Detection and Threat Prioritization of Pivoting Attacks in Large Networks

Giovanni Apruzzese, Fabio Pierazzi, Michele Colajanni, Mirco Marchetti

Abstract—Several advanced cyber attacks adopt the technique of “pivoting” through which attackers create a command propagation tunnel through two or more hosts in order to reach their final target. Identifying such malicious activities is one of the most tough research problems because of several challenges: command propagation is a rare event that cannot be detected through signatures, the huge amount of internal communications facilitates attackers evasion, timely pivoting discovery is computationally demanding. This paper describes the first pivoting detection algorithm that is based on network flows analyses, does not rely on any a-priori assumption on protocols and hosts, and leverages an original problem formalization in terms of temporal graph analytics. We also introduce a prioritization algorithm that ranks the detected paths on the basis of a threat score thus letting security analysts investigate just the most suspicious pivoting tunnels. Feasibility and effectiveness of our proposal are assessed through a broad set of experiments that demonstrate its higher accuracy and performance against related algorithms.

Keywords—pivoting, graph, island-hopping, lateral movement

1 Introduction

Defending large enterprise systems is an increasingly challenging task. Modern attacks may combine social engineering strategies with malware to exploit software vulnerabilities, allowing attackers to find their ways into the network. Advanced attacks tend to last for long periods [1], but most malicious activities are masked by the sheer amount of traffic flowing in and out enterprise networks and by the large volume of daily alarms overwhelming security analysts. Attackers typically begin by compromising any vulnerable internal host and then try to reach the most valuable targets by moving host-to-host laterally and deeper into the enterprise network. To this purpose, attackers are increasingly adopting the so called *pivoting* technique [2] in which a command propagation tunnel is created through one or more compromised internal host called *pivoters*. Unaware pivoters propagate malicious commands to the last host of the pivoting chain and return the results to the initial host. The terminal host can either represent the final target or be used by attackers to further increase the length of the pivoting tunnel. Several examples of pivoting can be found in the *Operation Aurora* [3], in the *Operation Night Dragon* [4], in the *BlackEnergy* [5] malware that in December 2015 compromised the Ukrainian power grid, in the *MEDJACK* [6] attack in 2016 where attackers stole health data by using medical devices as pivoters. Among the most recent cases at the time of writing, we can cite the *Archimedes* tool [7] that leverages pivoting to reach the LAN of target hosts, passively gathers their Web traffic, and injects forged Web pages with the goal of compromising hosts or stealing credentials. All these examples demonstrate that pivoting is an emerging and challenging research problem. Pivoting attacks are facilitated and the complexity of their detection is aggravated by various factors: the large number of heterogeneous host activities and the hundreds of thousands of communications occurring every day in any medium-large enterprise, which are difficult to analyze and may ease attackers evasion; the risk of false alarms because some activities detected as pivoting may be benign, such as SSH tunnels created by network administrators. Identifying malicious pivoting actions would require to keep stateful traces of all consecutive communications between internal hosts over long time periods but this approach is not viable in terms of computational and memory costs.

To address these issues, we propose the first algorithm for detection and threat prioritization of pivoting that analyzes internal network flows and does not rely on any a-priori knowledge about the adopted protocols and compromised hosts, which are instead needed by related solutions making them impractical for real contexts. For example, the algorithms in [8], [9] consider only a specific protocol (e.g., SSH brute-forcing), while another paper [10] considers only hosts that generate some IDS security alerts. Approaches in [11], [12] may work for detecting pivoting only if aggressive scan activities are performed over hundreds of hosts. We initially propose an original formalization of the pivoting detection problem into the temporal graph analytics domain, and then we present a pivoting detection algorithm that identifies pivoting tunnels through efficient network flow analyses that do not require any a-priori assumption

• *The authors are with the Department of Engineering “Enzo Ferrari”, University of Modena and Reggio Emilia, Italy. Email: {giovanni.apruzzese, fabio.pierazzi, michele.colajanni, mirco.marchetti}@unimore.it*

about involved protocols and hosts. As some paths may correspond to benign activities, we add an original threat prioritization algorithm that ranks the detected pivoting activities on the basis of a maliciousness score. The execution times of the pivoting detection algorithm are evaluated on extensive network traffic data of a large organization. The efficacy of the pivoting detection and threat prioritization algorithms is assessed by injecting realistic pivoting attacks and by comparing our results against those of related algorithms [8], [11].

The remainder of the paper is structured as follows. Section 2 compares this paper against related literature. Section 3 describes the details of the pivoting technique for cyber attacks. Section 4 presents the pivoting detection as a temporal graph analytics problem and proposes an original algorithm for path discovery. Section 5 proposes some key indicators for threat prioritization. Section 6 presents performance and efficacy results through an extensive campaign of experiments. Finally, Section 7 discusses conclusions and future work.

2 Related work

To the best of our knowledge, this paper presents the first algorithms for detection and threat prioritization of malicious pivoting activities: our proposal relies on the analysis of network flows, does not make assumptions about involved protocols and hosts, and is based on an original formulation of the pivoting detection problem in the temporal graph analytics domain.

A research area broadly related to pivoting studies malware propagation, whose two representative examples are worm detection [12] and botnet [13] discovery. These works define statistical models of normality for communications among internal hosts, where anomalies represent possible bot/worm propagations. Other papers in this area analyze network flows (e.g., [14], [15], [16]) as we do, but their solutions work only under two major assumptions that are not valid for pivoting: they require from tens to hundreds of hosts involved in malicious activities; they assume that an aggressive propagation model is adopted by the bot/worm. Unlike these scenarios, pivoting activities typically involve few internal hosts out of the thousands and more of a medium-large organization. Moreover, several research papers based on network flows analyses [17] aim to detect quite different attacks, such as data exfiltration [18], DDoS [19], port-scanning [20]. Other works focus on attack detection in Wireless Sensor Networks that does not include the identification of command propagation tunnels which is the core of our proposal. For example, the authors in [21] propose a protocol for detecting selective forwarding attacks, and the researchers in [22] present a framework for detecting ongoing attacks (such as DoS and ARP replay) through usage control and chance discovery.

Research on pivoting is still at the beginning. Many articles (e.g., [3], [4], [5], [6]) have a useful descriptive approach of popular pivoting-based attacks, but they do not propose any novel detection algorithm. Other works (e.g., [23], [24]) aim to train security experts on pivoting-related plans and attacks but they do not consider countermeasures. Some papers focus on prevention of pivoting-related activities, without proposing any detection approach. For example, Chapman et al. [25] simulate pivoting-based attacks through a game-theoretic framework and propose some best practices based on their observations. Johnson et al. [26] present an original graph metric that quantifies whether granting a certain privilege to an employee may increase chances of pivoting in Windows domains.

Other research efforts related to pivoting detection make strong assumptions that are not applicable to real contexts. Some papers [8], [9] are oriented to detect pivoting-related activities involving the SSH protocol and brute-force password guessing, while we do not make any assumption on host and protocol types. The heuristics proposed in [10] are valid to detect one-step pivoting activities, but just those related to security alerts issued by a signature-based network intrusion detection system. We can detect pivoting paths of any length and do not leverage alerts of other defensive systems, although integrations are feasible. Some works focus on *lateral movement* [27] activities where attackers move within the network to get closer to their final target. For example, the paper in [28] proposes a game-theoretic framework that models attackers behavior and simulates network responses aiming to prevent attackers from reaching valuable hosts. The problem is that expert attackers can easily evade similar defensive schemes by acting differently from the expected model. The detection algorithm proposed by Fawaz et al. [29] requires analyses of huge amounts of host-based logs, which are hard to collect in large organization and may be easily altered by attackers because we remark that they control the hosts of the pivoting tunnel. Unlike these proposals, we consider network flows because they are easier to collect and store; moreover, our analysis requires less processing costs than investigations carried out on raw traffic data [20].

