Detection and Threat Prioritization of Pivoting Attacks in Large Networks

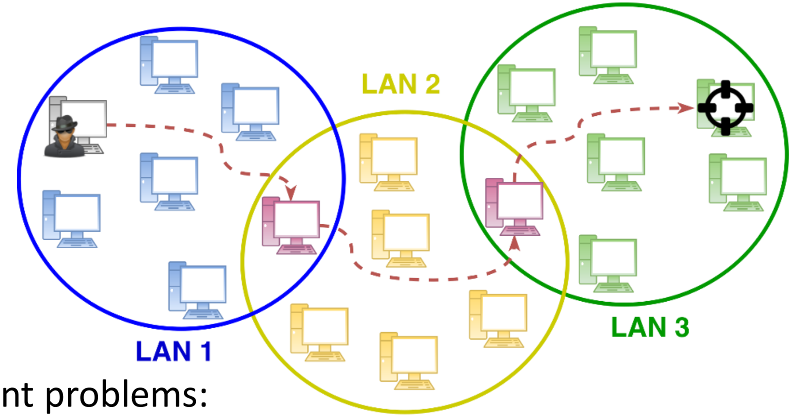
Giovanni Apruzzese, **Fabio Pierazzi**, **Michele Colajanni**, **Mirco Marchetti**
21st June, 2017
University of Modena and Reggio Emilia, Italy

Scenario

- Defending large enterprise systems is an extremely challenging task.
- Attackers want to control hosts with **higher privileges or more valuable data.**

→ Recent diffusion of *pivoting:*

- Operation Aurora (2010)
- Operation Night Dragon (2011)
- Black Energy malware (2015)
- MEDJACK (2016)
- Archimedes (2017)



- Countering pivoting poses significant problems:
  - Pivoting cannot be detected through signatures
  - False Positives
  - Evasion
  - Complexity

Related Work

- Limited literature
- Focuses on *prevention* instead of *detection*:
  - Game-theoretic models → **easily evaded**
  - Re-planning and re-structuring of the entire network → **unfeasible**
- Other detection approaches:
  - HIDS on every host → **unfeasible**
  - A-priori knowledge of adopted protocols → **easily evaded**

Our Proposal

- Original algorithm for pivoting **detection**
  - Based on network **flows**
    - Easy to collect, store and fast to analyze
  - No a-priori knowledge required

- Algorithm for threat **prioritization** of pivoting attacks
  - Ranks the detected pivoting activities

- **Feasible** for large networks

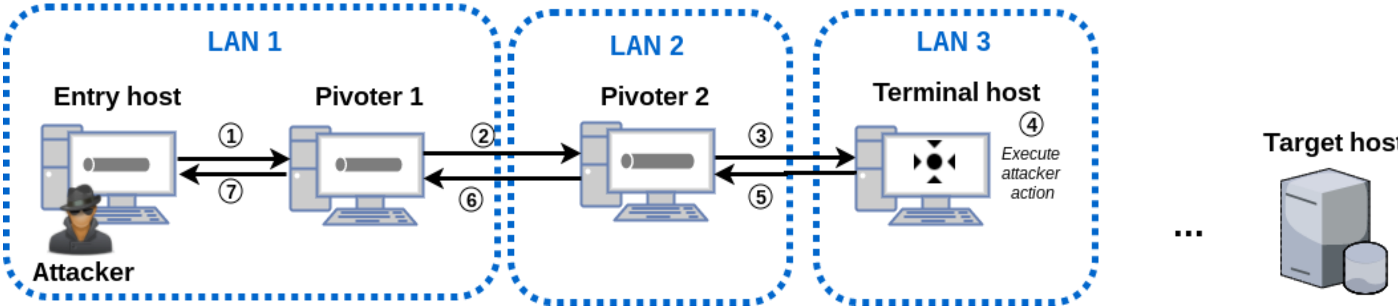
Pivoting Description

- Pivoting: any action in which a *command propagation tunnel* is created among three or more hosts

