Intelligent Systems with Applications 16 (2022) 200106

Contents lists available at ScienceDirect

Intelligent Systems with Applications

journal homepage: www.journals.elsevier.com/intelligent-systems-with-applications

Role-based lateral movement detection with unsupervised learning

Brian A. Powell

The Johns Hopkins University Applied Physics Laboratory, Laurel, MD 20723, United States

ARTICLE INFO

Keywords:
Intrusion detection
Lateral movement
Unsupervised learning

ABSTRACT

Adversarial lateral movement via compromised accounts remains difficult to discover via traditional rule-based defenses because it generally lacks explicit indicators of compromise. We propose a behavior-based, unsupervised method of lateral movement detection that makes essential use system role—the functions it performs on the network—to identify anomalous inter-system connections. It is based on the observation that the remote hosts a particular system communicates with over time can be organized into a stable and learnable set of roles, and that the roles of the two hosts on either end of a normal connection determine the dynamics of the processes that support the connection, *e.g.* authentication of a workstation against a Domain Controller involves an idiosyncratic sequences of processes. If a process is compromised by an attacker and used to facilitate lateral movement, these normal patterns might be disrupted in discernible ways. We use unsupervised learning to cluster systems according to role, and then apply frequent-itemset mining to process sequences to establish regular patterns of communication between systems based on role. Rare process sequences might indicate malicious lateral movement, as might generic connections made to remote hosts with novel roles.

1. Introduction

Lateral movement is an essential step in the cyber attack life cycle. Following a successful intrusion, the adversary will move within the target network, from system-to-system, performing reconnaissance, stealing credentials, and escalating privileges. Lateral movement is prevalent, with over 60% of known attacks incorporating the technique CB (2019). It is also difficult to detect, with fewer than half of known tactics discovered on networks in 2020 Mandiant (2020). Activities conducted via compromised accounts resist discovery because they tend to appear normal: accesses are properly authenticated, and authorized software available on the compromised system can be used to further access: on US Government networks in 2020, for example, a credential compromise known as pass-the-hash was used to conduct 30% of lateral movement activities, and remote desktop protocol, a native Windows utility, supported 25% of connections, with only around 10% of activities involving code exploits Cybersecurity and Agency (2021). Rule-based intrusion detection systems struggle greatly to detect activities like these that lack recognizable signatures of compromise.

Machine learning, and artificial intelligence more broad- ly, offers new avenues of cyber attack detection. While an overwhelming amount of research Ahmad et al. (2021); Buczak and Guven (2016); Hagemann and Katsarou (2020); Magán-Carrión et al. (2020) has been conducted on applying machine learning to the detection of traditional classes of intrusion (malware propagation, denial of service, botnets, and others) Moustafa and Slay (2015); Tavallaee et al. (2009); KDD (1999), there remains an urgent need to develop capabilities more in line with the techniques of modern advanced threats: namely, the use of authorized services and compromised accounts to move quietly through the network. While most classes of intrusion have common "tells" (*e.g.* many connections in quick succession indicate worms or reconnaissance, high-volume data transfers indicate exfiltration), authorized lateral movement need not exhibit such explicit indicators ATT&CK (2019); Hausknecht (2019). But, if we can bring methods of statistical pattern recognition and behavioral modeling to bear on data describing normal user, system, and network characteristics, it becomes possible to discover malicious activities without explicit indicators or historical precedent in the enterprise environment.

In this paper, we apply unsupervised learning to discover anomalous connections among systems on a network. We focus on general connections—not only authentications—and our method is not reliant on any explicit indicators of compromise, artifacts related to adversary tactics or techniques, or the use of any particular protocol or service. The approach emphasizes that a system's *role* within the network—essentially its function—is a strong indicator of how, and with what other systems, it communicates. We build role-based profiles of systems that allow us to identify connections that deviate from patterns suggested by role; such anomalous connections might indicate adversarial lateral

E-mail address: brian.a.powell@jhuapl.edu.

https://doi.org/10.1016/j.iswa.2022.200106
Received 3 November 2021; Received in revised form 7 July 2022; Accepted 24 July 2022
Available online 2 August 2022

movement.

These profiles model the timing characteristics of the processes involved in inter-system connections. All inter-system connections are initiated and maintained by processes running on the hosts involved: we hypothesize that, in general, normal process dynamics follow patterns based on the roles of the systems involved in the connection. Normal system functions, like authentications, network time protocol (NTP) syncs, and Domain Name System (DNS) queries, tend to involve recognizable combinations of processes over time; for example, when a Windows client authenticates against a Domain Controller, the process `lsass.exe` generally executes in close succession with the Security Reference Monitor (via the process `ntoskrnl.exe`) Russinovich et al. (2013). The former process enforces local security policy on the system, and the latter provides routines enabling access controls. Meanwhile, Domain Controllers interact with NTP servers or DNS servers for entirely different purposes, and so involve different processes and/or process dynamics.

Reliable patterns of normal behavior can be used to spot adversarial activities that disturb process timing or that employ novel processes relative to expectations based on system role, including specific techniques like process injection and hijacking ATT&CK (2021). If the adversary injects malicious code into an executable to open a connection to some remote system, it better be an executable that is normally involved in inter-system communications according to the remote system's role, and it better execute in the company of other executables in a recognizable temporal pattern. This generally constrains how, when, and to what other systems an attacker can move laterally.

Role-based models apply to individual systems of interest as follows: unsupervised learning is used to organize all the system's remote peer systems (those systems with which it connects over some time period) into roles based on port usage. Then, historical time series of process executions between the system and each of its peers are extracted from connection logs (*e.g.* Carbon Black Network Connect). The time series of connections to peers with the same role are then collected together and mined for patterns; these patterns (or some representation of them) constitute a model for how the system of interest communicates with remote systems of a particular role. When the system of interest makes a new connection, the role of the peer system is identified and the processes comprising the connection are tested against the appropriate model.

An obvious problem with this approach arises when there is a new connection to a system with a novel role, since then there is no historical process data from which to build a model, and the connection cannot be tested. However, a connection to a system with a novel role is an anomaly in itself, simply because it deviates from historical precedent. And these anomalies could well indicate malicious activity: attackers choose their movements based on efficiency or expediency, and might well link systems that have little business communicating otherwise. Examples abound: IoT devices with broad connectivity and no access controls offer a stealthy route through the network; however, many systems like workstations and core servers might rarely (if ever) communicate with them. Attackers can establish pivots on any compromised system through which they can channel command and control, route commands, or move data: these pivots might link systems that seldom directly communicate, *e.g.* workstation-to-workstation, file server-to-printer, mail server-to-workstation, and so on.

In summary, we propose an unsupervised framework that seeks to identify lateral movement on two levels. It essentially asks: should these two systems be talking? If so, are they talking in the correct way? This work is a contribution to the wanting and incomplete arsenal of methods capable of detecting apparently authorized lateral movement, a major component of modern cyber intrusion. To the author's knowledge, there are several novel aspects of this work. It is the first to use system roles to pattern inter-system connections, and to apply these patterns to the problem of lateral movement detection. The unsupervised approach to role assignment is also original, though similar methods exist which will be reviewed in the next section. No previous intrusion detection framework has explicitly examined the process dynamics involved in the initiation and maintenance of inter-system connections; as discussed in the next section, there is considerable similar research on the analysis of system call and Unix command sequences, but these apply to host-level intrusion detection (compared to inter-host, or network-level intrusion detection) and analysis of inter-system processes poses unique challenges, like the need to capture the temporal structure within process sequences. Finally, we emphasize the use of unsupervised learning as the detection paradigm, in contrast to supervised approaches that require labeled samples of sufficiently realistic activities to be useful in the deployed environment.

2. Related Work

This analysis concerns the detection of malicious connections within enterprise networks. Though generic, in the sense that the method is agnostic to protocol, technique, or payload, its benefit is most clearly seen as a means of detecting authenticated connections, for which explicit indicators of compromise and other rules are generally lacking. It is therefore applicable to the authenticated lateral movement by an external adversary via compromised accounts, but also to malicious insider and masquerade activity, which likewise proceed via authorized means.

There are two broad areas of prior research relevant to this work. The first has to do with system classification, by which we group systems into roles; the second has to do with the analysis of sequences of system events (user activities, system calls, process executions) for anomaly detection. We review the related literature for each separately below.

2.1. System classification

The unsupervised organization of systems into roles is the first step in the proposed method. There are many ways to perform this categorization: we opt for a grouping based on system role—how the system behaves in terms of services provided to other hosts on the network. This method makes use of only a single source of passive connection log data, and employs standard clustering algorithms on features derived from these data. It is therefore a simple *example* of how the central concept, one emphasizing the importance of system role in detecting malicious connections, might be implemented in an operational setting. The references that follow review the basic literature and suggest alternative conceptions of system role that could be implemented in our framework.

The problem of system and traffic identification, in terms of application, protocol, or service type, has seen widespread development, primarily for applications to quality of service, policy enforcement, and network management. We emphasize that we are not looking to perform traffic classification, that is, the assignment of traffic flows to known protocol or application categories like peer-to-peer communications, online gaming, or onion routing Kim et al. (2008); Nguyen and Armitage (2008); Salman et al. (2020). Nor are we interested in performing traffic anomaly detection (to identify things like worm propagation or port scanning) using traffic classification (see, for example Lakhina et al. (2005); McHugh et al. (2008) and references therein). Our problem is considerably easier, in that we do not care to positively identify the ground truth label of a particular group of systems; we simply wish to partition them according to functional characteristics. In this light, we focus on previous literature with similar goals.

The use of network flow data formed the basis of several early studies. The important work of Xu et al. (2005) performed clustering on the (`src_ip`, `src_port`, `dst_ip`, `dst_port`) quadruplet to resolve systems into broad categories, like servers, proxies, and traffic related to scanning or exploits. Also working with flow records, Wei et al. (2006) analyzed traffic statistics like daily byte totals, number of distinct destinations, and average time to live to cluster systems into broad categories like transmission control protocol (TCP) servers, user datagram

protocol (UDP)-only servers, and user workstations. The related work of Erman et al. (2006) applied a variety of clustering algorithms to flow-derived characteristics like number of packets, mean packet size, and mean inter-arrival time of packets, to organize traffic according to common services like web, email, and peer-to-peer. The technique of *graphlets*, small undirected graphs yielding topological representations of host-to-host communications using internet protocol address (IP) and port data, was used for supervised traffic classification in Karagiannis et al. (2005). Graphlets have been applied to the problem of unsupervised classification in Himura et al. (2013).

The unsupervised resolution of enterprise systems into roles based on connection patterns was studied in Tan et al. (2003). Hosts that are similar with respect to their neighboring system sets are grouped together hierarchically; the success of this approach hinges on the extent to which functionally similar systems are also similar with respect to this metric. In Xu et al. (2011), a graph of system associations is introduced where system vertices share an edge if they have made connections to the same remote system; edges are weighted according to the cardinality of this shared neighbor set. Spectral clustering is then performed on this graph to organize systems into groups with similar neighbor interactions. Dewaele et al. Dewaele et al. (2010) perform clustering on nine connection-related quantities, including things like number of peers, ratio of the number of destination ports to number of peers, and ratio of the entropies of second and fourth bytes of destination IP addresses. This approach successfully categorizes traffic by common protocols, like web, peer-to-peer, mail, and DNS.

Rather than perform role analysis by proxy (via connection topologies and other characteristics), we use connection log data to instead categorize systems by port usage: this kind of profiling gets closer to a functional description than connectivity patterns. But, as stated earlier, there is no *a priori* reason that the kinds of categorization emphasized in the above works cannot be used in this approach.

2.2. Lateral movement and masquerade detection

There is a tremendous body of research on the problems of lateral movement and insider threat detection, summarized in the following reviews Bertacchini and Fierens (2009); Bridges et al. (2019); Liu et al. (2018a); Salem et al. (2008). Much prior research in this area has focused on the detection of exploitation and malware activity Bhuyan et al. (2014); Drašar et al. (2014); Tavallaee et al. (2009), generally in the context of the KDD CUP 99 intrusion dataset. In contrast to that diverse body of work, the method we develop here is a behavior-based detection capability aimed at generic lateral movement with no explicit indicators of compromise.