3 Pivoting

In pivoting, attackers use a command propagation tunnel created among three or more internal hosts with the purpose of controlling a specific target. Past literature sometimes refers to pivoting with the term *island-hopping* [10], where attackers propagate commands through hosts in multiple LANs ("islands"). Many recent cyberattacks [3], [4], [5], [6] used pivoting in their propagation phase. It is of paramount importance to find novel approaches for early detection of pivoting because it would prevent attackers from reaching their target and damaging an organization.

Figure 1 reports an example of pivoting activity, where attackers have created a pivoting tunnel between four internal hosts belonging to three different LANs. From the

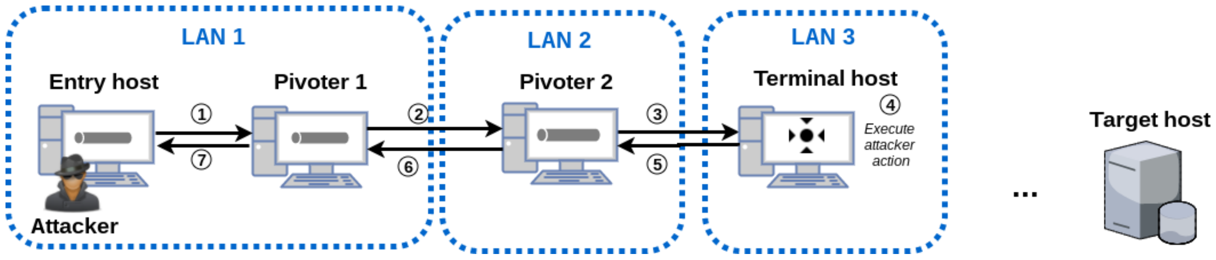


Fig. 1. Example of pivoting activity.

entry host, malicious commands are forwarded through intermediate hosts (called *pivoters*) to the last host of the pivoting chain, called *terminal host*, which executes these commands and returns the results. This last host can be the final target or it can be used to further extend the pivoting tunnel.

After having established a foothold within the internal network of the target organization, attackers typically perform the following steps:

1) **Reconnaissance**: attackers gather information about neighbor hosts through some active (e.g., port-scan) or passive means (e.g., by looking at the `arp` table).
2) **Compromise**: attackers compromise another internal host using a known or zero-day vulnerability; in such a way, they can increase the length of the pivoting chain, and the newly compromised host will become the terminal host as in Figure 1.
3) **Pivoting**: the commands of the attackers are propagated from the entry host to the terminal host through the pivoting chain. Any command issued to the terminal host is executed and the results are forwarded towards the entry host through the pivoting chain.

The first two actions are part of *lateral movement* [27] where attackers assume control of other hosts to get closer to their final targets, and the *pivoting* is the actual command propagation activity through the tunnel of compromised hosts. To perform pivoting, attackers may adopt different protocols and tools, which can be either standard (e.g., SSH or netcat), or ad-hoc software designed to avoid easy detection (e.g., meterpreter [30]). Depending on the chosen protocols, payloads may or may not be subject to encapsulation. For example, if attackers adopt SSH, then data is encrypted and encapsulated within protocol-specific headers; if they use netcat, then the original payload is forwarded without modifications.

It is important to highlight that detection techniques based on signatures [31] cannot detect pivoting, because they only perform pattern matching against packet headers and payloads, which is not sufficient to detect active command propagation tunnels. Some parts of the lateral movement phase (i.e., reconnaissance and compromise) may be detected by signature-based systems, but only if attackers perform active reconnaissance and uses publicly disclosed exploits [10].

We focus on the detection of pivoting involving internal hosts. Looking for suspicious communications among the internal and external traffic is a different and widely investigated problem by the literature on intrusion detection through signature-based [32], [33], [34], anomaly-based [35], [18], protocol-specific [36], [8] and attack-specific proposals [12], [11]. The approaches can be combined for a more accurate detection, but their integration is out of the scope of this paper.

There are several issues that make pivoting detection a tough problem. Command propagation is a rare event immersed in a huge amount of activities characterizing the network traffic of a medium-large organization. Since some pivoting activities are benign (e.g., for network administration), detection algorithms are affected by false alarms because malicious pivoting is an even rarer event. Finally, attackers can use a variety of tools and protocols in the attempt to evade detection. For these reasons, we propose an innovative approach that through the analysis of network flows identifies pivoting flow sequences and then integrates pivoting detection with the threat prioritization algorithm presented in Section 5. In such a way, we allow security analysts to inspect only few pivoting activities exhibiting the most suspicious behavior. Figure 2 outlines the main steps of the proposal.

4 Pivoting Detection

The proposed pivoting detection algorithm has the purpose of identifying all active pivoting tunnels within an internal network.

4.1 Pivoting detection as a temporal graph problem

We detect pivoting activities by searching for active command propagation tunnels. Network flows are the input data of our algorithm. A network flow represents a common way of summarizing traffic data [17] because it contains all information about connections, such as start-time, duration, transmitted bytes, involved IP addresses and port numbers. Using network flows instead of raw traffic data has the twofold benefit of simplifying data gathering and decreasing processing costs. A network

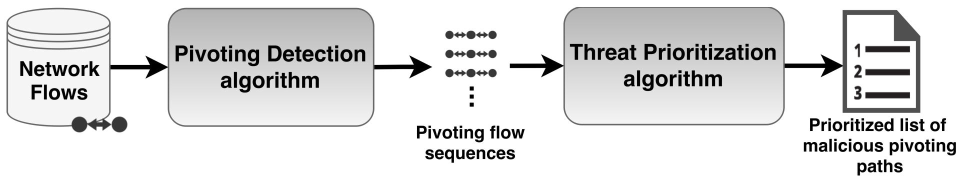

Network Flows
Pivoting Detection algorithm
Pivoting flow sequences
Threat Prioritization algorithm
1
2
3
Prioritized list of malicious pivoting paths

Fig. 2. Overview of proposed method.

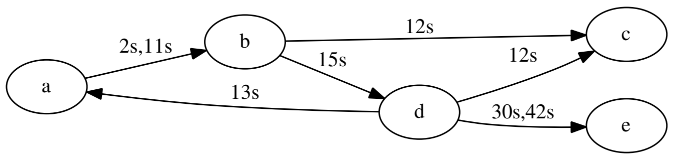


Fig. 3. Temporal graph representation of network flows between five hosts (a,b,c,d,e).

flow f can be defined as an ordered sequence:

$$f = (src, dst, p_{src}, p_{dst}, b_{in}, b_{out}, d, t) \quad (1)$$

where src and dst are the IP addresses of the source and destination hosts of the flow f; p_{src} and p_{dst} are the source and destination ports of their communications; b_{in} and b_{out} are the incoming and outgoing bytes; d is the duration (by default $120s$ [37]), and t is the timestamp corresponding to the beginning of the communication. For each network flow, src and dst represent the flow direction with respect to the node that started the communication, even if they keep exchanging packets in both directions. Flow direction can be reliably computed by traditional network security appliances through the analysis of packet headers and timings [37].

From network flows it is possible to model communications among internal hosts as a temporal graph [38] corresponding to a time window W, where nodes correspond to internal hosts and directed temporal edges represent flows. Command propagations in pivoting tunnels are represented by network flows connecting pivoters sequentially and separated by a low latency. For this reason, we introduce the concept of *maximum propagation delay* ε_{max}, defined as the maximum amount of time that can pass between two consequent communications to consider them as a part of the same pivoting activity.

We define a *pivoting flow sequence* $\mathcal{F}$ as an ordered set of flows $(f_1, f_2, ..., f_L)$, $L \in \mathbb{N},\ L > 1$, where:

1) all consecutive flows must be adjacent (dst of a flow is src for the following flow);
2) all connected nodes appear only once in the sequence;
3) consecutive flows are chronologically ordered;
4) consecutive flows are separated by at most ε_{max} time units.