- Pivoting **activities** are not necessarily malicious

- Pivoting **attacks** consist of three phases:
  - Reconnaissance
  - Compromise
  - Command Propagation ← **Our focus**

Pivoting Example



Definitions

- *(network) Flow:*
  - Aggregation of packets from a source host to a destination host

$$f = (src;\ dst;\ p_{src};\ p_{dst};\ b_{in};\ b_{out};\ d;\ t)$$

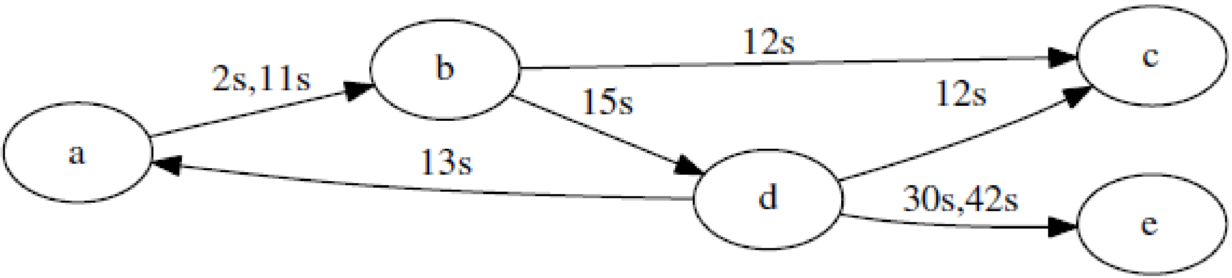
- *Flow-sequence:*
  - Ordered set of flows where consecutive flows are:
    - Chronologically ordered
    - Separated by at most $\boldsymbol{\varepsilon_{max}}$ time units
    - Adjacent
    - Not cyclical

Example of *flow-sequence* ($\varepsilon_{max} = 20s$):
(a,b,10s),(b,d,15s),(d,e,30s)

- *Pivoting path:*
  - A *pivoting path* is an ordered set of hosts for which at least one flow-sequence exists

From flow-sequence:
(a,b,10s),(b,d,15s),(d,e,30s)
...we can derive the *pivoting path*:
(a,b,d,e)

Example



If $\varepsilon_{max} = 27s$

Path	Length	Flow sequences
a,b,d	2	(a,b,2s),(b,d,15s) (a,b,11s),(b,d,15s)
a,b,c	2	(a,b,2s),(b,c,12s) (a,b,11s),(b,c,12s)
b,d,e	2	(b,d,15s),(d,e,30s) (b,d,15s),(d,e,42s)
a,b,d,e	3	(a,b,11s),(b,d,15s),(d,e,30s) (a,b,11s),(b,d,15s),(d,e,42s) (a,b,2s),(b,d,15s),(d,e,30s) (a,b,2s),(b,d,15s),(d,e,42s)

If $\varepsilon_{max} = 5s$

Path	Length	Flow sequences
a,b,d	2	(a,b,11s),(b,d,15s)
a,b,c	2	(a,b,11s),(b,c,12s)

Pivoting Detection Algorithm – 1

- **Input:**
  - All the **network flows** that occur within a time-window $\boldsymbol{W}$
  - The maximum propagation delay $\boldsymbol{\varepsilon_{max}}$
  - The maximum flow-sequence length $\boldsymbol{L_{max}}$

- **Output:**
  - List of all the **flow-sequences** occurring within the time-window W