2.2.1. Network-based indicators

A relevant body of research focuses on malicious authenticated accesses (or login events) inside the network, applicable to both insider and outsider threats. Authentication graphs, which represent systems as vertices and logins as edges between them, were introduced as a tool for detecting anomalous login activity in Kent et al. (2015). These graphs can be constructed from authentication log data, and have been analyzed in a number of works in both supervised Bai et al. (2019); Goodman et al. (2015); Kaiafas et al. (2018) and unsupervised settings Bowman et al. (2020); Chen et al. (2012); Eberle and Holder (2009); Eberle et al. (2010); Holt et al. (2019); Powell (2020); Siadati and Memon (2017). Some authors have augmented login data with other data sources to characterize lateral movement: Chen et al. (2018) combine authentication records with data on general network connections and DNS queries to create graph features that are used to train an autoencoder to perform anomaly detection. In Bian et al. (2019) data from network flows are combined with authentication records to build a supervised classifier to detect malicious connections. Drawing connection and command & control data from widely deployed monitors across the network, Bohara et al. (2017); Fawaz et al. (2016) create graphs of all network connections and seek to identify long chains of connections indicating multi-pivot lateral movement.

Long chains of pivoting activity are also the target of the flow-based detection schemes presented in Apruzzese et al. (2020); Husàk et al. (2021). Bipartite user-system graphs created from multiple data sources, including login, web access, email, and file access records, are used to train a one-class learner in Gamachchi and Boztas (2017); Gamachchi et al. (2017) to identify malicious connections. In Yen et al. (2013), data from multiple sources, including authentication and proxy logs, and connection-oriented data like user agent strings, are used as features that are clustered to identify anomalies; this approach applies to both malicious external and internal connections. Application- and technique-specific methods have also been explored: in Djidjev et al. (2011), graphs representing secure shell (SSH) connections are mined to identify subgraphs corresponding to single user activity, where large subgraphs of low probability are flagged as anomalous. In Purvine et al. (2016), reachability graphs representing logical routes through the network are analyzed to identify at-risk systems as those with high importance and high reachability; these insights can be useful for devising mitigation strategies.

In contrast to these works, our framework assesses individual connections (versus chains of accesses) for anomaly, where the connections can be of any kind (versus primarily authentications). Importantly, though, our approach should not be viewed as an alternative to these methodologies, but instead as potentially mutually reinforcing since each targets related by different aspects of the lateral movement problem.

2.2.2. Host-based indicators

Another major body of work is focused on host-based indicators of malicious activity, and includes the analysis of system call traces, command line usage, file access patterns, and other user-driven behaviors. Many of these studies make use of categorical and discrete sequence anomaly detection schemes that relate to, and contrast with, our method of identifying anomalous process clusters in time series (see also the recent reviews Bridges et al. (2019); Liu et al. (2018b)).

The earliest works in this area applied association rule learning to sequences of Unix commands to create models of user behavior Forrest et al. (1996); Lee et al. (1997); Teng et al. (1990). The basic idea is that a user's shell commands follow a pattern, and that deviations from this pattern might indicate that the account has been compromised, or that the user is a malicious actor. The latter two studies explored the use of RIPPER Cohen (1995) to identify anomalous commands not predicted by the rules. Unix command analysis would go on to serve as the basis for a great number of further studies. In Davison and Hirsh (1998) a method, named Ideal Online Learning Algorithm, assumes that a Markov process governs command sequences and applies exponential weighting of past Unix command data to predict the next command; anomalies can be identified as prediction errors. Naive Bayes is used in Maxion and Townsend (2002) to ascertain which user of a closed set generated a certain command sequence; it is effective at this problem but it is not apparently designed to handle novel users (like an outsider threat). Hidden Markov Models (HMM) were soon applied to the problem of modeling command sequences: in Lane (1999), HMM was applied to Unix commands and compared against Naive Bayes, revealing only a slight preference for HMM. This suggested that indeed there might be important temporal structure in command sequences that can be modeled probabilistically; however, the analysis Yeung and Ding (2003) concluded that HMMs were inferior to simpler frequency-based analyses. Further, Iglesias et al. (2009b) argues that χ^2 testing of command subsequences is superior to HMMs unless the subsequences contain $\mathscr{O}(1000)$ commands. Similarity comparisons of fixed-length command sequences were explored in Lane and Brodley (1999), where it was found that profiled users can be accurately differentiated by their command behaviors. A combination of Naive Bayes and similarity measures was proposed in Sharma and Paliwal (2007), where similarity

based on Gaussian kernels was shown to yield general improvement over Naive Bayes. Cosine similarity was explored in Iglesias et al. (2009a), where fixed-length sequences of user commands are compared against a prototype sequence from each class of user. A comparison study of several different probabilistic models is found in Schonlau et al. (2001), where it is reported that hybrid multi-step and Bayes one-step Markov processes are the best models of Unix command sequences. In Balajinath and Raghavan (2001), genetic algorithms were employed to model fixed-length user command sequences, with anomaly scores influenced by the proportion of correctly predicted commands in the sequence.

The other major subject of behavioral profiling useful for the detection of masquerades and account compromise is the system call trace: the sequence of processes used by a program to interact with the system's kernel. Malicious activity is expected to alter these sequences, and so models trained on normal system call sequences can be useful for anomaly detection. Just as program execution gives rise to a sequence of system calls, so too do network connections give rise to a sequence of inter-system processes. As we study the latter in this paper, several of the following approaches were consulted and will be discussed in connection with our use-case throughout the paper.

The earliest studies in this area are perhaps the works of Hofmeyr and collaborators, Hofmeyr et al. (1998); Kosoresow and Hofmeyer (1997). In these works, sequences of system calls are divided up into k-length subsequences and compared via Hamming distance, where this distance serves as an anomaly measure. In subsequent works, this technique is referred to as *sequence time delay embedding* (stide); we will have more to say about this approach later in the paper. In Warrender et al. (1999), stide was improved by replacing the Hamming distance with a simple mismatch count across the subsequence and compared with RIPPER and HMMs, where it was found that HMMs enjoyed the lowest false positive rate. Soon, recurrent Ghosh et al. (1999) and evolutionary neural networks Han et al. (2004) were brought to bear on the problem: both neural networks outperform stide. Neural networks with radial basis function units were considered in Ahmed and Masood (2009), where particular attention was paid to the window size defining individual attack subsequences among the longer sequences of system calls. This window size corresponds to k in the stide analyses, and, as with stide, is found to have a strong influence on accuracy. The fixed-length comparison window was done away with in ESKIN (2001), where sparse Markov transducers were used to identify context-dependent window sizes; these models generally outperformed stide and related fixed-window size methods. Further work to identify meaningful groupings of system calls was conducted in Creech and Hu (2014), where an extreme learning machine was able to learn to group calls into appropriate semantic units; this model outperforms stide and HMMs.

An approach that builds dictionaries of anomalous system call sequences (rather than working to model normal sequences) was developed in Cabrera et al. (2001) using stide to identify them; this approach performs best in the context of specific Unix programs, like *sendmail*, where anomalies are circumscribed by the process. In Xie et al. (2014), a one-class support vector machine was trained on a set of labeled malicious system call data, including common exploitation tools like the Metasploit meterpreter and hydra login cracker. It performs well, but is not an anomaly detection system. Support vector machines based on sequence-similarity kernels were shown to outperform radial basis function kernels in Tian et al. (2007).

If sequences are interpreted as documents, they can be analyzed using methods of text categorization: in Liao and Vemuri (2002), individual system calls are *tf-idf*-weighted and assessed for anomaly with k-nearest neighbors distance; in Rawat et al. (2006) system calls are binary weighted with comparable results to Liao and Vemuri (2002). Following in this vein, Chen et al. (2005) applied support vector machines and neural networks to these features in a supervised setting. A "bag-of-system calls" representation was adopted in Kang et al. (2005), which reports superior performance over fixed-length methods like stide. Similar frequency-oriented system call representations were analyzed in Ye et al. (2001) with a variety of statistical techniques like χ^2, Hotelling's T^2, and Markov chain-based tests.

Rule learning was applied to system call traces and compared against stide in Tandon and Chan (2003), where it was shown to do better than stide at detecting true positives labeled according the 1999 DARPA attack taxonomy. An analysis based on frequent itemsets was developed in Chen and Dong (2006) and shown to outperform a support vector machine trained on system calls of different users. HMMs were explored in Qian and Xin (2007) who found the method "practicable" but note that proliferation of hyperparameters makes these models difficult to tune; various speed-ups were explored for HMMs in Hoang and Hu (2004); Hu et al. (2009) resulting in efficient and accurate models. A probabilistic model based on kernel states was developed in Murtaza et al. (2013) and shown to yield lower false positive rates than stide and HMMs. An interesting model was developed in Tapiador and Clark (2010) that uses information theoretic measures to identify attackers that are trying to deceive anomaly detection capabilities, through such actions as command padding. In Maggi et al. (2010), analysis of system calls together with their arguments, initiated in Kruegel et al. (2003); Tandon and Chan (2003), was expanded by first clustering system calls into classes, and then using these classes to build an HMM model to recognize anomalous calls. This approach is conceptually similar to ours, which also employs clustering to group together processes occurring as part of larger meaningful functions; in our case, though, we seek temporal clusters whereas in Maggi et al. (2010) the intent is to group together calls with similar arguments.

Other aspects of user behavior have been modeled for anomaly detection: Salem and Stolfo (2011) modeled user actions related to file and information access, with the expectation that adversaries (making use of compromised accounts) won't be as directed and efficient in this task. Other characteristics of file system access, like timestamps and file size, were analyzed in Mehnaz and Bertino (2017). Graphical user interface (GUI) interactions, including keyboard activity and mouse movements were modeled via support vector machine in Garg et al. (2006) and random forests were applied to Microsoft Word interactions in El Masri et al. (2014). Recurrent and convolutional neural networks were employed in Singh et al. (2019) to model the temporal behavior of various user behaviors, like logon times, and types of applications and amounts of data accessed, to detect anomalies. The performance of these models is comparable to the collection of methods discussed in a)Lazarevic (Ertoz).

We now briefly reflect on some of the methodologies above in the context of role-based lateral movement detection. Methods that rely on fixed-size subsequences, or that don't tolerate subsequences of arbitrary sizes, will not perform well against our use-case. Fixed-sized subsequences include processes that can be arbitrarily far apart in time, and hence causally unrelated. Such subseqeunces won't correspond to higher-level system functions and won't exhibit the associated regularity. We verify that stide, with its fixed window size, performs poorly against our use-case. Additionally, association rule learning fails for the simple reason that there isn't a reliable way to handle subsequences of only a single element, a common situation for our problem (corresponding to single isolated processes). Finally, HMM and recurrent neural networks are high-quality temporal models; however, in this case we only wish to understand the temporal structure of individual process clusters, since these correspond to the higher-level system functions that might be disturbed by the adversary. These process clusters are small time series, with the majority containing fewer than five elements, far too few to reliably train these kinds of models or to contain the rich temporal dependencies worthy of their power.

3. Experimental Set-up and Data Source

In this study we derive data from and perform all testing on computer systems on an operational enterprise network of several thousand hosts. The role-based models we will develop apply to individual systems (the *subject*), monitoring each subject's connections across the network. As we shall discover, due to its computational performance and alert volume, role-based detection is most appropriately applied to a relatively small subset of key systems on the network. We therefore conduct testing on a "watch list" of 125 high-value systems that would realistically be monitored as part of an intrusion detection strategy. This subset includes critical infrastructure like Exchange, virtualization, and certificate authority servers; systems containing important operational data, like file shares and research department databases; application servers that support administrative tasks; and workstations of highly-privileges users like Domain and Desktop Administrators. The primary data source leveraged for our models is Carbon Black Network Connect logs retrieved from a central security incident and event management (SIEM) system, with several weeks of historical data retention.

All analysis was conducted on a Linux virtual machine (dual Xeon processors, 8 GB RAM) residing in an on-premises cloud environment, using a representational state transfer application programming interface (RESTful API) to query the SIEM. All code is custom Python, making use primarily of sklearn libraries.