We define a *pivoting path* $\mathcal{P}$ as an ordered set of hosts $(h_1, h_2, ..., h_N)$ for which at least one pivoting flow sequence exists. A pivoting path corresponds to one or more pivoting flow sequences, whereas each pivoting flow sequence corresponds to a specific pivoting path.

We illustrate some examples of flow sequences and pivoting paths by referring to the temporal graph shown in Figure 3. For the sake of clarity, multi-edges are represented by timestamp labels separated by a comma as in [39]. If we consider any value of $\varepsilon_{max} \geq 27s$, the temporal graph contains all the paths identified in Table 1. For the sake of simplicity, the represented flows only report src, dst and t.

TABLE 1
Example of pivoting paths and corresponding flow sequences from Figure 3 for $\varepsilon_{max} \geq 27s$.

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)

.

We observe that some paths are subsets of other paths, and paths involving the same vertices can occur at different timestamps. If we consider a maximum propagation delay $\varepsilon_{max} = 5s$, the pivoting paths of interest are reported in Table 2. This propagation delay defines the admitted tolerance for considering two flows as part of the same pivoting path.

TABLE 2
Example of pivoting paths and corresponding flow sequences from Figure 3 for $\varepsilon_{max} = 5s$.

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

4.2 Algorithm for pivoting detection

In Algorithm 1 we present a novel algorithm for pivoting detection that finds all the pivoting flow sequences

Algorithm 1: Algorithm for pivoting detection.

```
Input: List of $m$ temporal edges corresponding to time window
       $W$ ($Flows$), maximum propagation delay $\varepsilon_{max}$, minimum
       incoming and outgoing bytes $B^{in}_{min}$ and $B^{out}_{min}$, maximum
       flow duration $\delta_{min}$, maximum pivoting path length $L_{max}$
Output: List of pivoting flow sequences of length $\geq 2$
        (corresponding to pivoting paths)
1  // Initialization
2  $PivotingSequences \leftarrow$ emptyList();
3  $CandidateFlows \leftarrow$ emptyList();
4  for flow $f$ in $Flows$ do
5      if $(f.d \geq \delta_{min})$ and $(f.b_{in} \geq B^{in}_{min}$ and $f.b_{out} \geq B^{out}_{min})$
        then
6          Insert flow $f$ in $PivotingSequences$;
7          Insert flow $f$ in $CandidateFlows$;
8  // Look for possible pivoting flow sequences of
   length $\geq 2$
9  for flow sequence $\mathcal{F}$ in $PivotingSequences$ do
10     if length($\mathcal{F}$) $\geq L_{max}$ then
11         break;
12     $FoundSequences \leftarrow$
         $ExtendPivotingSequence(\mathcal{F}, CandidateFlows, \varepsilon_{max})$
13     Include $FoundSequences$ in $PivotingSequences$;
14 return List of elements in $PivotingSequences$ with length $\geq 2$;
15 // Function to find flow sequences of length
   $(L+1)$ given a sequence $\mathcal{F}$ of length $L$
16 Function ExtendPivotingSequence($\mathcal{F}$,$CandidateFlows$,$\varepsilon_{max}$)
17     $FoundSequences \leftarrow$ emptyList();
18     $h_{\mathcal{F}} \leftarrow$ last host in pivoting flow sequence $\mathcal{F}$
19     $t_{\mathcal{F}} \leftarrow$ lastest timestamp of $\mathcal{F}$
20     $FlowsWithinDelay \leftarrow$
         BinarySearch($CandidateFlows[t_{\mathcal{F}} : t_{\mathcal{F}} + \varepsilon_{max}]$)
21     for flow $f$ in $FlowsWithinDelay$ do
22         if (($f.src$ equal to $h_{\mathcal{F}}$) and ($f.dst$ not in sequence $\mathcal{F}$)) then
23             $NewSequence \leftarrow$ (sequence $\mathcal{F}$ with flow $f$);
24             Insert $NewSequence$ in $FoundSequences$;
25     return $FoundSequences$;
```

within a temporal graph representing network communications among internal hosts. We observe that once the pivoting flow sequences are found, it is immediate to enumerate the corresponding pivoting paths (as in Table 1). The pivoting detection algorithm takes the following input parameters:

- the temporal graph of internal network communications built over the time window W, represented as the list of its m edges. Without loss of generality, we consider that the list of edges (representing network flows) is ordered according to their timestamp;
- the maximum propagation delay ε_{max}, which is the maximum amount of time tolerated between two consecutive flows to be part of the same pivoting flow sequence;
- the minimum flow duration δ_{min}, which is the minimum duration of a network flow to consider it part of a pivoting path;
- the minimum incoming and outgoing bytes B^{in}_{min} and B^{out}_{min}, which is the minimum number of bytes transmitted in a network flow to consider it as a possible portion of pivoting;
- the maximum pivoting sequence length L_{max}, which is the maximum length of the pivoting paths that the algorithm will search for. This parameter may be seen as a terminating condition.

In Table 3, we report the most relevant symbols used in this paper.

TABLE 3
Symbol table.

Symbol	Description
$\mathcal{F}$	Pivoting flow sequence.
$\mathcal{P}$	Pivoting path.
$\mathcal{F}_{\mathcal{P}}$	Set of pivoting flow sequences associated with path $\mathcal{P}$.
W	Time window (e.g. 1 hour) analyzed by the pivoting detection algorithm.
m	Number of network flows within W.
$B^{in/out}_{min}$	Minimum number of incoming/outgoing bytes of a network flow for building flow sequences.
δ_{min}	Minimum network flow duration for building flow sequences.
L	Length of a pivoting flow sequence (number of edges).
L_{max}	Maximum pivoting flow sequence length tolerated by the pivoting detection algorithm.
ε	Command propagation delay in a pivoting tunnel.
ε_{max}	Maximum command propagation delay tolerated by the pivoting detection algorithm.

Algorithm 1 can be divided into two main phases: *initialization*, and *extending pivoting sequences*.

Initialization: This phase takes the m edges of the temporal graph as its input and stores them into two separate lists: $PivotingSequences$ contains the list of all the sequences of length $L = 1$ and is used to store new sequences as the algorithm proceeds; $CandidateFlows$ contains the flows that are evaluated for extending the existing sequences. In line 5, an initial pruning condition is reported as a function of the inputs: for the analysis of pivoting detection, only flows with at least δ_{min} duration and B^{in}_{min} and B^{out}_{min} bytes are considered for this analysis. If $B^{in}_{min} = B^{out}_{min} = \delta_{min} = 0$, then the m input flows are considered without pruning. Conditions in line 5 are considered because any pivoting activity has to last for some time (since attackers are interacting through some command interpreter), and has to transfer some bytes in both directions (the propagated command, and the corresponding response).

Extending pivoting sequences: This is the core phase of the algorithm that repeats a common function multiple times. In line 9 there is a cycle that iterates through all the $PivotingSequences$ found so far. Initially, it considers only sequences of length $L = 1$ corresponding to the flows themselves. For each flow sequence $\mathcal{F}$ of length L in the $PivotingSequences$, the algorithm executes a function that searches for all possible flow sequences of length $(L+1)$ that extend $\mathcal{F}$. This function is called $ExtendPivotingSequence$ (line 16), and takes as its input a pivoting flow sequence $\mathcal{F}$, a maximum propagation delay ε_{max} and the $CandidateFlows$. The algorithm extracts the last host in the pivoting flow sequence $h_{\mathcal{F}}$

and its most recent timestamp $t_{\mathcal{F}}$. Then, it performs a binary search in line 20 on the $CandidateFlows$ in the time interval $[t_{\mathcal{F}}, t_{\mathcal{F}} + \varepsilon_{max}]$. This binary search is admissible because the $CandidateFlows$ are listed in sorted order of timestamp. The results of the binary search are stored in $FlowsWithinDelay$, which are then iterated to check whether a flow $f \in FlowsWithinDelay$ can extend the currently analyzed pivoting flow sequence $\mathcal{F}$. There are two conditions in line 22 that must be verified for $\mathcal{F}$ to be extended with f:

- the source node of f must be equal to $h_{\mathcal{F}}$ (which is the last host of $\mathcal{F}$);
- the destination node of f must not be in sequence $\mathcal{F}$ (it must be a new host in the sequence).