Pivoting Detection Algorithm – 2

1. Read all the input flows and store them in F

2. Iterate over F:
   - Build flow-sequences of length-1 and store them in P

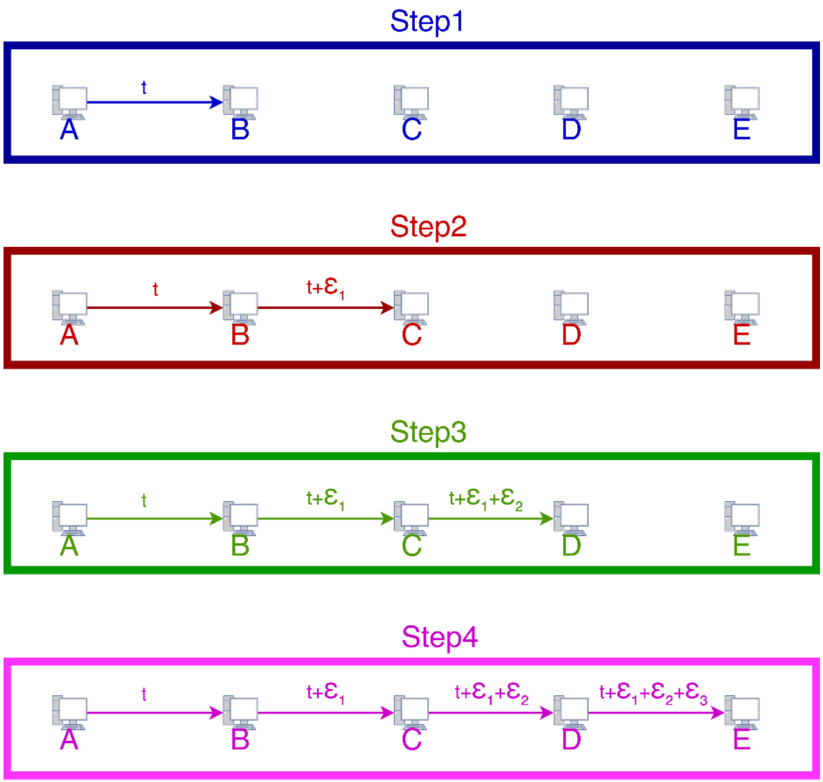
Flow-sequences of length-1 are the same as flows

3. For $i = 1$ to L_{max}:
   - For every flow-sequence k of length-i in P, check if you can extend k to a flow-sequence k' of length-$(i + 1)$ with any flow in F
     - If you can, then add k' at the end of P
     - Keep checking for all extensions of k of length-$(i + 1)$
   - If you cannot find any flow-sequence of length-$(i + 1)$, **stop**

4. Return P

Pivoting Detection Algorithm – 3

$$\varepsilon_i \leq \varepsilon_{max}, \forall i$$



Threat Prioritization Algorithm

- Reminder: pivoting activities are not necessarily malicious
- Need to discriminate between “benign” and “malicious” pivoting
- **Solution**: Rank the detected pivoting activities on the basis of threatening characteristics displayed
- Characteristics considered by the algorithm:
  - Novelty
  - Reconnaissance Activities
  - Uncommon Ports
  - LANs involved
  - Anomalous Data Transfers

Experimental Evaluation – Testbed

- Collected the network flows of a large real organization (over 90M flows)

- Assessed the capabilities of our proposals to:
  - **Detect** benign and malicious pivoting activities
  - **Prioritize** malicious pivoting activities
  - Perform the analyses in **feasible times** for large organizations

- Malicious pivoting activities injected in the regular traffic

Experimental Evaluation – Results

- Execution of the Detection algorithm on the injected real dataset with $\varepsilon_{max} = 1s$:
  - All injected attacks have been detected
  - Also the benign pivoting activities have been detected (≅1800 flow-sequences)

- Results of the Prioritization algorithm:

	average rank	**standard deviation**
Attack Class 1 (ω)	1.38	1.32
Attack Class 1 (β)	1.17	0.72
Attack Class 2 (ω)	2.01	1.18
Attack Class 2 (β)	1.55	1.04
Attack Class 3 (ω)	1.00	0.00
Attack Class 3 (β)	1.00	0.00
Attack Class 4 (ω)	1.13	0.51
Attack Class 4 (β)	1.14	0.68
Attack Class 5 (ω)	1.15	0.83
Attack Class 5 (β)	1.14	0.78