4. Organizing Systems by Functional Role

The first step in building role-based models is the organization of a subject system's peers (those systems with which it connects over some time period) into groups based on functional *role*. We proceed under the assumption that the ground truth role of each peer system is unknown, and must instead be inferred from connection data. This avoids the need to manually assign each system on the network to one of a pre-defined and comprehensive set of roles. To organize a subject's peer systems—its *neighborhood*—into roles, we propose a system classification method based on port usage: each system is given a *server profile* based on the local ports it uses to *serve* data to other systems over some time period. Systems can additionally be given a *client profile* based on the remote ports of servers accessed by the system over some time period. These two profiles are meant to summarize essentially how a particular system acts as a server and how it acts as a client. These profiles take the form of vectors in "port space" with values quantifying the usage of the port. The term *usage* is fairly general: one could consider the relative amount of data sent over the port, the relative number of connections involving the port, or something else. A more coarse-grained representation would be a simple binary "on/off" for each port.

Roles are assigned by clustering the server and/or client profiles of peers in the subject's neighborhood: systems whose profiles belong to the same cluster are considered to have the same role. Roles are never positively identified (that is, given descriptive labels like "Domain Controller"); they are merely bins for organizing systems. Furthermore, a peer system's role depends on the other peers in the subject's neighborhood (since the number and quality of clusters depends on the dataset); as a result, the same system might assume two different roles in the neighborhoods of two different subjects.

One important caveat of this approach is that, while servers of a certain type (say, Domain Controllers) need not make use of "standard" ports for its functions (because we are not interested in positively identifying the system role), they all need to make use of the *same* ports for the same functions (*e.g.* Kerberos must be listening on the same port on all Domain Controllers, whatever it happens to be).

We discuss this procedure in more detail below.

4.1. Creating a server profile

To create a server profile, a data source is needed that records information about port-to-port connections involving the system. Relevant data fields are the IPs of the systems and the ports involved in the connection, and timestamp. Network traffic data, like Netflow, and endpoint logs, like Carbon Black Network Connect (netconn) data, are two suitable sources that provide these data. In this study, we make use of netconn data collected on a large number of endpoints within a real, operational enterprise network; specific fields of interest are `local_ip`, `remote_ip`, `local_port`, `remote_port`, and `timestamp`.

In what follows we walk through how to build a server profile for a single system, which we take to be a Domain Controller (DC) on a Windows network. First, the netconn database is queried for all records with either `local_ip` or `remote_ip` matching the DC's IP address. We are interested in records across a time period sufficiently-long to capture a representative sample of connections; time enough to include several thousand records is a good rule of thumb, but some experimentation might be needed.

To create the DC's server profile, we must identify which ports local to the DC are serving data to peer systems. We start with the list of all DC ports found in the record: these are the `local_port`s when the DC IP matches `local_ip` and the `remote_port`s when it matches `remote_ip`. This list obviously includes both client ports (those ports local to the DC that are initiating connections to remote systems) and server ports (those ports local to the DC that are receiving connections initiated by remote systems). Client ports are generally ephemeral, high-numbered ports that are chosen at random when the DC initiates a connection; as such, we do not expect to see many instances of such ports in the record. We therefore implement a basic heuristic for deciding which ports are most likely server ports: we rank ports in descending order by number of connections in the record. Fig. 1 shows this ranking for the DC.

Notice that there are only appreciable numbers of connections associated with the first dozen or so ports: these are almost certainly server ports. We impose a simple rule to collect only the most-used ports: we keep a port if its connection count is within a fixed factor of the count of the consecutively higher-ranked port. In this study, we select a factor of three, which corresponds to the red horizontal line in Fig. 1. For example, port 636 had 8289 connections giving a cut-off of $8289/3 = 2763$ for the next highest-usage port; since port 139 had only 2282 connections, it is not included. With this rule the system's server profile is based only on the highest-usage server ports, and client ports are eliminated. Fig. 2 gives a schematic representation of the DC's server profile in terms of connection proportion: we can readily verify that these are standard ports associated with the major server functions of the DC (*e.g.* Kerberos authentication, network time protocol, lightweight directory access protocol, remote procedure calls, etc).

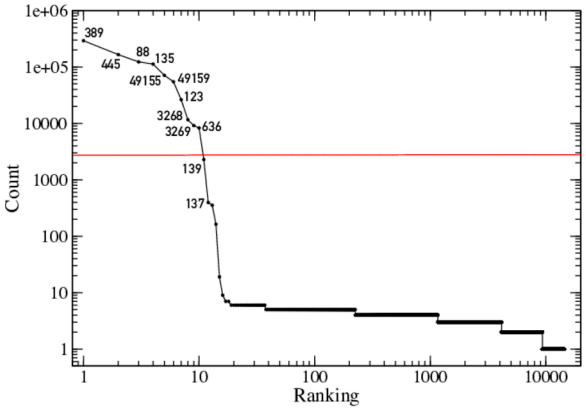

Fig. 1. Domain Controller ports ranked according to the number of connections they were involved in over a 24-hour period. Only the most-used ports are numbered. Only those ports above the red line will be used to profile the system.

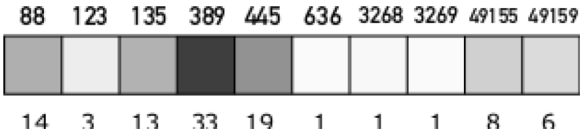

Fig. 2. Domain Controller server profile schematic. Port numbers across the top label the boxes, which are colored according to the percentage of all connections they were involved in over a 24-hour period. Percentages are provided numerically below each box.

The size of the DC's server profile is typical of other servers—at most a dozen or so dimensions, making it a succinct summary of how the system acts as a server.

4.2. Creating a client profile

The creation of a system's client profile parallels the above discussion, but here we are interested in understanding how the system tends to act as a *client*. For this purpose, we focus on the remote ports accessed by the system, which are the `local_port`s when the DC IP matches `remote_ip`, and the `remote_port`s when it matches `local_ip`. We again need to eliminate ephemeral ports on the remote systems (which are associated with *their* client activity) so that only remote server ports are considered. The same usage ranking heuristic employed to create the server profile is useful here as well, but we won't repeat the details. The end of this process results in a client profile vector akin to the server profile vector of Fig. 2.

4.3. Identifying system roles

To organize a subject's peer systems into roles, we first must identify all of its peers over some time period. Since we are focused on detecting lateral movement, which is internal to the system's network, only systems internal to the network are considered. Once the list of peers is obtained, each one must be profiled using the above method. Then, similar profiles must be grouped together: these groups are the system *roles*. In Fig. 3, a small example neighborhood of three peer systems is shown: the DC from above, a virtualization server (with active ports 80, 135, 443, and 445), and a voice over IP (VoIP) server (with active ports 5060 and 8443) in the community. Notice how the DC's profile has been expanded to include (zero) entries for the other systems' ports (and likewise for each of them); this is done so that the peers can be clustered together.

We explore several clustering techniques and profile representations to find those that work best for this application. Two profile representations are considered: the one discussed in section 3.1 and depicted in Figs. 2 and 3, in which each port is assigned a value between 0 and 100 corresponding to the percentage of connections involving the port over the course of the historical record. The other representation is the more coarse-grained binary: a simple "open" or "closed" value for each port. In what follows, the former is referred to as the *proportioned* and the latter the *binary* representation.

4.4. Partitional clustering of profiles

The approach to system classification pursued in this analysis is independent of any particular clustering scheme, and so a variety of methods are explored using the two feature representations introduced above. The first method considered is partitional clustering applied to the profiles, where each profile is a vector in a space with a dimension given by the number of ports represented across the neighborhood; in the example of Fig. 3 the dimension of the feature space is 14. In practice, the dimensionality of this space can become large; for example, the workstations in our data sample have an average 50 or so ports represented across their neighborhoods. As is well known, distance-oriented algorithms can perform poorly and counter-intuitively in high dimensional spaces. Care must therefore be taken when applying partitional clustering algorithms based on Euclidean distance, $d(x,y) = \| x - y \|$, like *k*-means or mean-shift clustering. To ameliorate the effect of high-dimensionality on clustering, there is a *spherical* *k*-means algorithm Hornik et al. (2012) in which the closeness of vectors **x** and **y** is instead measured via the *cosine similarity*,

$$d_\theta(\mathbf{x},\mathbf{y}) = \cos\theta = \frac{\mathbf{x}\cdot\mathbf{y}}{\| \mathbf{x} \| \| \mathbf{y} \|}. \tag{1}$$

The spherical *k*-means algorithm normalizes all vectors so that they are projected to points on the surface of a unit hypersphere. These points are then partitioned on the surface of the sphere according to cosine similarity; this approach, in which points are considered close if they lie along similar directions from the sphere's center, might have better high-dimensional performance than ordinary *k*-means using Euclidean distance as a similarity measure. We test both *k*-means and spherical *k*-means in this study.

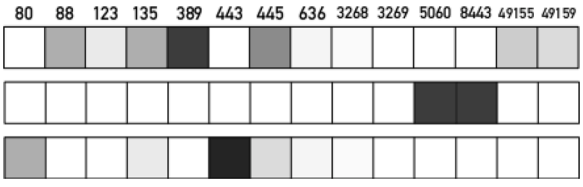

Fig. 3. A group of server profiles.

4.5. Clustering profile similarity

Rather than clustering the profile vectors directly, we can instead analyze the similarity matrix, $\mathbf{S} \in \mathbb{R}^{n\times n}$, consisting of pairwise similarities computed among all profile vectors in a neighborhood of n systems. The matrix is normalized so that similar vectors x,y have $S_{x,y} \approx 1$. For proportioned data, cosine similarity is selected to give good high-dimensional behavior; for binary data, we in addition consider the Jaccard index applied to the systems' port sets (the set of ports appearing in each system's profile),

$$J(x,y) = \frac{|p_x \cap p_y|}{|p_x \cup p_y|}, \tag{2}$$

where $p_{x,y}$ is the set of ports in profile x,y. Once the similarity matrix is in hand, a variety of clustering methods can be applied directly to it. We consider spectral and agglomerative hierarchical clustering as two conceptually distinct approaches to this problem.

Spectral clustering interprets the symmetric similarity matrix as the adjacency matrix of an undirected graph, and computes the graph Laplacian, $\mathbf{L} = \mathbf{D} - \mathbf{S}$, where the components of the degree matrix, **D**, are

$$D_{ii} = \sum_{j=1}^{n} S_{ij}. \tag{3}$$

Next, the matrix $\mathbf{V} \in \mathbb{R}^{n\times\ell}$ with columns the ℓ most-relevant eigenvectors of **L** is constructed, and *k*-means clustering is applied to the *rows* of this matrix. The relevance of eigenvectors can be ascertained by looking for the "elbow" in the plot of the corresponding eigenvalues; alternatively, the number of eigenvectors to include can be determined empirically using a quality measure of the resulting clusters. In what follows, we adopt the latter approach.

Rather than cluster on its spectrum, hierarchical clustering applies directly the similarity matrix. Agglomerative clustering begins with each datum in its own cluster, and then successively merges them into larger clusters. First, the most-similar samples are merged. A new similarity matrix is then computed based on these merged points (clusters), where

similarity scores are computed among the new clusters using one of a variety of rules: here, we employ *average linkage* in which the new similarity score between two clusters is computed as the average of the pairwise similarities of all constituent points. Clusters are continually merged until either all clusters below some similarity threshold have been merged, or, as with other clustering methods, the number of clusters can be determined empirically using a measure of cluster quality. As with spectral clustering, we adopt this latter approach.