Whenever both conditions are satisfied, a new pivoting sequence is added to the $FoundSequences$ list, and then returned. By repeating the cycle on line 9 until a pivoting path of length L_{max} is reached (line 11) or until no flow sequences can be further extended, all the pivoting sequences of length at most L_{max} are enumerated. We observe that in line 14 the algorithm returns only pivoting sequences of length $L \geq 2$, because those with $L = 1$ are the flows themselves.

Let us describe the algorithm by considering the example graph in Figure 3 and the function $ExtendPivotingSequence$ with parameters $\varepsilon_{max} = 5s$, flow sequence $\mathcal{F} = ((a, b, 11s))$ of length $L = 1$. For the sake of simplicity, we report only (src, dst, t) of each flow. After the binary search in line 20, the flows within delay $[11s, 11s + 5s]$ are: $(a, b, 11s), (b, c, 12s), (d, a, 13s), (d, c, 12s), (b, d, 15s)$. Only the flows starting with b and not containing a can extend $\mathcal{F} = ((a, b, 11s))$. Only the flows $(b, c, 12s)$ and $(b, d, 15s)$ satisfy both conditions, and hence the new extended sequences found are $((a, b, 11s), (b, c, 12s))$ and $((a, b, 11s), (b, d, 15s))$. It is immediate to get the corresponding pivoting paths (a, b, c) and (a, b, d).

4.3 Computational complexity

The proposed algorithm for pivoting detection has an overall worst-case time complexity of:

$$\mathcal{O}(m^{L_{max}} \cdot \log_2(m) \cdot \tau) \tag{2}$$

where m is the number of network flows within the window W, L_{max} is the maximum pivoting tunnel length we are looking for, and τ is the maximum number of flows between any $[t, t + \varepsilon_{max}]$ interval. A proof of Eq. 2 is reported in Appendix A (see Supplementary Material) along with a comparison against two possible alternatives: the *subgraph isomorphism* and the *brute force enumeration* algorithms.

For small values of ε_{max} representing the common case, the parameter $\tau \ll m$, hence the complexity may be simplified as follows:

$$\mathcal{O}(m^{L_{max}} \cdot \log_2(m)) \tag{3}$$

Although the computational complexity of the Eq. 2 remains high, in Section 6 we verify through a large set of experiments that the proposed algorithm is efficient and applicable to real contexts. In particular, we can limit the propagation time ε_{max} because in many real attacks [30], [3], [4], [5], [6] commands are propagated as fast as possible, thus leading to values of ε of few milliseconds. Although advanced attackers could introduce some delays to evade detection, values of ε higher than few seconds make pivoting unpractical. Moreover, post-exploitation activities, such as command execution or data exfiltration, require flows lasting for at least few seconds, and comprising at least some tens of bytes. These conditions allow the pruning phase of the pivoting detection algorithm to reduce up to 90% the number of edges with respect to the worst case. As shown in Section 6, in practice the computational cost decreases significantly, and the execution time in a COTS machine renders the proposed algorithm applicable even for large network environments.

5 Threat prioritization

Some pivoting activities may be benign, hence it is important to avoid or limit the false alarms and let the security analysts focus on the most likely threats. For this reason, we integrate the algorithm presented in the previous section with a threat prioritization approach that ranks the detected paths on the basis of their suspicious factors. We define the following threat indicators for each detected pivoting path:

- path novelty,
- reconnaissance activities,
- involved LANs,
- use of uncommon ports,
- anomalous data transfers.

Path novelty: The appearance of paths involving hosts that have never previously communicated may be related to some unauthorized activities, as also highlighted in [27]. Hence, the score of the communication path novelty should be high if all the communications occurring among the involved hosts and their ports have never been seen in the recent past. Taking into consideration the pivoting path $\mathcal{P}$, we consider the set $\mathcal{F}_{\mathcal{P}}$ of the flow sequences $\mathcal{F}$ associated with $\mathcal{P}$ (see Table 2), we define the score $N(\mathcal{P})$ as the highest percentage of novel communications between hosts involved in the path $\mathcal{P}$:

$$N(\mathcal{P}) = \max_{\mathcal{F} \in \mathcal{F}_{\mathcal{P}}} \left\{ \left(\frac{len(\mathcal{F}) - len(S_{\mathcal{F}})}{len(\mathcal{F}) + 1} \right) \right\} \tag{4}$$

where $N(\mathcal{P}) \in [0, 1]$, $len(\mathcal{F})$ is the number of edges (flows) in $\mathcal{F}$, and $S_{\mathcal{F}}$ is the longest common subsequence of $\mathcal{F}$ found among the flow sequences related to the path detected in the recent past runs of Algorithm 1. We define a common subsequence as any subsequence of $\mathcal{F}$ where all flows have the same src and dst, and either the same p_{src} or p_{dst} ports but not necessarily both.

This latter condition is useful because in client-server communications it is common to have different clients ports over time but only one server port. If $N(\mathcal{P}) = 1$, all communications within the path are novel; if $N(\mathcal{P}) = 0$, all communications have occurred in the past.

Reconnaissance activities: Attackers perform reconnaissance to look for neighbor hosts that they can compromise. The paths including hosts involved in such activities should have a high risk score $R(\mathcal{P})$ as defined below:

$$R(\mathcal{P}) = \frac{\text{n.hosts of } \mathcal{P} \text{ involved in recon.}}{len(\mathcal{P})} \tag{5}$$

where $R(\mathcal{P}) \in [0, 1]$, and $len(\mathcal{P})$ is the number of hosts in the pivoting path $\mathcal{P}$. This score reports the percentage of hosts within a path that have been involved in reconnaissance activities. As the reconnaissance detection problem is out of the scope of this paper, we rely on existing techniques, such as those proposed in [20].

Involved LANs: Reaching internal hosts not easily accessible from the external network is one of the main reasons leading attackers to adopt pivoting techniques [40]. Thus, paths including hosts of different LAN segments may be a risk signal that should be prioritized. We define the $Z(\mathcal{P})$ score as:

$$Z(\mathcal{P}) = \frac{\text{n.hosts of } \mathcal{P} \text{ in different LANs}}{len(\mathcal{P})} \tag{6}$$

where $Z(\mathcal{P}) \in [0, 1]$, and $len(\mathcal{P})$ is the number of hosts in a path $\mathcal{P}$. This score denotes the percentage of different LANs included in a path, where $Z(\mathcal{P}) = 0$ if all hosts belong to the same LAN, and $Z(\mathcal{P}) = 1$ if each host belongs to a different LAN.

Use of uncommon ports: Some ports, such as number 22 (SSH), 23 (Telnet), 443 (SSLH Multiplexing), 3389 (Windows remote desktop protocol) or 5938 (Team Viewer), are commonly used for benign tunneling activities. Others are uncommon and possibly risky. We consider an edge (flow) to be uncommon if both p_{src} and p_{dst} are uncommon ports. The related score is computed for each path as the maximum of the following ratio:

$$S(\mathcal{P}) = \max_{\mathcal{F} \in \mathcal{F}_{\mathcal{P}}} \left\{ \frac{\text{n.flows in } \mathcal{F} \text{ with uncomm. ports}}{len(\mathcal{F})} \right\} \tag{7}$$

where $S(\mathcal{P}) \in [0, 1]$, $\mathcal{F}_{\mathcal{P}}$ is the set of flow sequences associated with the pivoting path $\mathcal{P}$, and $len(\mathcal{F})$ is the number of flows contained in $\mathcal{F}$. This score represents the maximum percentage of uncommon ports used in any communication of a pivoting path. For example, $S(\mathcal{P}) = 0.5$ when there is at least one flow sequence of $\mathcal{P}$ whose 50% of the flows use uncommon ports.

