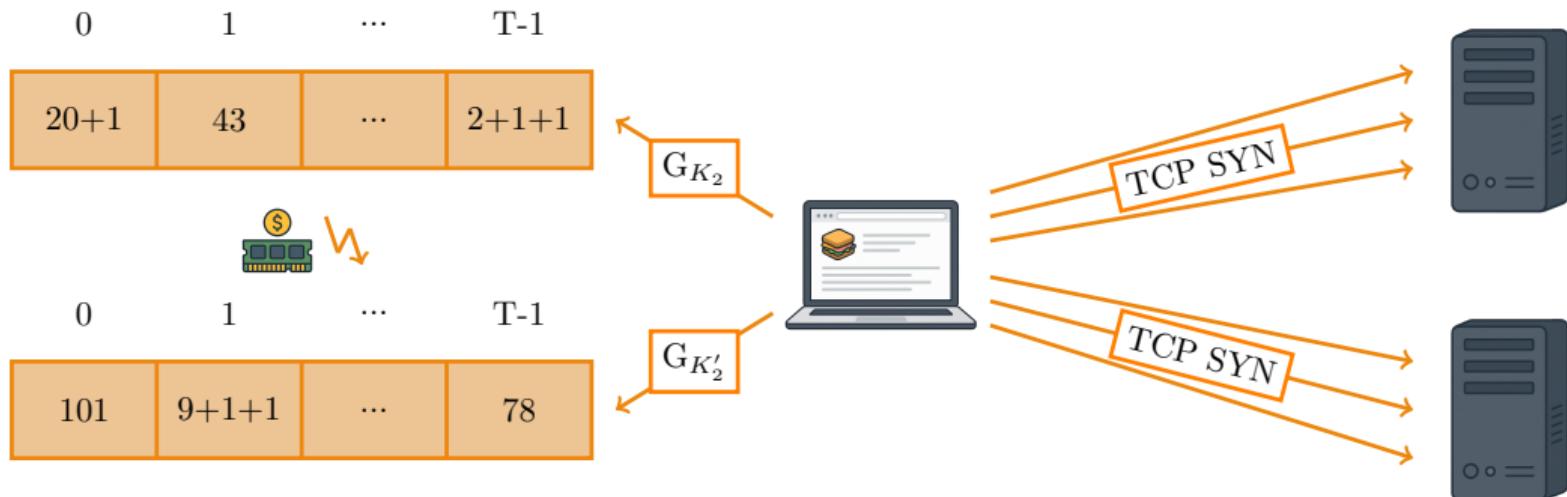
Root cause



- ▶ shared perturbation table \Rightarrow 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:



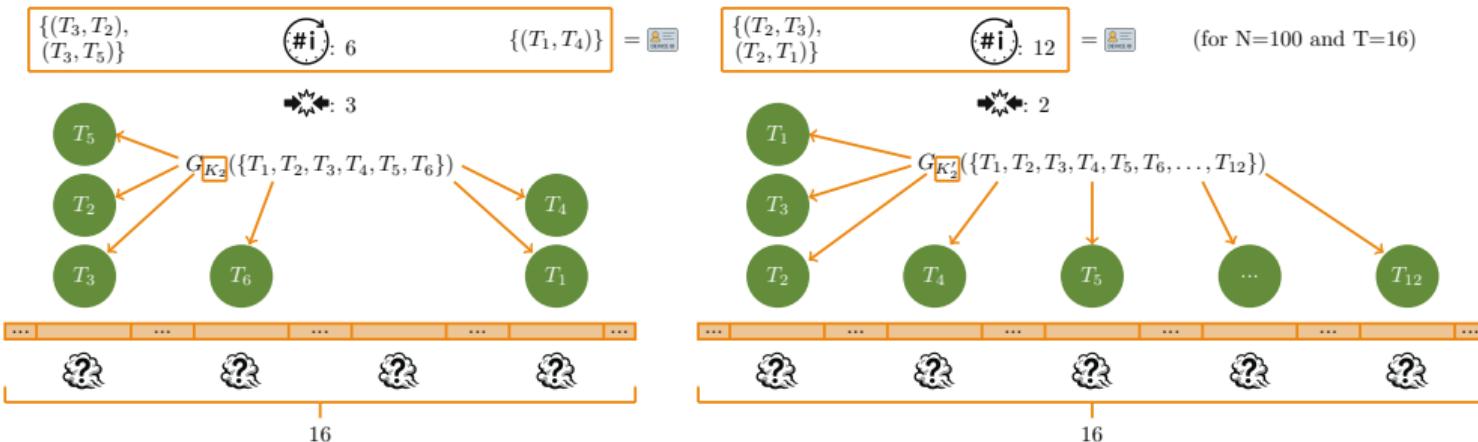
- ◆ limit time:



Countermeasures Overview

Implemented changes

- ▶ Periodic re-keying: Every 10 seconds

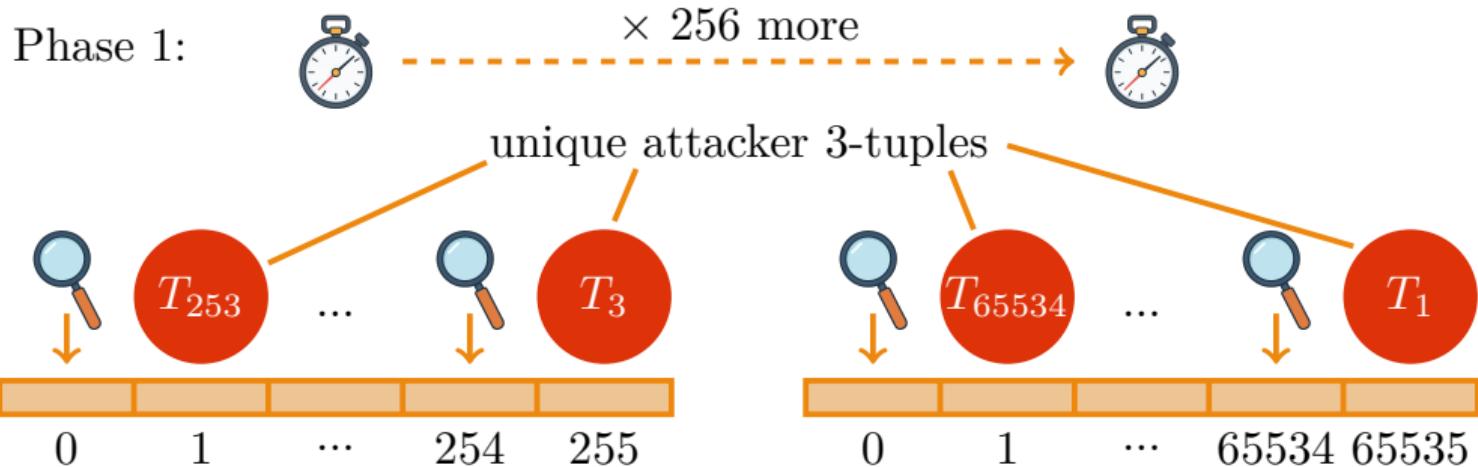


- ▶ Increase table size T : From 256 to 64 KiB
- ▶ Introduce more noise: Each increment randomly between 1 and 8

Countermeasures Overview

Implemented changes

- ▶ Periodic re-keying: Every 10 seconds
- ▶ Increase table size T : From 256 to 64 KiB



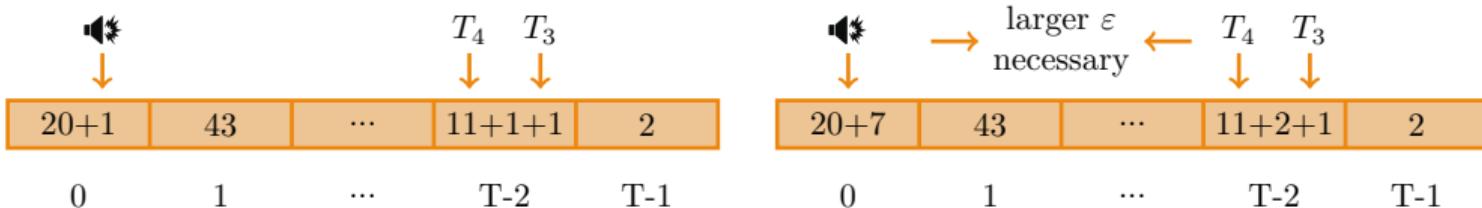
- ▶ Introduce more noise: Each increment randomly between 1 and 8

Countermeasures Overview

Implemented changes

- ▶ Periodic re-keying: Every 10 seconds
- ▶ Increase table size T : From 256 to 64 KiB
- ▶ Introduce more noise: Each increment randomly between 1 and 8

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



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>

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](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](https://en.wikipedia.org/w/index.php?title=Spectre_(security_vulnerability)&oldid=1295923886) (visited on 07/16/2025).

Literature II

- ▶ All other images have been generated with ChatGPT and DALL-E 3