4.6. Comparison of clustering methods

The measure of cluster quality employed in this analysis is the *silhouette score*, which is the average of the silhouette coefficients, $s(x)$, over all points, x, where

$$s(x) = \frac{b(x) - a(x)}{\max\{a(x), b(x)\}} \tag{4}$$

where, for x in cluster, C,

$$a(x) = \frac{1}{|C| - 1} \sum_{y \in C, x \neq y} d(x, y), \tag{5}$$

is the mean intra-cluster distance $d(x, y)$ between x and all other points y in the cluster, and

$$b(x) = \min_{C' \neq C} \frac{1}{|C'|} \sum_{y \in C'} d(x, y), \tag{6}$$

is the minimum mean inter-cluster distance between x and all other points y not in C. The coefficient satisfies $-1 \leq s(x) \leq 1$, with coefficients close to one indicating "tight" clusters. In practice, the number of clusters, n_C, to use in a particular method is determined by computing the silhouette score over a range of $n_C \in [1, n]$ and selecting n_C with the largest score.

We take as our test data set the "watch list" of 125 high-value systems described in Section 3. Each system on the watch list is a subject: for each subject, the peers in its neighborhood are identified over a four week period via netconn records. We then test each clustering method on the proportioned and binary versions of these profiles. For spectral and agglomerative clustering using the binary features, we construct similarity matrices using each of cosine similarity and Jaccard index; for the proportioned features, only cosine similarity is appropriate. Each test system will have a different number of peers of different types, and consequently different numbers of clusters of varying quality.

Results are presented in Fig. 4: (a) shows the average number of clusters and the average silhouette scores over all test subjects for each profile representation (proportioned and binary) for each clustering method. In Fig. 4 (b), these quantities are broken out by test subject (listed along the x-axis) for the five methods highlighted in red in (a). There are a few important things to note: first, with only a few exceptions (spectral clustering on binary features using cosine similarity and spectral clustering on proportioned features), all methods find fairly high-quality clusters (mean silhouette scores > 0.75). Second, among these successful methods, we see good breadth in the average numbers of clusters, from around 22 for k-means on proportioned features up to 32 for k-means on binary features. The fact that these methods all yield tight clusters indicates that there are natural clusterings on different scales (*i.e.* a cluster can be divided into two sub-clusters without hurting the silhouette score if that cluster stands in relation to other clusters the way its sub-clusters stand in relation to it.) This is useful in practice because it allows one to vary the resolution of system roles, which can affect the performance of the anomaly detection algorithm. Also of note, as shown in Fig. 4 (b), is that the high-quality methods track each other well, suggesting that this featurization is quite robust to clustering method.

To examine the results of clustering more closely, in Fig. 5 we present the results of clustering the peers of one test subject (a workstation). The clusters are superimposed on the peer server profile feature vectors for two different clustering approaches: (a) spherical k-means on proportioned features (with 27 clusters) and (b) spectral clustering on binary features (with 22 clusters). Most systems in the figure have been labeled according to their primary function or server role as ascertained through discussions with administrators, use of standard service ports, or hostnames. These labels serve as a putative "ground truth" against which to judge the clustering, though it is important to emphasize that not all systems with the same role will have the same behavior in terms of port usage. For example, one root DC and one primary DC have notably different port profiles than the rest of their respective groups: spherical k-means clustering on the proportioned profiles separates these systems into their own classes, while spectral clustering on the binary profiles does not. Which is preferable ultimately depends on how each affects the anomaly detection rate.

Sometimes the finer clustering achieved with spherical k-means seems *too* fine: consider how the first two groups of virtualization servers are further broken up (the first group of three systems is resolved into two clusters, the second group of three systems into three clusters). Look closely at the second virtualization group: the first two open ports have close to equivalent usage across all systems in the group; the systems

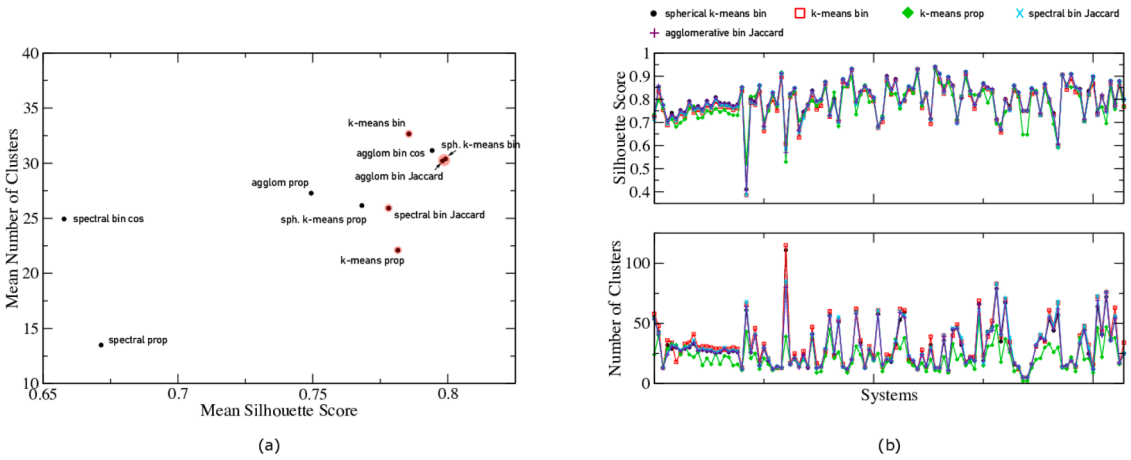

Fig. 4. Results of applying multiple clustering approaches to the neighborhoods of 125 test subjects. (a) Mean number of clusters and mean silhouette scores over all 125 subjects for each approach. Those approaches highlighted in red are considered highest-quality (largest silhouette score). See text for descriptions of the various approaches; abbreviations are as follows: "prop" refers to proportioned features, "bin" to binary features, "cos" to the use of cosine similarity in the similarity matrix, "Jaccard" to the use of the namesake index in the similarity matrix. (b) Silhouette scores and numbers of clusters found per test subject.

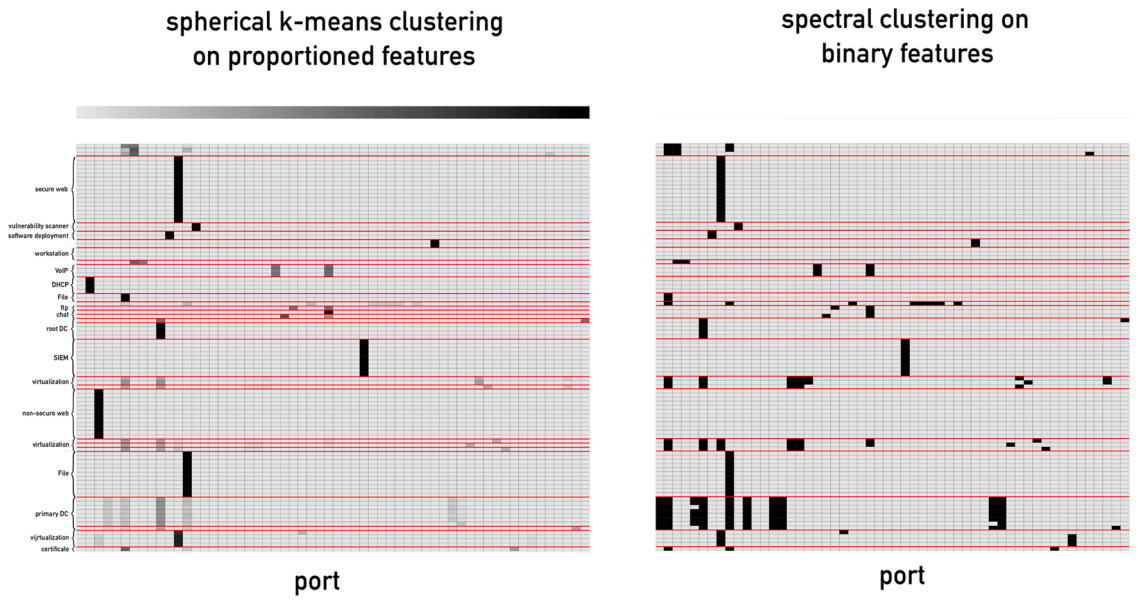

Fig. 5. Detailed results of two clustering approaches: spherical *k*-means on proportioned features and spectral clustering on binary features using Jaccard index as similarity measure, for a single test subject. Ports (unlabeled) run across the horizontal axis, peer systems run along the vertical axis. Systems are grouped according to role, and shading of boxes indicates usage. For binary features boxes are either black or white. Red lines enclose clusters found by each method.

differ only in which high-numbered ports they have open. Given that these ports have relatively low proportionate use, it is perhaps surprising that they are sufficiently important to affect the clustering. In contrast, one might naively suppose that binary clustering would be *more* sensitive to single port differences among systems, since even low-proportion ports are given the full binary value of 1, exaggerating any differences. But, examination of spectral clustering results on the binary features reveals this *not* to be the case. Indeed, both groups of virtualization servers are given clusters corresponding to their roles. While differences in open ports (however disproportionately those ports might be used) are emphasized, binary features also exaggerate the similarity of systems with open ports in common, regardless of any usage differences between them because all open ports are given equal values of 1. As long as systems have more open ports in common than not, clustering based on binary features will tend to group them together.

5. Modeling Process Dynamics

Inter-system connections relevant to lateral movement are facilitated by *processes*. By "process" we have in mind executable programs that are involved in maintaining connections and transferring data between systems. The processes comprising standard communications tend to follow patterns, and we posit that these process patterns depend on the roles of the systems involved in the communication. This is based on the premise that the role of a system determines what kinds of functions it requests or performs, and that these functions generally depend on the role of the other system involved in the connection. For example, a workstation authenticates against a Domain Controller by opening a connection to port 389 (LDAP) via the process `lsass.exe`. The Domain Controller in turn receives this connection with its own invocation of `lsass.exe`. This authentication step might be followed by additional tasks; for example, if dynamic-link libraries (DLL) need to be loaded, the process `ntoskrnl.exe` will be invoked on the Domain Controller. The combination {`lsass.exe`, `lsass.exe`, `ntoskrnl.exe`} in short succession might therefore correspond to a standard authentication operation. There are certainly variations on this theme: on networks with Windows Advanced Threat Analytics deployed, the Domain Controller immediately queries the authenticating client for threat analytics via the process `microsoft. tri.gateway.exe`, and so we might see this process sometimes included in the above sequence. Meanwhile, the interactions between the Domain Controller and an NTP server, or between two workstations, involve different processes or process patterns because the connections facilitate different functions.

These patterns of normal operations, if they can be learned, can serve as a basis for discovering malicious activity. A popular method of lateral movement, and malicious activity more broadly, is process *injection* or *hijacking*, whereby the adversary runs an illicit process under the name and process identification (PID) number of a legitimate process, or enlists a legitimate process to execute or load illicit processes or libraries on the attacker's behalf ATT&CK (2019); Hausknecht (2019). These can be very subtle techniques, especially if the legitimate processes involved are very common and executed as part of a wide range of system functions. If the legitimate processes involved in standard communications and system functions can be reliably profiled, then it becomes possible to potentially recognize illicit process injection or hijacking that alters these profiles. For example, as we've seen, `lsass.exe` frequently executes close together in time with `ntoskrnl.exe` during client authentication; if `lsass.exe` is instead used by the attacker for another purpose, we shouldn't expect to see the standard sequence of authentication-oriented processes execute on either system and this malicious connection would appear anomalous. Alternatively, lateral movement may progress via non-standard protocols (like psexec or dcom) or remote procedure calls invoked from customized `powershell` scripts following their own, possibly novel, patterns. Against the backdrop of recognizable standard process patterns, these kinds of activities are expected to stand out.

We therefore seek a means of mining patterns in process dynamics between a subject system and its peers in each role. For each subject, we then have a process model for each role, *e.g.* Domain Controllers, file shares, web servers, and so on. These models are based on historical data—several days worth in this analysis—and can be kept current by using a "rolling", fixed-length history up to the present day. The models can then be used to test new connections to peers with known roles; for example, when the subject makes a new connection to a Domain Controller, we can compare the process sequence against the connections between the subject and all other known Domain Controllers over some historical time period.