Anomalous data transfers: Attackers often leverage pivoting tunnels to perform data exfiltrations. A method to prioritize such activities is to verify whether the amount of data exchanged among the hosts of a given path, considering all its corresponding flow sequences, has increased with the respect to the recent past. To this purpose, we define the $E(\mathcal{P})$ score as follows:

$$E(\mathcal{P}) = \begin{cases} 1, & \text{if a data transfer anomaly is detected} \\ 0, & \text{otherwise} \end{cases} \tag{8}$$

To detect whether an anomaly occurred, we apply the boxplot rule [41] to the exchanged traffic among hosts.

Overall threat score: We compute an overall threat score $\mathcal{T}(\mathcal{P})$ of the path $\mathcal{P}$ as the sum of the indicators (we omit $\mathcal{P}$ for readability) where higher values denote paths that are more likely malicious:

$$\mathcal{T} = (N + R + Z + S + E), \mathcal{T} \in [0, 5] \tag{9}$$

All pivoting paths identified with the proposed detection algorithm are ranked according to their threat score. Section 6 validates the effectiveness of our threat prioritization score in different scenarios and with respect to other baseline algorithms.

6 Experimental evaluation

We demonstrate the effectiveness and feasibility of the proposed pivoting detection and threat prioritization algorithms when deployed on a real network of a large organization. We initially describe the dataset used for experiments. In Section 6.1 we demonstrate that our algorithm is able to detect all existing pivoting activities; we then inject different pivoting attacks into real network traffic, and show that our algorithms are able to detect and prioritize all the injected threats. Section 6.2 faces the additional challenge posed by the evasion strategies that may be adopted by expert attackers; we show the robustness of our proposals even against these attempts. In Section 6.3 we compare our solutions against two related algorithms. Finally, in Section 6.4 we measure the execution times of the proposed pivoting detection algorithm with the goal of demonstrating its applicability to real contexts.

Our test dataset consists of real network traffic captured by probes situated in a large organization, collected over a time span of over five months (160 days). The dataset is composed by more than half a billion $(657, 213, 849)$ network flows describing the internal network communications among $8, 198$ hosts. This scenario can be modeled as a graph where the number of temporal edges (flows) is significantly larger than the number of nodes (hosts). A portion of the used dataset has been made publicly available, and additional information is included in the Appendix B.

6.1 Pivoting detection and prioritization

We evaluate how the proposed algorithms for pivoting detection and threat prioritization perform under varying attack conditions. We initially execute the pivoting detection algorithm on the test dataset once per hour ($W = 60$ minutes) until each of the 160 days of the dataset has been analyzed. In typical pivoting activities the propagation delay is in the order of milliseconds [3], [4], [5], [6], hence we initially consider the performance with $\varepsilon_{max} = 1s$ and no limit on the maximum length of the pivoting path ($L_{max} = \infty$). Manual inspection of each path detected by the algorithm reveals that they are all true and benign pivoting activities corresponding to SSH tunneling and proxying. As the algorithm has been able to identify all tunneling activities occurring within the network for propagation delay $\varepsilon \leq 1s$, we can conclude that, in absence of attacks with propagation delay $\varepsilon > 1s$, it achieves 100% Accuracy.

As no malicious pivoting activity occurs in the testbed considered for evaluation, we emulate five classes of pivoting attacks, which are summarized in Table 4, and inject them into the real traffic traces. AC1 refers to an attacker creating an SSH tunnel and performing port-scans to detect the next victim, which is compromised after a brute-force SSH password guessing. In AC2, the attacker uses the SSH protocol for exchanging 30MB of data, but no brute-forcing nor active reconnaissance is performed. AC3, AC4 and AC5 are performed through the Metasploit toolset. The Attack Classes in Table 4 consider also increasing length of the pivoting chain to show that our algorithm can still detect longer chains and how this affects prioritization.

TABLE 4
Pivoting Attack Classes.

Attack Class	Vector	Len	Recon	LANs	Data
AC1	SSH	2	✓	2	10 MB
AC2	SSH	2	✗	2	30 MB
AC3	Metasploit	4	✓	5	100 MB
AC4	Metasploit	3	✗	4	< 1 MB
AC5	Metasploit	4	✗	1	5 MB

The network traffic generated by each Attack Class is collected as netflow data and then injected into each day of the real traffic dataset. This injection process is realized through an ad-hoc script that merges each attack flow with the dataset by replacing the IP addresses of the virtual pivoter hosts with those of real hosts. To avoid bias when choosing real hosts as pivoters, we identify two main sets of hosts: ω and β. The ω set contains the hosts with a total number of communications above the 95-th percentile, while the β set those below the 5-th percentile. In other words, ω and β sets contain high-activity and low-activity hosts, respectively. Since it is easier to prioritize pivoting activities in β because it contains hosts that rarely interact, we can observe that ω and β represent a worst- and best-case scenarios for prioritization, respectively.

We execute the pivoting detection algorithm on the entire injected dataset for $\varepsilon_{max} = 1s$, no bounds on the maximum length of the pivoting paths ($L_{max} = \infty$), and $W = 60$ minutes. Our algorithm is able to correctly detect all the injected pivoting paths thus achieving a 100% Precision. Unfortunately, in addition to malicious pivoting paths, the algorithm detects some benign pivoting activities present in the dataset. This result motivates our choice of relying on the threat prioritization algorithm presented in Section 5. The goal now is to assess whether this algorithm is actually able to prioritize malicious pivoting activities by placing them at the top positions in the ranking. Table 5 reports the results of the threat prioritization algorithm for each Attack Class and each set of injected hosts. Each row displays the average rank and its standard deviation obtained by each Attack Class after considering all the pivoting paths found within each day in the dataset. Lower values of rank correspond to higher prioritization. For example, if a pivoting path is ranked in position 1, it implies that it has the highest likelihood of being malicious among all pivoting paths detected in the same day. We observe that all attacks of each class have always an average rank lower than 2 even for the worst-case high-activity hosts belonging to the ω set. Moreover, a standard deviation that is always below 1.4 indicates that the vast majority of malicious pivoting paths are ranked in the top positions. These important results show that the threat prioritization algorithm is capable of assigning a high, stable rank to all the considered Attack Classes, thus ensuring the prioritization of the malicious pivoting activities.

TABLE 5
Performance of the threat prioritization algorithm.

Attack Class	average rank	standard deviation
AC1 (ω)	1.38	1.32
AC1 (β)	1.17	0.72
AC2 (ω)	2.01	1.18
AC2 (β)	1.55	1.04
AC3 (ω)	1.00	0.00
AC3 (β)	1.00	0.00
AC4 (ω)	1.13	0.51
AC4 (β)	1.14	0.68
AC5 (ω)	1.15	0.83
AC5 (β)	1.14	0.78

We can also observe that AC3 is always ranked first with standard deviation equal to zero, because the attacks in this class span over five LANs and involve about 100MB of exfiltration traffic (see Table 4). On the other hand, the attacks in AC1 and AC2 exhibit slightly lower ranks. This is motivated by the fact that these attacks are performed through SSH, which is a standard protocol often used by system administrators for legitimate tunneling operations; as expected, the threat scores of AC1 and AC2 are slightly lower than those performed through Metasploit (see Section 5). Nevertheless, the average ranks of these two classes are always lower than 3.