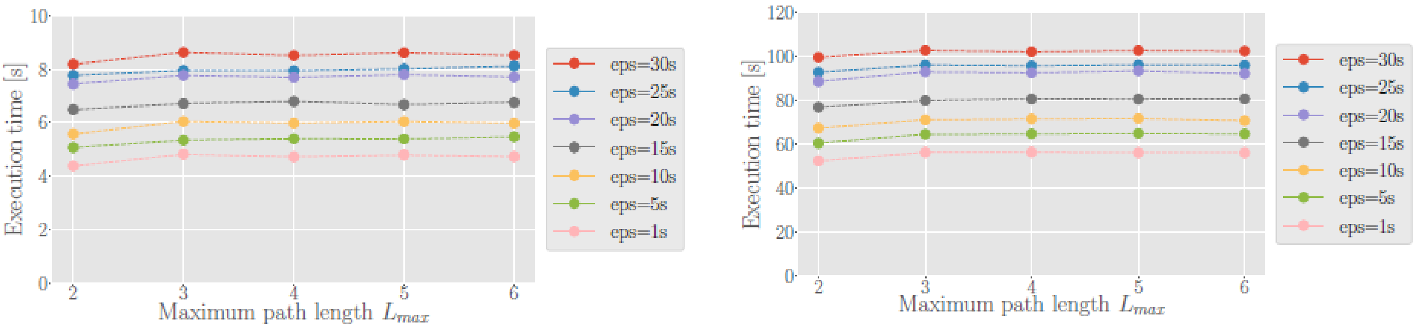
Experimental Evaluation – Evasion

- Attackers may try to elude detection by increasing the command propagation delay
- Increasing ε_{max} also increases the number of false positives
  → Priotization algorithm can help in these situations
- Results of the algorithms on the (new) injected dataset:

	$1s$	$5s$	$10s$	$15s$	$20s$	$25s$	$30s$
Attack Class 1 (ω)	✗	✓ 1.48 (1.67)	✓ 1.55 (1.84)	✓ 1.48 (1.58)	✓ 1.62 (1.91)	✓ 1.65 (1.93)	✓ 1.69 (1.98)
Attack Class 1 (β)	✗	✓ 1.21 (1.09)	✓ 1.21 (1.12)	✓ 1.21 (1.10)	✓ 1.21 (0.92)	✓ 1.21 (0.93)	✓ 1.21 (0.99)
Attack Class 2 (ω)	✗	✓ 2.11 (1.23)	✓ 2.24 (1.26)	✓ 2.27 (1.46)	✓ 2.52 (1.57)	✓ 2.65 (1.66)	✓ 2.80 (1.94)
Attack Class 2 (β)	✗	✓ 1.61 (1.11)	✓ 1.72 (1.19)	✓ 1.81 (1.34)	✓ 2.04 (1.29)	✓ 2.09 (1.54)	✓ 2.21 (1.65)
Attack Class 3 (ω)	✗	✗	✓ 1.00 (0.00)	✓ 1.00 (0.00)	✓ 1.00 (0.00)	✓ 1.00 (0.00)	✓ 1.00 (0.00)
Attack Class 3 (β)	✗	✗	✓ 1.00 (0.00)	✓ 1.00 (0.00)	✓ 1.00 (0.00)	✓ 1.00 (0.00)	✓ 1.00 (0.00)
Attack Class 4 (ω)	✗	✗	✓ 1.26 (0.86)	✓ 1.26 (1.14)	✓ 1.21 (1.31)	✓ 1.21 (1.00)	✓ 1.21 (1.63)
Attack Class 4 (β)	✗	✗	✓ 1.21 (0.75)	✓ 1.21 (1.06)	✓ 1.17 (1.23)	✓ 1.17 (1.32)	✓ 1.17 (1.37)
Attack Class 5 (ω)	✗	✗	✗	✓ 1.26 (1.16)	✓ 1.21 (1.44)	✓ 1.21 (1.56)	✓ 1.21 (1.86)
Attack Class 5 (β)	✗	✗	✗	✓ 1.21 (1.15)	✓ 1.17 (1.28)	✓ 1.17 (1.29)	✓ 1.17 (1.54)

Experimental Evaluation – Execution times

- Execution times of the Detection Algorithm on the entire injected dataset with different input values of ε_{max}, L_{max} and W:



(a) Analysis of 1 hour of data ($\sigma_{max} = 4.8$s).