To make this tangible, Fig. 6 shows the time series of normal process activity between a workstation and four different Domain Controllers (DCs) over the same four hour period. Carbon Black Network Connect

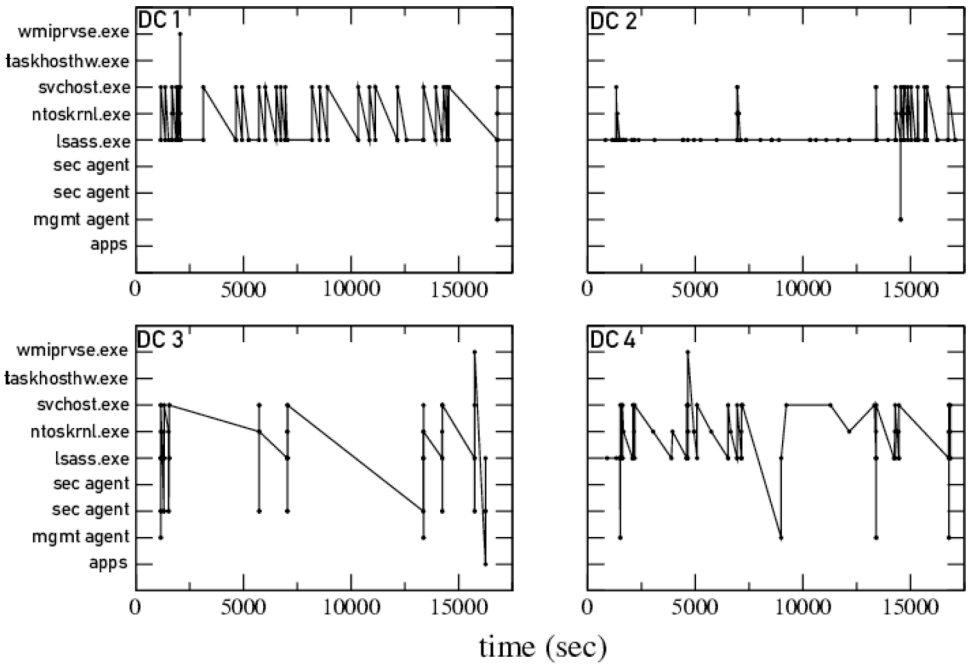

Fig. 6. Time series of inter-system communications by process type between a single workstation and four Domain Controllers.

(netconn) logs were used to create these representations, but any alternative data source that provides process names and execution timestamps will support the following analysis. Rather than work with time series, the simplest model of process dynamics ignores temporal information and considers each time series as an ordered sequence of processes. This kind of modeling is similar to the analyses of Unix command line behaviors and system call traces discussed in Section 2, and so we apply the popular method of *sequence time delay embedding* (stide) used in many of those analyses to our process time series. Stide is a method of subsequence comparison: the test data is organized into subsequences of k consecutive processes, and these subsequences are compared against a store of normal subsequences of length k. Anomaly scores are therefore applied at the subsequence level: in Hofmeyr et al. (1998) the Hamming distance between test and normal subsequences was used, and in Warrender et al. (1999) *locality frame count* (LFC) was used. The LFC is simply the number of mismatches between the test and the normal subsequences. The choice of k is arbitrary, though some authors suggest that $k = 6$ is optimal in a wide range of applications Tan and Maxion (2002). We choose $k = 6$ in this analysis, but will see that model performance does not hinge on this value.

To test a subject system with stide, we first need a collection (hereafter, *database*) of its historical connections to serve as our normal instances (this is the "model" for stide). The database is built from netconn data collected over some historical period: we fix this historical period at ten days. For each peer system, the netconn data (which can be visualized in raw form as the time series of Fig. 6) is translated into a sequence of processes. In this study we wish to test all new connections arriving in a 24-hour period as a batch process (though these methods work just as well in streaming deployments or in batches of smaller or larger duration; we select 24 hours merely for demonstration). The new records for each peer system are translated into process sequences, added to the peer's database, and then the whole database is segmented into k-length subsequences. Next, all peer systems are assigned roles via one of the clustering methods detailed in the last section. Finally, for each peer system, each k-length subsequence with new processes (called a *test subsequence*) is compared against all k-length subsequences in the historical databases of all peer systems with the same role. The number of historical records that match the test subsequence serves as a simple anomaly score (with low scores indicating more anomalous subsequences).

To give stide a thorough investigation, we use it to test the same 125 high-value systems used to test role identification in the last section. We are first and foremost interested in the distribution of scores: we would like to understand how sensitive stide is to rare test subsequences, which tells us how well stide captures the normal process behavior of each role. The z-score is computed for each subsequence, $z = (x - \mu)/\sigma$, where x is the number of matches between the test subsequence and those in the historical database with the same role, and μ and σ are the mean and standard deviation of these records.

Fig. 7 shows the cumulative distribution of the z-score, which gives the probability that a sample will satisfy $(x - \mu)/\sigma < z$.

It is apparent that stide finds many subsequences with low numbers of matches: these are subsequences with large z-scores. For example, around 15% of the subsequences lie beyond $z = 10$; that is, a little over a tenth of subsequences are *very* rare (at least 10 standard deviations from the mean). In practice, our anomaly threshold should be based on subsequence rarity, and so in order to keep the number of anomalies to a manageable level we are forced to consider only subsequences of the most extreme rarity. For example, across our 125 test subjects, there are

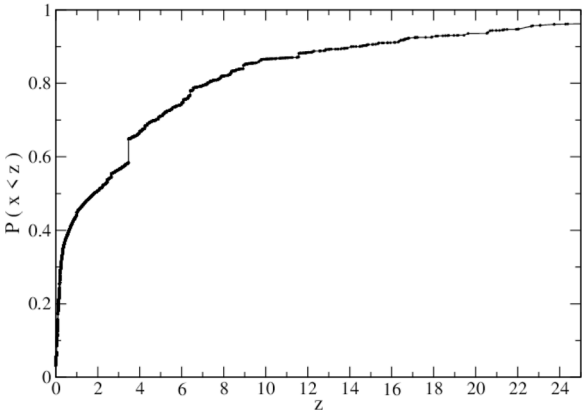

Fig. 7. Cumulative probability distribution of test sample z-scores for stide, aggregated over all 125 test subjects for one 24-hour test period.

around 140,000 test subsequences in a 24-hour period: even with a rather extreme $z = 10$ threshold on subsequence similarity, there will still be 21,000 anomalous records to investigate (around 150 per system). This is too many to be useful in any realistic defensive scenario.

One plausible reason that stide performs so poorly is that it discards all temporal information but process order: adjacent processes in the sequence could be seconds or hours apart, and stide treats them identically. It therefore mixes long- and short-timescale process dynamics, and there is unlikely to be much discernible order in such sequences. The reasoning goes as follows: as a user interacts with a computer, discrete functions are performed: authentications, DNS queries, NTP syncs, file downloads, and so on. For a given user, these events might occur roughly randomly in time (aside from possible regularities in such things as daily login times). When these functions are decomposed into their constituent processes, as we are doing here, this disorder persists and algorithms like stide struggle to identify any regularity. Recall the sample process subsequence corresponding to client authentication: {`lsass.exe`, `lsass.exe`, `ntoskrnl.exe`}. With $k = 6$, stide will never analyze this as its own subsequence, but always in combination with other potentially functionally-irrelevant processes. We therefore must find a way to first organize the longer process time series like those of Fig. 6 into more localized groups, or clusters, of processes (which are more likely to correspond to discrete system events and functions) and analyze *these groups* for anomalous behavior. Sadly, stide was doomed from the outset: we explored it merely as a cautionary tale against treating process time series as flat, fixed-size sequences.

5.1. Mining process clusters

In this section we propose a way to isolate process subsequences that might correspond to higher-level system functions. To identify them in time series like Fig. 6, we apply density-based clustering in the temporal domain to the process time series with a time threshold on the order of seconds. The algorithm DBSCAN Ester et al. (1996) creates dense clusters as follows: 1) a *core point*, which lies within a distance ϵ from at least *MinPts* other points (its ϵ-neighbors) is placed in a cluster along with its ϵ-neighbors; 2) if any of these ϵ-neighbors is a core point, all of its ϵ-neighbors are added to the cluster; 3) this process is repeated for all core points. The choice of *MinPts* and ϵ are application-specific: since we are interested in process clusters of any size, we choose[1] *MinPts* $= 2$. The appropriate choice of ϵ is less clear: one could base its value on some "natural" timescale of process dynamics, but this certainly depends on the function. Instead, we elect to set ϵ equal to the typical distance between each point in the time series and its *MinPts* nearest-neighbors[2]. In this way, ϵ is the natural distance under the assumption that the smallest clusters should have a size of *MinPts*.

Fig. 8 shows what these clusters look like (red boxes) for a sample process time series of connections between a workstation and a Domain Controller. We include the stide window with $k = 6$ for comparison, to show how it rather arbitrarily combines processes that are likely parts of different system functions. Though not visible in the plot, occasionally two or more processes will execute simultaneously (and so overlap in the time series plot). For example, the fourth cluster in Fig. 8 consists of the processes

{`svchost`, `svchost`, `lsass`, `sec`, `lsass`, `lsass`},

wherein the two `svchost` processes occur simultaneously but are destined for different ports on the Domain Controller.

The basis of this methodology is that there is generally something behaviorally relevant about these clusters, both in terms of chronology and timing. Quantitatively, this suggests we might find the same clusters appearing more than once in the history of a given system, or within the histories of systems with the same role. Conversely, rare clusters might indicate unusual process behavior related to a possible malicious connection.

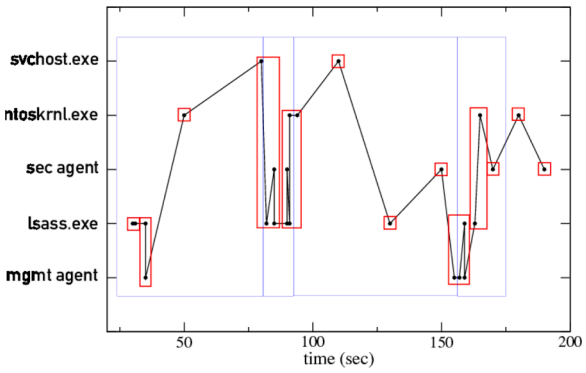

Fig. 8. Time series of inter-system communications by process type between a single client and a Domain Controller. Process clusters identified via density-based clustering in time are shown in red. The stide locality frame is shown in blue for comparison.

To test this approach, we apply it to the same 125 subjects used previously to test stide. Each subject's historical database is still organized by peer system, but instead of equal-sized subsequences, the processes are grouped into density-based clusters. The sizes of these clusters vary: across the full set of 125 test subjects, cluster sizes follow the distribution of Fig. 9. Most processes are singletons, separated in time from neighboring processes by more than ϵ seconds. Interestingly, clusters can become large; for example, several hundred clusters have 10 or more process. To test a connection between the subject and one of its peers, the new connection data is similarly clustered and compared against the historical process clusters of the other systems with the peer's role. Like stide, we count the number of matches and compute the z-score of each test sample, where here samples are process clusters instead of fixed-length subsequences.

These results are shown in Fig. 10. There is dramatic improvement over stide; for example, only 10% of clusters have z-scores exceeding 4. By clustering processes in time, we are doing a better job of capturing the regularities of the process dynamics for each system role. In essence, density-based process clusters are more *meaningful* than subsequences with fixed locality frame sizes. Another advantage of using clusters over subsequences is that there are a factor of 10 fewer of them, and so not only is there a smaller percentage of samples lying beyond a given value of z, but also fewer in absolute number.

We now look at how these process clusters are distributed for

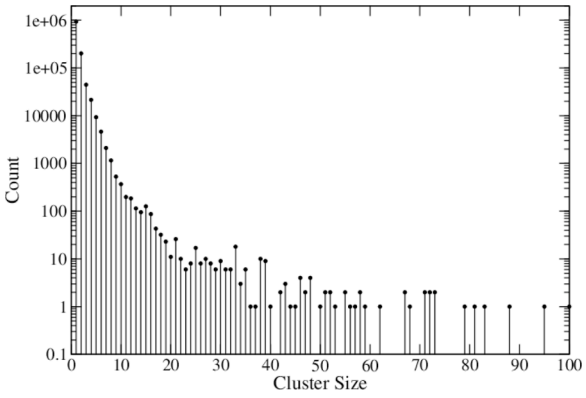

Fig. 9. Distribution of process clusters aggregated over the databases of all 125 test systems.