6.2 Evasion techniques

Our pivoting detection and prioritization algorithms are able to detect and properly prioritize malicious pivoting tunnels occurring in internal networks. We now further stress the algorithms by considering skilled attackers that employ evasion techniques based on the introduction of some propagation delay in their pivoting communications. Detecting these stealthy attacks requires that the algorithm works with higher values of the parameter ε_{max}. Increasing this parameter has two consequences on the results of the detection algorithm: its execution times increase; it may label as pivoting activities some flow sequences that pertain to normal traffic, that is, it may be affected by false positives. For these reasons, it is of paramount importance to evaluate the robustness of the detection algorithm against evasion techniques of expert attackers. To this purpose, we repeat the attacks of Table 4 by adding the propagation delays reported in Table 6, and then inject the delayed attacks in each day of the entire dataset. For the sake of completeness, we also consider delays equal or longer than 10 seconds, although they are unpractical in reality. The motivation is simple: if for example the attackers want to perform an action in a pivoting chain of 5 hosts with a delay of $\varepsilon = 25$s, they should wait for $5\,hosts \times 25s \times 2\,directions$, which is more than four minutes delay for each issued command.

TABLE 6
Emulated propagation delays ε for the Attack Classes.

	Attack Class				
	AC1	**AC2**	**AC3**	**AC4**	**AC5**
Delay	$2s$	$4s$	$8s$	$10s$	$15s$

We then execute the detection algorithm on the injected dataset for values of ε_{max} up to 30s and we do not set any bound on the maximum length of the pivoting path ($L_{max} = \infty$), while the size of the time-window is still set to $W = 60$ minutes. The detection results are reported in Table 7. From this table we can conclude that our detection algorithm is able to detect all the Attack Classes when it is executed with an adequate value of ε_{max}.

TABLE 7
Pivoting attack detection for increasing ε_{max}.

Attack Class	$1s$	$5s$	$10s$	$15s$	$20s$	$25s$	$30s$
AC1	✗	✓	✓	✓	✓	✓	✓
AC2	✗	✓	✓	✓	✓	✓	✓
AC3	✗	✗	✓	✓	✓	✓	✓
AC4	✗	✗	✓	✓	✓	✓	✓
AC5	✗	✗	✗	✓	✓	✓	✓

We then execute the threat prioritization algorithm and report in Table 8 the daily average and standard deviation of the rank obtained by attacks belonging to each Attack Class over the entire injected dataset. Rows refer to the attacks belonging to different Attack Classes and columns to different values of the parameter ε_{max}. Each cell reports the average rank and its standard deviation between parentheses. We observe that the average rank produced by our algorithm is always lower than 3 and its standard deviation is always lower than 2.

The accuracy of the prioritization algorithm is evaluated by denoting the *detection rate* as the percentage of days in which an injected Attack Class has been prioritized within the top 5 threats. Table 9 reports the detection rates for increasing values of ε_{max}. The best results are achieved on attacks of type AC3, where the algorithm always prioritizes the injected attacks within the top 5. As expected, the lowest detection rate (between 93% and 97%) is obtained for AC2 attacks because they represent a stealthy activity with standard SSH protocol and no reconnaissance (see Table 4). We also recall that ω represents the set of high-activity internal hosts, which are much more challenging to prioritize. Overall, we can be satisfied by the results in Table 9 because our algorithm prioritizes the large majority of Attack Classes with detection rates between 97% and 99%.

We can motivate these results by considering that the proposed approach integrates a *pivoting detection* with a *threat prioritization* phase. If the value of ε_{max} is high enough to tolerate the propagation delay inserted by the attackers, then the pivoting detection algorithm achieves 100% Recall, that is, all pivoting tunnels are detected. In particular, if $\varepsilon_{max} = 1s$ then all the detected flow sequences belong to actual pivoting paths, therefore achieving 100% Precision and 100% Recall (see Section 6.1). If we set $\varepsilon_{max} = 1s$ and the attackers perform some evasion techniques, the algorithm may be affected by some false negatives. In these cases, detection of evasive pivoting tunnels is achieved by setting $\varepsilon_{max} > 1s$ because any flow sequence with a propagation delay greater than $1s$ is either a false positive or an evasive pivoting attack, and is never benign because benign operations have short latencies. To overcome the limitations of the pivoting detection algorithm in case of evasive attackers, we designed the threat prioritization algorithm to rank the pivoting tunnels most likely related to threats in top positions, despite the presence of benign tunnels and false positives. We can conclude that the combination of the proposed pivoting detection and threat prioritization algorithms allows effective triage of even the most evasive attackers, as demonstrated by the results in Table 8.

6.3 Comparison with other detection algorithms

To the best of our knowledge, this paper proposes the first approach targeted to pivoting detection and threat prioritization that relies solely on network flows without any assumption on protocols and hosts. Nonetheless, we find meaningful to compare our solution with two algorithms relying on network flows, *SSHC* [8] and *WHL* [11], that we consider the most related to our approach. SSHC detects SSH-based attacks involving port-scans

TABLE 8
Threat prioritization: average ranking for increasing ε_{max}.

Attack Class	$1s$	$5s$	$10s$	$15s$	$20s$	$25s$	$30s$
AC1 (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)
AC1 (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)
AC2 (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)
AC2 (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)
AC3 (ω)	✗	✗	✓ 1.00 (0.00)	✓ 1.00 (0.00)	✓ 1.00 (0.00)	✓ 1.00 (0.00)	✓ 1.00 (0.00)
AC3 (β)	✗	✗	✓ 1.00 (0.00)	✓ 1.00 (0.00)	✓ 1.00 (0.00)	✓ 1.00 (0.00)	✓ 1.00 (0.00)
AC4 (ω)	✗	✗	✓ 1.26 (0.86)	✓ 1.26 (1.14)	✓ 1.21 (1.31)	✓ 1.21 (1.00)	✓ 1.21 (1.63)
AC4 (β)	✗	✗	✓ 1.21 (0.75)	✓ 1.21 (1.06)	✓ 1.17 (1.23)	✓ 1.17 (1.32)	✓ 1.17 (1.37)
AC5 (ω)	✗	✗	✗	✓ 1.26 (1.16)	✓ 1.21 (1.44)	✓ 1.21 (1.56)	✓ 1.21 (1.86)
AC5 (β)	✗	✗	✗	✓ 1.21 (1.15)	✓ 1.17 (1.28)	✓ 1.17 (1.29)	✓ 1.17 (1.54)

TABLE 9
Detection rate in top 5 for increasing ε_{max}.

Attack Class	1s	5s	10s	15s	20s	25s	30s
AC1 (ω)	✗	98.1%	97.5%	97.5%	97.0%	97.0%	97.0%
AC1 (β)	✗	98.1%	98.1%	98.1%	98.1%	97.5%	97.5%
AC2 (ω)	✗	94.4%	94.4%	94.4%	93.8%	93.8%	93.8%
AC2 (β)	✗	97.5%	97.0%	97.0%	95.0%	95.0%	95.0%
AC3 (ω)	✗	✗	100.0%	100.0%	100.0%	100.0%	100.0%
AC3 (β)	✗	✗	100.0%	100.0%	100.0%	100.0%	100.0%
AC4 (ω)	✗	✗	98.1%	98.1%	98.8%	98.8%	98.8%
AC4 (β)	✗	✗	99.4%	99.4%	98.8%	98.8%	98.8%
AC5 (ω)	✗	✗	✗	98.1%	98.8%	98.8%	98.8%
AC5 (β)	✗	✗	✗	99.4%	99.4%	98.8%	98.8%

TABLE 10
Comparison of detection algorithms.