(b) Analysis of 12 hours of data ($\sigma_{max} = 17.8$s).

- Analyses performed on an Intel Xeon E5-2609 v2 CPU, 128GB RAM.

Conclusions

- Pivoting is an **increasingly adopted** technique by attackers.

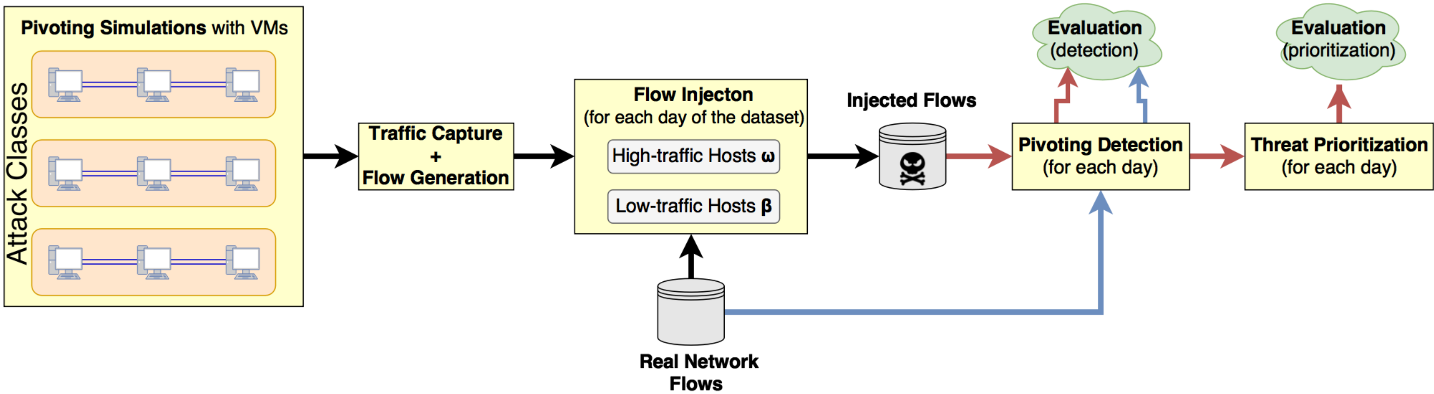
- Proposed novel algorithms for:
  - **Detection** of pivoting activities
  - **Threat Prioritization** of pivoting attacks

- Extensive analyses of the proposed solutions confirmed their:
  - **Effectiveness**
  - **Efficency**
  - **Applicability** to practical contexts

Detection and Threat Prioritization of Pivoting Attacks in Large Networks

Giovanni Apruzzese, **Fabio Pierazzi**, **Michele Colajanni**, **Mirco Marchetti**
21st June, 2017
University of Modena and Reggio Emilia, Italy

Experimental Evaluation – Workflow



Pivoting Attack Classes.

	Vector	Len	Recon	LANs	Data
Attack Class 1	SSH	2	✓	2	10 MB
Attack Class 2	SSH	2	✗	2	30 MB
Attack Class 3	Metasploit	4	✓	5	100 MB
Attack Class 4	Metasploit	3	✗	4	< 1 MB
Attack Class 5	Metasploit	4	✗	1	5 MB

Propagation delays for pivoting Attack Classes.

	Delay
Attack Class 1	$2s$
Attack Class 2	$4s$
Attack Class 3	$8s$
Attack Class 4	$10s$
Attack Class 5	$15s$

Pivoting Detection Algorithm – full

Algorithm 1: Algorithm for pivoting detection.