[1] Clusters of size 1 will still be found as noise by DBSCAN.

[2] In practice, this distance is plotted as an increasing function of points and the "elbow" of the curve is chosen for ϵ.

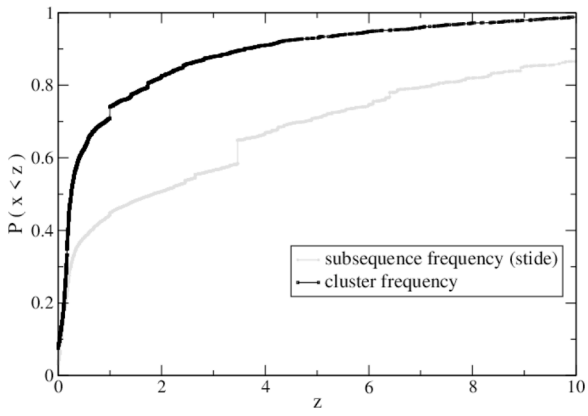

Fig. 10. Cumulative probability distribution of test sample z-scores resulting from comparison of density-based process clusters, aggregated over all 125 test subjects for one 24-hour test period (black). The results of stide, discussed previously, are provided for comparison (gray).

different system roles. In Fig. 11 we show the process cluster distributions of six roles: internal web services, file servers, root Domain Controllers, DNS servers, primary Domain Controllers, and client workstations.

Along the x-axis of each subplot are the distinct process clusters, like {`lsass`, `svchost`} and {`ntoskrnl`, `svchost`, `svchost`}, though here given number labels to conserve space. The y-axis gives their frequency of occurrence in the historical database of the role. Most roles have many rare clusters; for example, 60% (amounting to several hundred) of Primary DC clusters are unique (have a single occurrence in the database). Conversely, most roles exhibit their process dynamics in terms of only a few types of clusters; for example, for web servers, a single cluster accounts for half of all occurrences. These examples suggest an interesting claim, that relatively few clusters account for a good majority of all occurrences within a role, and that there are moderately long tales of rare clusters. The first part of this claim supports the idea that systems with the same role "act the same" with respect to processes dynamics, and furthermore that density-based clustering is able to isolate these dynamics in terms of frequently-occurring clusters. But, the second part of the claim contends that there is still considerable variability in the process dynamics of system roles, manifested in a preponderance of rare clusters. To understand this tension, we now take a closer look at these rare clusters.

If we look at the file server role, there is a single cluster that occurs only once: it is labeled '8' in the plot, but its true identity is the doublet {`ntoskrnl`, `ntoskrnl`}. The process `ntoskrnl` that comprises it happens to be the most common cluster when it occurs alone (labeled '0' in the plot, it is a singleton cluster accounting for around half of all occurrences). Now, if the DBSCAN ϵ parameter, which roughly defines the maximum time separation between processes to be considered part of the same cluster, were increased a small amount this rare doublet would separate into these more common singlets, and there would be no novel process clusters in the file server role. This observation suggests that, though we have attempted to set ϵ to an appropriate time separation for data with *MinPts* $= 2$, process clusters might not map so cleanly onto higher-level system functions; some clusters might simply be associations of unrelated processes executing close together in time. The correct way to view {`ntoskrnl`, `ntoskrnl`} is then perhaps not as some rare and significant combination of two sequential processes, but instead as two separate {`ntoskrnl`} processes that just happened to occur in close succession. We should then consider the doublet {`ntoskrnl`, `ntoskrnl`} to be essentially *as common* as the singleton {`ntoskrnl`} out of which it is built. We are therefore interested in relating the frequency of a process cluster to the frequency of its subclusters. This is essentially the problem of *frequent item-set mining* in transactional databases, which we now describe.

5.2. Finding frequent sub-clusters

The particular method we adopt here is Krimp van Leeuwen and Vreeken (2014); van Leeuwen et al. (2006); b)Siebes (Vreeken), which identifies frequent item-sets in a database as those which maximally

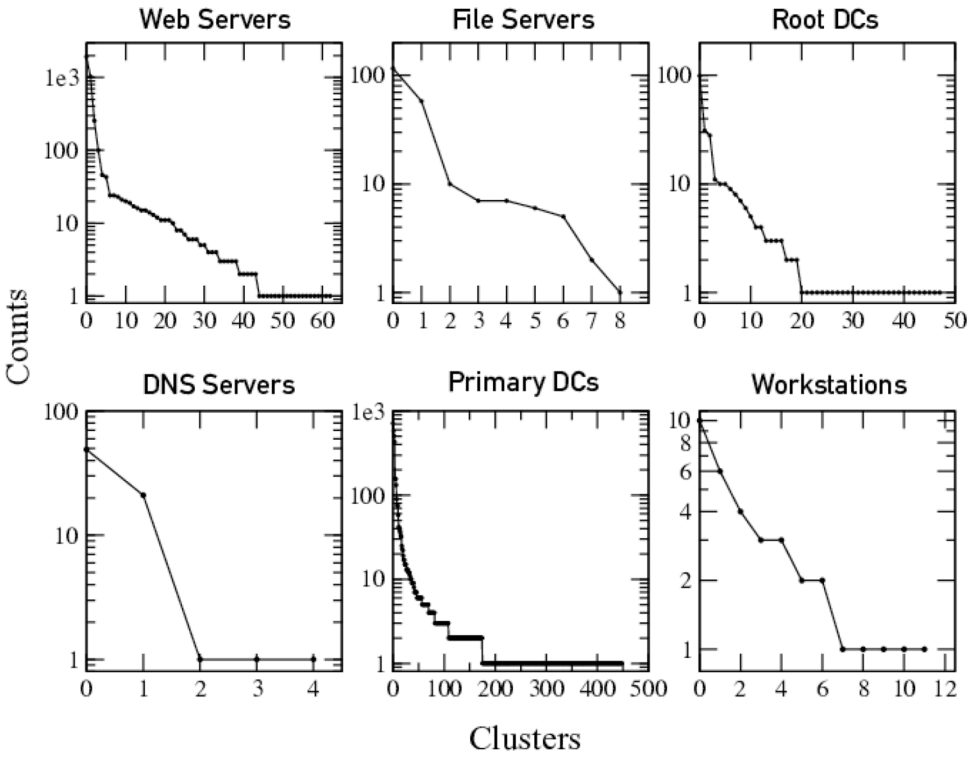

Fig. 11. Counts of distinct process clusters for different roles among a single test subject's peers, ranked from high to low.

compress it. There are other compression-based methods (see, for example, Bhattacharyya and Vreeken (2017); Fowkes and Sutton (2016); Hoang et al. (2012); Lam et al. (2014); Smets and Vreeken (2012); Tatti and Vreeken (2012) and the comprehensive survey Galbrun (2021)) but Krimp is technically and conceptually simple and it applies cleanly to our use-case. Krimp identifies frequent subsets in a database of transactions, and then uses these subsets to build compressed instances of each transaction. The length of the compressed transaction can then be used to infer its typicality: those transactions with comparatively long lengths are candidate outliers Akoglu et al. (2012); Smets and Vreeken (2011). In our application, "transactions" are process clusters and we apply Krimp ultimately in order to identify relatively incompressible clusters. Such incompressible clusters are anomalous, signalling that there is something unusual about the timing, ordering, prevalence, or type of processes that comprise it. The concept of identifying anomalies by process sequences is by no means married to this particular technique, and we consider several other common approaches from the literature later in this paper.

Krimp works as follows. Given a database, D, and set of models, $\mathscr{M}$, the model $M \in \mathscr{M}$ that minimizes the *description length*,

$$L(M) + L(D|M), \tag{7}$$

is the *optimal compressor* of the data. Krimp is a method for finding an approximation of this optimal compressor. In the following, we briefly review Krimp using the notation and terminology as presented in van Leeuwen et al. (2006); b)Siebes (Vreeken), and point out a few key modifications to the original implementation needed for our application. Krimp concerns databases built out of discrete items from a set, $\mathscr{I}$. A transaction, t, is a *sequence*[3] drawn from $\mathscr{I}$. A transaction of length n therefore belongs to the set $\mathscr{I}^n$. As a sequence, the order of t matters and items within t can be repeated. In our application items are individual processes, like `lsass.exe` or `svchost.exe`, and the set $\mathscr{I}$ is the collection of all such processes. A transaction, t, is then a process cluster. The database, D, is the collection of all process clusters over a certain time period between a given system and its peer systems within a given role.

Krimp seeks to compress the database, D, by identifying frequent item-sequences, $X \in \mathscr{I}^n$, appearing in the set of transactions[4]. The models considered by Krimp are *code tables*, CT, which are simply lists of these item-sequences along with their encodings. The optimal compressor is then the code table which leads to the shortest encoding of the database, where the length is computed as

$$L(D|CT) = \sum_{t \in D} L(t|CT). \tag{8}$$

The length of a transaction, t, is given by the lengths of the encoded item-sequences that appear in it,

$$L(t|CT) = \sum_{X \in cov(t)} L(X|CT), \tag{9}$$

where $cov(t)$ is the set of item-sequences appearing in, or *covering*, the transaction, t. The item-sequences that cover a given transaction must be disjoint (that is, each item in t must belong to only one item-sequence). Finally, the length of the item-sequence is where the compression comes in: the encoding is based on the frequency of the item-sequence in the database,

$$L(X|CT) = -\log(P(X|D)) = \frac{usage(X)}{\sum_{X' \in CT} usage(X')}. \tag{10}$$

The function $usage(X)$ counts the number of transactions with X in their covers. Krimp begins with the standard code table, which includes only item-sequences corresponding to individual items, and successively adds composite item-sequences one at a time: if the database encoding length is reduced, the item-sequence is added permanently to the code table; otherwise, it is discarded permanently. In this way, Krimp is a greedy algorithm that works to identify the collection of item-sequences that best cover the transactions in the database, that is, that lead to a shortest encoding of the entire database.

In our application, item-sequences are sub-clusters: sub-sequences of consecutive processes that make up larger clusters. We wish to apply Krimp to identify those common sub-clusters that entail an optimal compression of the collection of process clusters. Clusters that fail to compress well in comparison with the bulk of the collection are considered via this method to be anomalous.

We now apply Krimp to our 125 test subjects, with results presented in Fig. 12. Included in this figure for comparison are the z-score probabilities from the cluster frequency analysis of Fig. 10.

The larger cumulative probability at small z for the method based on cluster frequencies reveals that the distributions of z-scores under this method are more centralized than under Krimp; however, Krimp has skinnier tales. This is evident in the crossing of the distributions at around $z = 3$: Krimp has less probability mass below a given z for $z \gtrsim 3$. Since anomaly detection pertains to the tails of a distribution (large z), this is an important finding. For example, if we set an anomaly threshold at $z = 4$, Krimp labels 5% of the test samples anomalous versus 10% using cluster frequency analysis. With generally a factor of two improvement, compression based on frequent item-set mining, as demonstrated by Krimp, appears to resolve additional structure within density-based process clusters useful for understanding process dynamics.

To gain some intuition for how things improve with Krimp, we revisit the results of Fig. 11 showing the frequencies of process clusters for different system roles. In Fig. 13, we plot the length of the encoded process cluster, $L(t|CT)$, versus its frequency for the web server role as an example.

Suppose that we wish to identify only novel clusters as anomalies. In this case, there are 19 clusters that appear only once in the web server role and are anomalous according to cluster frequency. In looking at $L(t|CT)$, we see that the encoding length does not have a simple dependence on the cluster's frequency. This has the welcome effect that

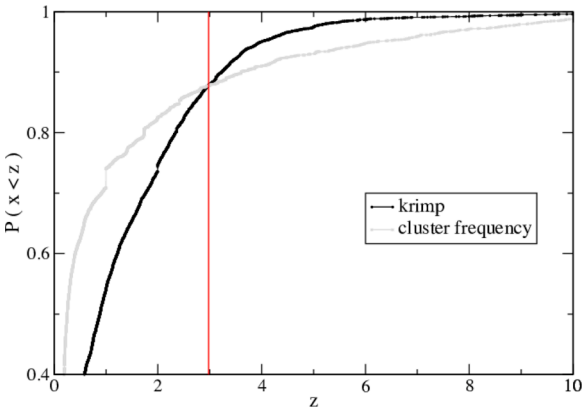

Fig. 12. Cumulative probability distribution of test sample z-scores resulting from Krimp (black) and a clusters frequencies (gray), aggregated over all 125 test systems for one 24-hour test period. The red vertical line marks where the Krimp model becomes preferable (smaller probability mass beyond the given z-score).