Attack Class	**Alg** (ε_{max}**=15s**)	**SSHC**	**WHL**
AC1 (ω)	✓ 1.48 (1.58)	✓	✓
AC1 (β)	✓ 1.21 (1.10)	✓	✓
AC2 (ω)	✓ 2.27 (1.46)	✗	✗
AC2 (β)	✓ 1.81 (1.34)	✗	✗
AC3 (ω)	✓ 1.00 (0.00)	✗	✓
AC3 (β)	✓ 1.00 (0.00)	✗	✓
AC4 (ω)	✓ 1.26 (1.14)	✗	✗
AC4 (β)	✓ 1.21 (1.06)	✗	✗
AC5 (ω)	✓ 1.26 (1.16)	✗	✗
AC5 (β)	✓ 1.21 (1.15)	✗	✗

and brute-force password guessing; when these actions are performed through a remotely controlled internal host, they are true pivoting attacks. WHL evaluates variations in the graph of internal communications to detect malware propagations; in this case, rapid malware propagation through different hosts can be modeled as a pivoting attack.

Comparative results are presented in Table 10, where each row refers to a specific Attack Class. For this set of experiments we assume an advanced attacker that is trying to evade detection by applying the delays described in Table 6. Each column of Table 10 summarizes the results achieved by a different detection algorithm. The first column refers to the detection and prioritization algorithms proposed in this paper, in which $\varepsilon_{max} = 15$ seconds, $W = 60$ minutes and $L_{max} = \infty$. The last two columns refer to the detection results obtained by SSHC and WHL. Detected and not detected attacks are marked by ✓ and ✗ symbol, respectively. The proposed algorithm detects all pivoting paths, but it is also able to prioritize those representing real threats. For this reason, in Table 10 we also include the daily average rank and standard deviation associated to the attacks belonging to the same class. Since SSHC and WHL perform just detection with no prioritization, we report whether the injected pivoting attack has been detected or not.

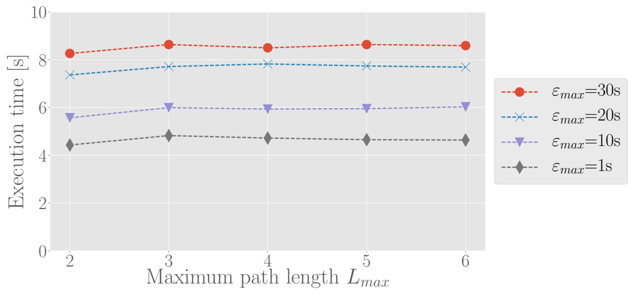
From these results, we can observe that attacks in AC1 can be detected by all three approaches. Our algorithm detects and prioritizes it within the top 5 hosts; SSHC recognizes the attack associated with a brute-force attempt, and WHL detects a sudden change in the communication structure due to the reconnaissance performed after pivoting. AC3 is always detected by our approach and by WHL, however it is not detected by SSHC because it does not involve brute-forcing of an individual host, but a reconnaissance on multiple hosts. Finally, we can observe that attacks in AC2, AC4 and AC5 cannot be detected by SSHC and WHL because they involve more subtle pivoting activities and they include a propagation delay between $2s$ and $15s$ introduced by the attackers to improve his chances of evasion. We can conclude that our algorithm is by far the most effective in detection and prioritization of pivoting activities even if the attackers adopt evasive attack strategies.

6.4 Execution times

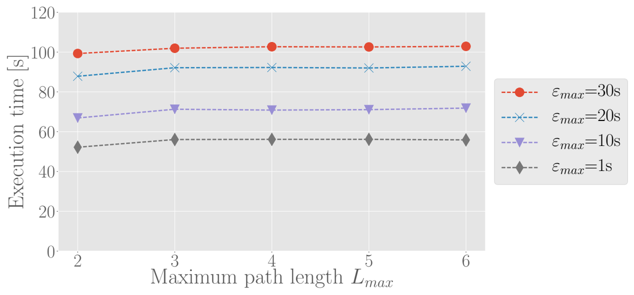
The computational complexity of pivoting detection algorithms is a fundamental challenge for evaluating if they can be applied to real contexts. As discussed in Section 4.3, the worst-case computational complexity of our algorithm is exponential with respect to the parameter L_{max} (see Eq. 2), that is, the maximum pivoting path length. Nevertheless, the parameter L_{max} is just a theoretical worst-case upper boundary because in real networks the large majority of pivoting paths have length $L = 2$ or $L = 3$. Hence, we show that the time required to analyze even large datasets has little dependence on L_{max}.

We execute the algorithm multiple times on the injected dataset, each time providing different input values for ε_{max} (1s, 10s, 20s, 30s) and L_{max} (2, 3, 4, 5, 6, ∞). The size of the time-window W is set to 1 hour and 12 hours. Analyses are performed on a COTS server equipped with one Intel Xeon E5-2609 v2 CPU with 4 physical cores and 128GB RAM. Figures 4 report the average execution times required for each run of the algorithm. The X-axis in both figures corresponds to different input values of L_{max}; the Y-axis represents the average time (in seconds) required for the computation, and each line refers to different input values of ε_{max}. For the sake of completeness, we report that the maximum standard deviation σ_{max} among all the experiments with time window $W = 1$ hour and $W = 12$ hours are of 4.8 seconds and 17.8 seconds, respectively. The results in these figures show that the execution times are almost constant with respect to the parameter L_{max} for $L_{max} > 3$ thus confirming that the upper bound L_{max} has no practical influence on the paths present in the real dataset.

As expected, the execution times increase for increasing ε_{max} values because this parameter affects the research space of the detection algorithm (see Section 4). Nevertheless, these results show that the amount of time required for analyzing even large datasets of flows is short (the majority of execution times are below 2 minutes even when for analyses of 12 hours of traffic), therefore demonstrating the applicability of the detection algorithm to large real networks.



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



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

Fig. 4. Execution times of the pivoting detection algorithm.

7 Conclusions

This paper presents a novel approach that integrates detection and threat prioritization of pivoting attacks. We formalize pivoting as a temporal graph problem, and then we devise an innovative algorithm for detecting all pivoting paths occurring within the network by analyzing the internal network flows. The reduction of false alarms related to benign pivoting paths is achieved through a novel threat prioritization algorithm that considers different threat indicators typical of malicious pivoting activities. Extensive experimental evaluation on a real dataset consisting of nearly 650M communications collected over more than five months shows that the proposed approach is able to effectively detect and prioritize malicious pivoting activities even against attackers that adopt evasion techniques. We also show that the short execution times (few minutes to analyze one day of traffic of a large organization) make the proposal applicable to real contexts.

We should consider that a modern defensive system adopts multi-layer analyses, hence it is important to evidence that the proposed algorithm can be integrated with any other detection schemes based on black- and white-lists of hosts, on analyses of DNS traffic and of internal-to-external traffic. Similar extensions can make detection of malicious activities much more effective.

References

[1] "Mandiant M-Trends 2015," https://www2.fireeye.com/rs/fireye/images/rpt-m-trends-2015.pdf, visited in Sep. 2017.

[2] "Cisco Annual Security Report (2014)." http://www.cisco.com/web/offer/gist_ty2_asset/Cisco_2014_ASR.pdf, visited in Sep. 2017.

[3] C. Tankard, "Advanced Persistent Threats and how to monitor and deter them," *Elsevier Network Security*, 2011.

[4] "McAfee Technical Report on Night Dragon Operation." https://www.mcafee.com/hk/resources/white-papers/wp-global-energy-cyberattacks-night-dragon.pdf, visited in Sep. 2017.

[5] "Analysis of the Cyber Attack on the Ukrainian Power Grid." http://www.nerc.com/pa/CI/ESISAC/Documents/E-ISAC_SANS_Ukraine_DUC_18Mar2016.pdf, visited in Sep. 2017.

[6] L. Ayala, "Active Medical Device Cyber-Attacks," in *Cybersecurity for Hospitals and Healthcare Facilities*. Springer, 2016, pp. 19–37.

[7] "WikiLeaks Vault7: Archimedes documentation." https://wikileaks.org/vault7/#Archimedes, visited in Sep. 2017.