Input: List of m temporal edges corresponding to time window W ($Flows$), maximum propagation delay ε, minimum incoming and outgoing bytes B_{in} and B_{out}, maximum flow duration δ, maximum pivoting path length L_{max}

Output: List of *pivoting flow sequences* of length ≥ 2 (corresponding to *pivoting paths*)

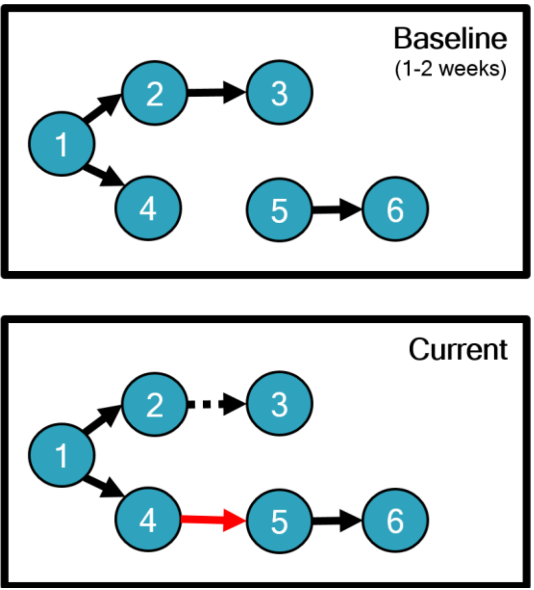
// Initialization
$PivotingSequences \leftarrow$ emptyList();
$CandidateFlows \leftarrow$ emptyList();
for *flow* f *in* $Flows$ **do**
    if $(f.d \geq \delta)$ **and** $(f.b_{in} \geq B_{in}$ **and** $f.b_{out} \geq B_{out})$ **then**
        Insert flow f in $PivotingSequences$;
        Insert flow f in $CandidateFlows$;
// Look for possible pivoting flow sequences of length ≥ 2
for *flow sequence* $\mathcal{F}$ *in* $PivotingSequences$ **do**
    if *length*($\mathcal{F}$) $\geq L_{max}$ **then**
        break;
    $FoundSequences \leftarrow ExtendPivotingSequence(\mathcal{F}, CandidateFlows, \varepsilon)$
    Include $FoundSequences$ in $PivotingSequences$;
return List of elements in $PivotingSequences$ with length ≥ 2;
// Function to find flow sequences of length $(\ell + 1)$ given a sequence $\mathcal{F}$ of length ℓ
Function *ExtendPivotingSequence*($\mathcal{F}$,$CandidateFlows$,ε)
    $FoundSequences \leftarrow$ emptyList();
    $h_{\mathcal{F}} \leftarrow$ last host in pivoting flow sequence $\mathcal{F}$
    $t_{\mathcal{F}} \leftarrow$ lastest timestamp of $\mathcal{F}$
    $FlowsWithinDelay \leftarrow$ BinarySearch($CandidateFlows[t_{\mathcal{F}} : t_{\mathcal{F}} + \varepsilon]$)
    for *flow* f *in* $FlowsWithinDelay$ **do**
        if (($f.src$ *equal to* $h_{\mathcal{F}}$) **and** ($f.dst$ *not in sequence* $\mathcal{F}$)) **then**
            $NewSequence \leftarrow$ (sequence $\mathcal{F}$ with flow f);
            Insert $NewSequence$ in $FoundSequences$;
    return $FoundSequences$;

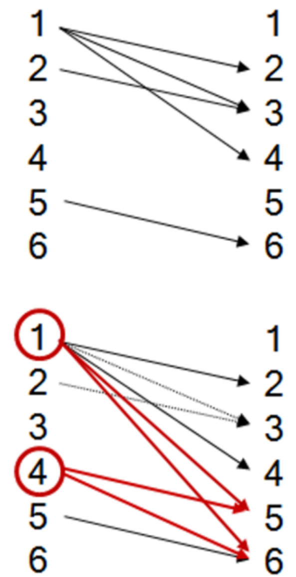
Backstory…

- Our original goal was to focus on **Lateral Movement** as a whole, not on pivoting.