[3] In van Leeuwen et al. (2006); b)Siebes (Vreeken), t is a *set* and so items cannot be repeated and order is irrelevant. These constraints are inappropriate for our application, since the same process can meaningfully occur multiple times in a cluster, and chronology is important. For these reasons we define t to be a sequence.

[4] Contrast with the *itemsets*, $X \subseteq \mathscr{I}$ of van Leeuwen et al. (2006); b)Siebes (Vreeken), which, again, do not consider order or allow for repeated items.

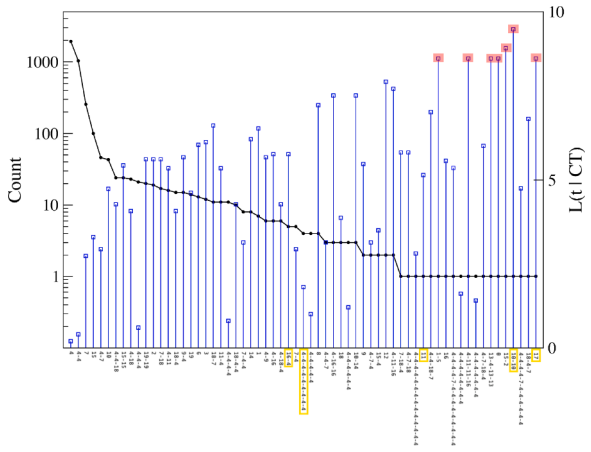

Fig. 13. Count (black dots) and encoded length (blue boxes), $L(t|CT)$, of each process cluster found for the web server role over the historical period considered in this analysis. Individual processes are numbered 0 through 19, and the process clusters are labeled along the x-axis. Red-highlighted boxes are process clusters deemed anomalous using the threshold discussed in the text. Process clusters highlighted in yellow are referenced in the text.

most of these 19 clusters have relatively small encodings and therefore look normal. We set the detection threshold of Krimp using the z-score of the novel clusters from the cluster frequency distribution (corresponding to an encoding length $L(t|CT) = 8.61$), and find that only 7 of the 19 singleton clusters are deemed anomalous, a reduction in false alarm rate of more than a factor of two. This is consistent with the more general performance comparison undertaken above (cf. Fig. 12.)

To understand this, notice that process `4` in Fig. 13 is the most prevalent process cluster and hence will have the shortest encoding as per Eq. (9). As a result, clusters built from process `4` will likewise have short encodings: for example, the large cluster of nine process `4`'s (highlighted in the figure) has a shorter encoding than *smaller* clusters that happen to appear even *more* often, like the cluster $\{16, 4\}$ (also highlighted). Another property of Krimp is that singleton clusters that appear infrequently can still have short encodings if the process finds itself in other, larger clusters (the converse of the effect seen with process `4`). For example, process `11` occurs only once as a singleton cluster, but has a shorter encoding than other once-appearing singletons, like process `17` (both highlighted). As one looks through the processes listed along the x-axis, process `11` appears repeatedly in other clusters, with the result that its encoding is shortened relative to processes like `17` which only appears as a singleton cluster.

Finally, we note a somewhat counter-intuitive result of Krimp: how is it that certain infrequent process clusters, like cluster `10-10` (highlighted in Fig. 13), have longer encodings than they would if treated as a single, novel item-sequence? Since `10-10` appears only once, we might expect that it should have an encoding length no longer than other once-appearing clusters, like process `0`, but in fact we find that it is a little longer. This is because the item-sequence $\{10\}$ has already been included as part of the standard code table, and it is more economical overall to simply build `10-10` out of this item-sequence then to introduce the item-sequence $\{10 - 10\}$. Krimp is the consummate utilitarian, doing what is best for the entire database, sometimes at the cost of longer encodings for a few transactions.

As we close this section, we summarize our journey: we started with stide, which identified anomalies in k-length subsequences according to their prevalence in the database. It performed poorly because temporal information was ignored, and consecutive processes were generally unrelated at the system function level. To recover this temporal structure, with the hopes of grouping together processes participating in the same higher-level functions, we applied density-based clustering in time. The frequency distribution of these clusters was considerably more centralized than that of stide, indicating that the clustering allows us to better model the regularities of inter-system processes within a given system role. This method was also imperfect, as happenstance occasionally prevailed to cluster together unrelated processes, contributing to the preponderance of rare clusters. To address this problem, we performed frequent item-set mining to the database of clusters to identify common substructures within these clusters, and found that the encoding length of each cluster was revealed to be a better indicator of novelty than its frequency.

6. Results and Comparison with Other Methods

In this section we present the results of applying role-based lateral movement detection to the same set of 125 test systems used throughout this paper. The data are derived from Carbon Black netconn logs and used to build 10-day histories of process activity between each subject and each of its peers. Testing is performed on the most recent 24-hours-worth of activity.

6.1. Comparison of Krimp with other methods

The analysis up to now has focused on how well Krimp is able to model the normal process activity within a given system role. In particular, we studied Krimp's sensitivity by analyzing the distribution of z-scores of encoding lengths, $L(t|CT)$, for each class. We now perform an experiment to study how well Krimp identifies anomalous clusters injected into the histories of normal process activity as follows: After resolving each subject's peers into roles, to each role we add a single process cluster chosen at random from a different, randomly selected role. Since the added cluster is not from the role, it is generally anomalous (though, in practice, different roles can have the same process clusters, particularly those involving very common processes). We test Krimp's ability to spot these anomalous processes while recognizing those rightly belonging to each role as normal.

We compare Krimp's performance with a variety of other approaches from the recent literature. We select Interesting Sequence Miner (ISM) Fowkes and Sutton (2016), which is a compression-based frequent item-set mining algorithm that shares a genealogy with Krimp, in order to test Krimp against a similar, slightly more sophisticated approach. We also compare against three unrelated methods: frequent pattern outlier factor (FPOF) He et al. (2005); a categorical data version of the local outlier factor, called κ-LOF Yu et al. (2006); and common-neighbor-based outlier factor (CNB) Li et al. (2007). Though based on frequent patterns, FPOF is non-compressive and its outlier factor is determined by a support threshold; the κ-LOF and CNB methods are density- and distance-based, respectively. Together, the four comparison models are of fundamentally different types and so offer a glimpse of how conceptually distinct outlier factors tackle our problem.

There are many dozens more methods in the relevant areas of anomaly detection in categorical data Taha and Hadi (2019), frequent pattern mining Aggarwal et al. (2014), and discrete sequences Chandola et al. (2012); Domingues et al. (2020), and it would be prohibitive to consider them all. We have selected models for comparison that are directly applicable to our problem (that can accommodate sets or sequences of varying sizes, even those containing a single element) and those with publicly available software or algorithms reasonably easy to code from scratch. For example, while the field of association rule mining offers several approaches useful for anomaly detection in discrete categorical sequences, it is not clear how best to adapt these methods to address test sequences with only a single element, a very common occurrence in our use-case. We now briefly describe each method.

The ISM is a generative model that builds a database of sequences out of a small set of interesting subsequences, $\mathscr{S}$. The database is generated by randomly interleaving these subsequences with different multiplicities. ISM learns the probability distribution, Π, of interesting sub-

sequences as follows: like Krimp, it begins with only singleton subsequences (the *standard code table* in Krimp parlance), and then it iteratively adds candidate subsequences to $\mathscr{I}$ and performs expectation-maximization to optimize the parameters of the distribution. When this process completes, a sequence, X, from the database can be encoded via Shannon's theorem with approximately $-\log_2 p(X|\Pi, \mathscr{I})$ bits. As a compression-based frequent item-sequence miner, ISM is similar to Krimp; however, Krimp uses a greedy heuristic to generate its code table, and its construction rules are simpler (multiplicity and concatenation, with no interleaving).

The frequent pattern based outlier detection of He et al. (2005) mines frequent itemsets in a database by directly appealing to their *support*: the support of an itemset, X, is the percentage of transactions, t, in the database for which $X \subseteq t$. Given a set of items, $\mathscr{I}$, the set of *frequent patterns* (FPS) are those sets $X \subseteq \mathscr{I}$ with at least s_0 support. The frequent pattern outlier factor (FPOF) of t is then defined,

$$\text{FPOF}(t) = \frac{\sum_{X \subseteq t, X \in \text{FPS}} \text{supp}(X)}{|\text{FPS}|}, \tag{11}$$

where $\text{supp}(X)$ is the support of itemset X. The FPOF of a transaction is simply the percentage of frequent patterns appearing in it, and so small scores suggest anomalies. As done with Krimp, we make obvious adjustments to adapt this method to apply to sequences instead of sets. The minimal support, s_0, is a free parameter that can be tuned to performance requirements.

The κ-LOF was conceived as a categorical version of the well-known local outlier factor Yu et al. (2006) originally devised for numerical data. The method is given an undirected graph representation, with each transaction in the database a vertex, and edges connecting vertices with a weight proportional to their similarity. The notion of similarity employed here is based on graph walks, where a *κ-walk* between two vertices t and t' is any sequence of κ edges starting at t and ending at t'. The *similarity of κ-walks* between two transactions t and t' is $s^\kappa(t,t') = \sum_{t'',t'} w(t'',t') s^{\kappa-1}(t,t'')$, where $w(t'',t')$ is the *weight* of the edge between vertices t'' and t', and $s^0 = 1$. In Yu et al. (2006), the weight is defined as the number of common categorical elements between transactions t'' and t'; since our transactions are in general different lengths, we define the weight as the number of matches relative to the length of the longer transaction. The *accumulated similarity* between t and t' is $S^\kappa(t,t') = \sum_{i=1}^{\kappa} s^i(t,t')$, and this quantity is used to define the outlier factor,

$$\kappa\text{–LOF}(t) = \frac{\text{avg}\{S^\kappa(t,t') | S^\kappa(t,t') > 0\}}{S^\kappa(t,t)} \tag{12}$$

for vertices t' reachable from t within κ-walks. The denominator is the accumulated similarity of closed walks (those that start and end on t) which acts to measure the similarity of t with its local neighborhood; meanwhile, the numerator measures the average similarity of t with vertices further away, up to a distance κ. If κ-LOF(t) is small, then t is more similar to its immediate neighbors than these others, and is not considered an outlier. Conversely, large κ-LOF(t) indicates that t is in a neighborhood with vertices more similar to each other than they are to t. This makes t anomalous according to the paradigm of the density-based LOF. One important aspect of this method is that no account is taken of how frequently a transaction appears in the database. This would need to be incorporated as a vertex attribute of some sort, but does not appear to be considered in Yu et al. (2006). As we will see, this degrades performance on our database of clusters, for which frequency of a cluster (or its sub-clusters) is an essential aspect of its novelty.

Finally, the CNB method computes the "distance" between a transaction t and all others, and defines an outlier factor based on the distance to t's k^{th}-nearest neighbor. The method begins with a notion of similarity, defined in Li et al. (2007) as the number matches between equal-length transactions. Because our transactions are of variable length, we base similarity on common subsequences: the similarity between transactions t and t' is defined as the average length of the largest closed common subsequences relative to the length of the longer transaction. For example, for $t = (3,4,1,2)$ and $t' = (1,2,3,4)$, the two largest closed common subsequences are (1,2) and (3,4), with an average length of 2. Following Li et al. (2007), the similarity measure is used to construct the neighbor set of t, $NS(t)$, including all transactions t' with a similarity to t greater than some threshold, θ. The common neighbor set, CNS, between two transactions t and t' is then defined

$$\text{CNS}(t,t',\theta) = NS(t,\theta) \cap NS(t',\theta). \tag{13}$$

The distance between t and t' is

$$d(t,t') = 1 - \frac{\log_2 |\text{CNS}(t,t',\theta)|}{\log_2 |\mathscr{D}|}, \tag{14}$$

where $\mathscr{D}$ is the database and vertical bars denote cardinality. This distance has a simple interpretation: two transactions are close-together if they have many neighbors in common; in our problem, two process clusters are closer together the more sub-clusters they have in common. The outlier factor is then sum of the distances between t and its k-nearest neighbors. The CNB method includes two free parameters: the similarity threshold, θ, and k. Like κ-LOF, this method also does not take into account transaction frequency when assessing novelty.