[8] L. Hellemons, L. Hendriks, R. Hofstede, A. Sperotto, R. Sadre, and A. Pras, "Sshcure: a flow-based ssh intrusion detection system," in *IFIP AIMS*. Springer, 2012.

[9] A. Sperotto, R. Sadre, P.-T. de Boer, and A. Pras, "Hidden markov model modeling of ssh brute-force attacks," in *DSOM Workshop*. Springer, 2009.

[10] F. Valeur, G. Vigna, C. Kruegel, and R. A. Kemmerer, "Comprehensive approach to intrusion detection alert correlation," *IEEE TDSC*, 2004.

[11] M. P. Collins and M. K. Reiter, "Hit-list worm detection and bot identification in large networks using protocol graphs," in *RAID*. Springer, 2007.

[12] P. Li, M. Salour, and X. Su, "A survey of internet worm detection and containment," *IEEE Communications Surveys & Tutorials*, vol. 10, no. 1, 2008.

[13] M. Feily, A. Shahrestani, and S. Ramadass, "A survey of botnet and botnet detection," in *Emerging Security Information, Systems and Technologies, 2009.* IEEE, 2009, pp. 268–273.

[14] L. Bilge, D. Balzarotti, W. Robertson, E. Kirda, and C. Kruegel, "Disclosure: detecting botnet command and control servers through large-scale netflow analysis," in *Proceedings of the 28th Annual Computer Security Applications Conference*. ACM, 2012, pp. 129–138.

[15] G. Gu, R. Perdisci, J. Zhang, W. Lee *et al.*, "BotMiner: Clustering Analysis of Network Traffic for Protocol-and Structure-Independent Botnet Detection." in *USENIX Security*, 2008.

[16] D. R. Ellis, J. G. Aiken, K. S. Attwood, and S. D. Tenaglia, "A behavioral approach to worm detection," in *Proc. ACM Work. Rapid Malcode*, 2004.

[17] B. Li, J. Springer, G. Bebis, and M. H. Gunes, "A survey of network flow applications," *Journal of Network and Computer Applications*, 2013.

[18] M. Marchetti, F. Pierazzi, M. Colajanni, and A. Guido, "Analysis of high volumes of network traffic for advanced persistent threat detection," *Computer Networks*, 2016.

[19] S. T. Zargar, J. Joshi, and D. Tipper, "A survey of defense mechanisms against distributed denial of service (ddos) flooding attacks," *IEEE Communications Surveys & Tutorials*, 2013.

[20] A. Sperotto, G. Schaffrath, R. Sadre, C. Morariu, A. Pras, and B. Stiller, "An overview of ip flow-based intrusion detection," *IEEE Communications Surveys & Tutorials*, 2010.

[21] A. Liu, M. Dong, K. Ota, and J. Long, "PHACK: An efficient scheme for selective forwarding attack detection in WSNs," *Sensors*, 2015.

[22] J. Wu, K. Ota, M. Dong, and C. Li, "A hierarchical security framework for defending against sophisticated attacks on wireless sensor networks in smart cities," *IEEE Access*, 2016.

[23] A. Furtună, V.-V. Patriciu, and I. Bica, "A structured approach for implementing cyber security exercises," in *IEEE COMM*, 2010.

[24] J. L. Obes, C. Sarraute, and G. Richarte, "Attack planning in the real world," *Springer Work. on Intelligent Security*, 2010.

[25] M. Chapman, G. Tyson, P. McBurney, M. Luck, and S. Parsons, "Playing hide-and-seek: an abstract game for cyber security," in *Proc. ACM Int. Work. on Agents and CyberSecurity*, 2014.

[26] J. R. Johnson, E. Hogan *et al.*, "A graph analytic metric for mitigating advanced persistent threat," in *IEEE ISI*, 2013.

[27] R. Brewer, "Advanced persistent threats: minimising the damage," *Elsevier Network Security*, 2014.

[28] M. A. Noureddine, A. Fawaz, W. H. Sanders, and T. Başar, "A game-theoretic approach to respond to attacker lateral movement," in *Int. Conf. Decision and Game Theory for Security*. Springer, 2016.

[29] A. Fawaz, A. Bohara, C. Cheh, and W. H. Sanders, "Lateral movement detection using distributed data fusion," in *IEEE SRDS*, 2016.

[30] "Penetration Testing Software." https://www.metasploit.com/, visited in Sep. 2017.

[31] D. E. Denning, "An intrusion-detection model," *IEEE TSE*, 1987.

[32] "Snort manual," http://manual.snort.org/, visited in Sep. 2017.

[33] C. Kruegel, W. Robertson, and G. Vigna, "Using alert verification to identify successful intrusion attempts," *Practice in Information Processing and Communication*, 2004.

[34] F. Pierazzi, S. Casolari, M. Colajanni, and M. Marchetti, "Exploratory security analytics for anomaly detection," *Computers & Security*, 2016.

[35] C. Kruegel and G. Vigna, "Anomaly detection of web-based attacks," in *ACM CCS*, 2003.

[36] L. Bilge, E. Kirda, C. Kruegel, and M. Balduzzi, "EXPOSURE: Finding Malicious Domains Using Passive DNS Analysis," in *NDSS*, 2011.

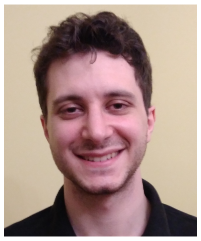
[37] "nProbe," http://www.ntop.org/products/netflow/nprobe/, visited in Sep. 2017.

[38] H. Wu, J. Cheng, S. Huang, Y. Ke, Y. Lu, and Y. Xu, "Path problems in temporal graphs," *VLDB*, 2014.

[39] A. Paranjape, A. R. Benson, and J. Leskovec, "Motifs in Temporal Networks," in *ACM WSDM*, 2017.

[40] A. Arora and V. Dutt, "Cyber security: evaluating the effects of attack strategy and base rate through instance based learning," in *Int. Conf. Cognitive Modeling*, 2013.

[41] T. T. Soong, *Fundamentals of probability and statistics for engineers*. John Wiley & Sons, 2004.



Giovanni Apruzzese is a PhD student at the International Doctorate School in Information and Communication Technologies (ICT), at the Department of Engineering "Enzo Ferrari" of the University of Modena and Reggio Emilia, Modena, Italy. He received the Master's Degree in Computer Engineering summa cum laude from the same University in 2016. His research interests include security analytics and network science. Homepage: http://weblab.ing.unimo.it/people/apruzzese/

Fabio Pierazzi is a Postdoctoral Researcher at the University of Modena and Reggio Emilia, Modena, Italy. In the same institute, he completed the Master's Degree in Computer Engineering 2013 and the PhD in Computer Science in 2017. During 2016, he spent 10 months as a visiting research scholar at the Department of Computer Science, University of Maryland, College Park, MD, USA. His research interests focus on security analytics, network security, malware classification, intrusion detection, and all solutions that combine cybersecurity and data mining. Home page: http://weblab.ing.unimo.it/people/fpierazzi.

Michele Colajanni is full professor in computer engineering at the University of Modena and Reggio Emilia since 2000. He received the Master degree in computer science from the University of Pisa, and the Ph.D. degree in computer engineering from the University of Roma in 1992. He manages the Interdepartmental Research Center on Security and Safety (CRIS), and the Master in Information Security: Technology and Law. His research interests include security of large scale systems, performance and prediction models, Web and cloud systems. Homepage: http://weblab.ing.unimo.it/people/colajanni/

Mirco Marchetti is a researcher at the Department of Engineering "Enzo Ferrari" of the University of Modena and Reggio Emilia (Italy). He received a Ph.D. in Information and Communication Technologies in 2009. His research interests include all aspects of system and network security, security for cyber-physical systems and automotive, cryptography applied to cloud security and outsourced data and services. Homepage: https://weblab.ing.unimore.it/people/marchetti