This could be achieved with a ***reachability graph***

Baseline vs. Current





Idea:
sudden increase in **reachable destinations** ≅ **malicious activity**

Problem

- The *paths* from which the desired reachability graph is built have the following definition:
  - Ordered set of $L > 2$ unique hosts where each host $i \leq L$ received a communication from host $(i-1)$ after that host $(i-1)$ received a communication from $(i-2)$

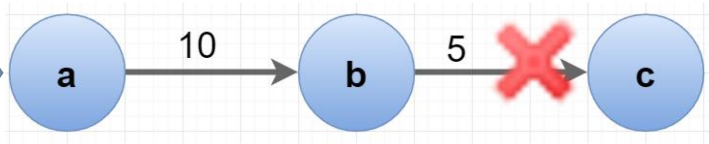
- How to compute such a reachability graph:
  - Starting from network flows
  - Fast enough to support **online analyses** in a large enterprise network

Hint: we could obtain a reachability graph of one day by providing an $\varepsilon_{max} = 24\text{h}$ to the pivoting detection algorithm...
→ **This takes hours to complete!**

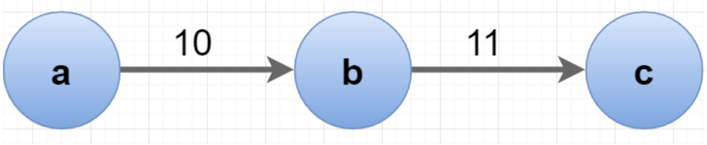
Solutions…? – 1

- **IDEA:** reduce computation time by decreasing the amount of reads on the input flows
- **First attempt:** keep only the *first* flow between each pair of hosts.
  - Create paths by joining adjacent flows, in which the timestamp of the latter is higher than the timestamp of the former
  - After adding a new host, set the timestamp of this host to the highest value of the timestamp of all hosts of the path
- **Problem:** false negatives: some paths are not detected

Assume the following flows:
(b,c,5), (a,b, 10), (b,c,11), (a,c,20)



This is what the attacker did:



Solutions…? – 2.1

- **IDEA**: reduce computation time by decreasing the amount of reads on the input flows

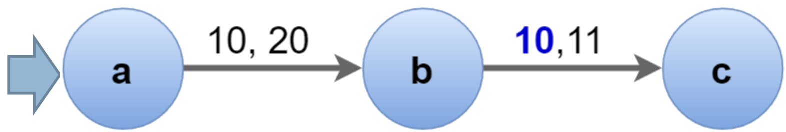
- **Second attempt**: keep only the *first* and *last* flows between each pair of hosts.
  - Create paths by joining adjacent flows, in which the *last* timestamp of the latter is <u>higher</u> than the *first* timestamp of the former.
  - After adding a new host, set the *first* timestamp of this host to the highest value of the *first* timestamp of all hosts of the path

- This solution solves the previous situation:

Assume the following flows:
(b,c,5), (a,b,10), (b,c,11), (a,b,20)

↓

(a,b,10,20), (b,c,5,11)

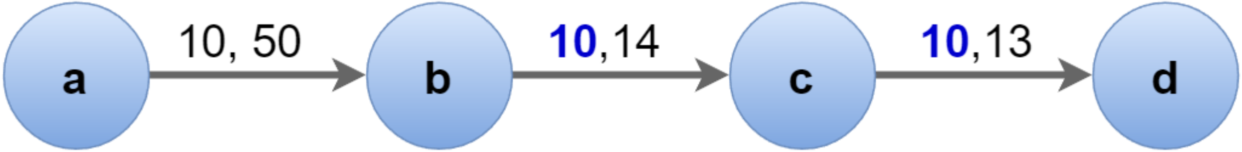


Solutions…? – 2.2

- **Problem:** false positives: some detected paths are not actually paths

Assume the following flows:
(c,d,1), (b,c,5), (a,b,10), (c,d,13), (b,c,14), (a,b,50)

(a,b,10,50), (b,c,5,14), (c,d,1,13)



However, in reality no path **(a,b,c,d)** could possibly exist from those flows!
It would exist if there were an additional flow, e.g.:
(b,c,11)

Solutions…?

- The second solution still requires validation of all the detected paths, to check if they actually exist and are not false positives.
  → **expensive**

- However, the second solution always works if the path has only 3 hosts.

- Focusing on pivoting introduced the concept expressed by $\boldsymbol{\varepsilon}$, which dramatically reduced computation times due to a powerful filtering criteria.