We plot the receiver operating characteristic (ROC) curves of each model in Fig. 14. For methods with free parameters (FPOF, κ-LOF, and CNB), we performed a grid search and report results for the model with the largest area under curve (AUC).

FPOF is sensitive to the support threshold, s_0: as s_0 is increased, novelty becomes more commonplace as only the most frequent itemsets are included in FPS. We find that AUC is greatest with $s_0 = 3$. The κ-LOF has the greatest time complexity, $\mathscr{O}(n^2(q+\kappa))$ for n transactions and q within κ-walks of transaction t: with $\kappa > 2$, the time-performance trade-off tips heavily against this method. For CND, AUC is best for $\theta = 0.25$ and $k = 2$.

In terms of absolute numbers, there are on average 57,779 true negative samples per day across all 125 subjects, and we injected a total of 2898 true positive anomalous clusters (one for each system role among the peers of each subject). To compare the different methods, we consider a variety of model summary statistics. The area under the ROC curve (AUC) is a common measure of overall performance; however, it takes into account performance at undesirable thresholds (high false positive rates). Alternatively, there are a range of metrics that apply at a single threshold. Because of the imbalance in the numbers of true positives and negatives, metrics like the F-score are difficult to interpret.

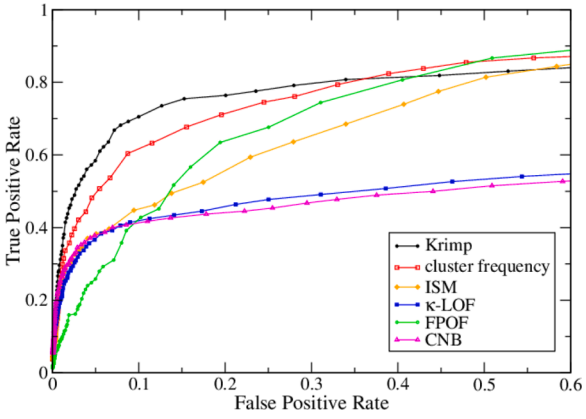

Fig. 14. ROC curves for Krimp, cluster frequency, interesting sequence miner (ISM), κ-local outlier factor, frequent pattern outlier factor (FPOF), and common neighbor-based outlier factor (CND).

Instead, we evaluate the Matthews correlation coefficient (MCC), which takes into account the number of true negatives and so is robust to this imbalance,

$$\text{MCC} = \frac{TP \times TN - FP \times FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}, . \tag{15}$$

where T, F, P, and N stand for "true", "false", "positive", and "negative". Finally, in between the "global" AUC and "local" MCC measures, we consider the *partial* AUC introduced in McClish (1989); Thompson and Zucchini (1989), which is simply the area under the ROC curve, $r(x)$, within some range of false positive rates (FPR),

$$p\text{AUC}(x_1, x_2) = \int_{x_1}^{x_2} r(x)\text{d}x. \tag{16}$$

If normalized by the range $\Delta x = x_2 - x_1$, this becomes the average sensitivity of the model, written $\widetilde{p\text{AUC}}$. This metric was studied in Carrington et al. (2021) along with several others as a means of finding a "middle ground" between traditional local and global measures. Values for these different metrics for each method are provided in Table 1. Krimp is best-performing, but this comparison tells use very little about how well role-based detection actually works in the operational environment.

The selection of thresholds for the MCC and $\widetilde{p\text{AUC}}$ depends very much on the nature of the deployment; in particular, the volume of alerts that can be tolerated. In the one-class or anomaly detection setting, thresholds are typically determined by the number of permissible false positives. With on average of 57,779 test cases per day, even low FPR thresholds like 1% will result in hundreds of false positives. And, to achieve decent true positive rates (TPR) we need even higher FPR thresholds; for example, a 70% TPR requires an FPR threshold of 10%, which will result in thousands of daily alerts, likely too many to make feasible any kind of "per alert" incident response.

We therefore recommend that role-based anomaly detection be incorporated into a larger-scale correlation analysis, in which data from multiple sensors are aggregated to identify common indicators. Indicators implicated by multiple sensors (of which role-based detection would be just one) constitute alerts with the highest risk. Systems like HOLMES Milajerdi et al. (2019) provide this capability. With these technologies, higher volumes of alerts are actually desirable because they increase the chances of a real detection, and those alerts which fail to correlate with others are simply ignored after a short while. In Table 1, we therefore evaluate the MCC at FPR = 10% and the $\widetilde{p\text{AUC}}$ between 5% and 10%.

6.2. Results of testing novel roles

The foregoing analysis was applicable to connections made to peer systems with *known* roles. As discussed in the Introduction, the process dynamics of connections to new peers with *novel* roles cannot be tested due to a lack of history; however, these connections should still be viewed with suspicion because they indicate departures from the subject's established peer role associations.

As a final portion of our analysis, we therefore test how often our 125 test subjects make connections to new peers with novel roles over the course of a typical 24-hour test period. New peers with novel roles are identified in the clustering step as systems existing in their own clusters, *i.e.* they are unlike any known peer. Since the different clustering methods explored in Section 3 have varying degrees of sensitivity, some will identify novelty while others do not; hence, in the operational setting they will generate different volumes of alerts. We therefore consider the 5 approaches highlighted in Fig. 4: 1) spherical k-means on proportioned features, 2) k-means on proportioned features, 3) k-means on binary features, 4) agglomerative clustering on binary features with Jaccard index, and 5) spectral clustering on binary features with Jaccard index. As before, we collected all netconn records of internal connections between these systems and their peers, but this time we vary the length of the historical record and study its effect on performance. There tend to be more roles in neighborhoods spanning longer time periods, and so we might expect the number of alerts per monitored system to drop as the duration of the historical is increased. Fig. 15 presents the number of anomalies identified (novel peer systems with novel roles over the most recent 24-hour period), aggregated over all 125 subjects, as the size of the historical record of each system is increased from 10 to 30 days.

In keeping with earlier results, k-means on binary features, with its propensity for numerous clusters, results in the most novel clusters and hence most anomalies; it behaves similarly to agglomerative clustering on binary features using the Jaccard index as similarity measure. Meanwhile, spherical k-means and k-means applied to the proportioned features perform comparatively, indicating that the dimensionality of the feature space is not so large as to benefit from the spherical k-means algorithm. Ultimately, these results indicate that stability tends to be reached for each of the clustering methods after around 15 days of history. With 30 alerts per day on average over all 125 subjects, k-means clustering on proportioned features yields a manageable number of alerts, with k-means on binary features offering a more sensitive alternative.

6.3. Performance

A single virtual machine with dual Xeon processor and 8GB of RAM was used to benchmark performance. Testing a single system involves two separate computations: the clustering of the test system's peers into roles, and the application of Krimp (or alternative outlier detection scheme) to the process sequences associated with all of the test system's new connections. The performance of the clustering step depends on the number of peers (the number of points to cluster) and the dimensionality of the feature space, here determined by the number of service ports in use across the set of peer systems. Each test system has on average 142

Table 1
Model comparison measures applied to each method discussed in the text.

Method	AUC	MCC FPR = 0.1	$\widetilde{p\text{AUC}}$ FPR $\in (0.05, 0.1)$
Krimp	0.81	0.38	0.66
Frequency	0.79	0.33	0.56
ISM	0.735	0.23	0.42
κ-LOF	0.58	0.2	0.4
FPOF	0.77	0.2	0.36
CNB	0.57	0.2	0.42

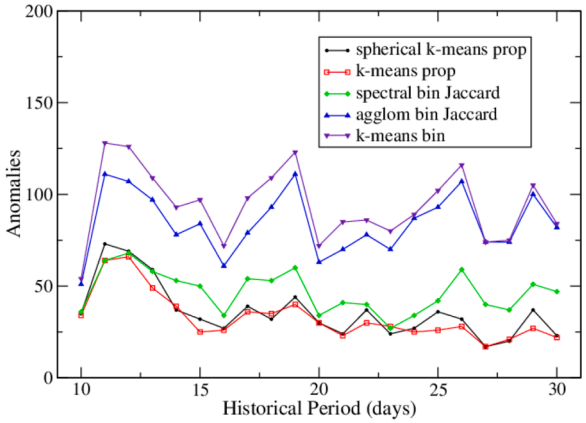

Fig. 15. Number of anomalies found across 125 test systems as a function of length of the historical period for each of the five clustering approaches highlighted in red in Fig. 4.

peers with between around 50 and 75 dimensional feature spaces. With an average of 25 clusters per test system, an algorithm like *k*-means can perform clustering in a few seconds, even using relative validity criteria like silhouette scores that must be evaluated multiple times over a range possible cluster numbers. Our network, and for sure other large enterprise networks like it, have a relatively small number of exceptionally busy systems like Domain Controllers and Exchange Servers. These systems have thousands of peers of many different types: the Domain Controllers in our test set had on average of 8000 peers organized across 300 roles. To speed up clustering for these systems, we test cluster numbers in a small window around the number found clusters from the previous day; for example, if 275 clusters were found yesterday, for today's test we consider a range of cluster numbers between 250-300, greatly limiting the number of trial partitions that are guaranteed to be sub-optimal. With this scheme, these exceptional systems can be analyzed in around 30 minutes on a single processor, and constitute that great majority of time spent performing clustering on the test systems.

To operationalize this framework with either a quicker test cycle or on a network with many high-volume systems, performance must be improved. Obvious steps might include deploying this analytic across multiple systems, as it is imminently parallelizable (each system tests a subset of the watch-list). Additionally, more efficient clustering algorithms could be explored, like rank-constrained spectral clustering Li et al. (2018a,b), which also can efficiently handle high-dimensional data like Luo et al. (2018a,b), or online *k*-means Charikar et al. (2004) which saves one from computing the same (or similar) clusters every test day. Lastly, the silhouette score has $\mathcal{O}(N^2 \times d)$ complexity (where N is the number of data points and d the dimensionality); more computationally efficient relative cluster validity measures that are linear in N are available Vendramin et al. (2010).

The second step of the framework is the testing of process sequences for outliers. Over a 24-hour period, the systems on our watch list make connections to an average of 38 other unique systems involving 870 processes. Krimp works in a matter of seconds on these systems to identify outliers. As with clustering, the trouble tends to be with the chattier systems, like Domain Controllers. For these systems the Krimp code tables can become large and the analysis can take upwards of 30 minutes on our test virtual machine. The Krimp analysis could be sped up if it were possible to iteratively update code tables, rather than having to create them anew each test day; however, at present such a method does not seem to exist. Performance would also improve if the historical record is shortened: reducing the period from 10 to 5 days, for example, would cut run times in half (since compute times are proportional to the number of processes analyzed, which are roughly constant from day to day for most systems).

In total, it takes around 12 hours to test our 125 systems on our single virtual machine. Testing every 24-hours thus gives a maximum detection latency of 36 hours. As this performance was acceptable given our rate of testing, we did not explore possible speed-ups discussed above. But, for example, by adding a second virtual machine and testing every 12 instead of 24 hours, detection latency could be reduced to 18 hours. In general, the more often testing is performed, the shorter the incident response time.

7. Security and Limitations of Role-based Detection

In this section we discuss general limitations of, and whether and how the adversary might be able to fool or circumvent role-based lateral movement detection. We assume the adversary has full knowledge of the detection algorithms and knows that systems of interest are being monitored. The key questions we address are 1) what information does the adversary need and 2) what actions must they take, to defeat role-based detection. We then comment on the feasibility of success.

Role-based detection applies to generic connections to novel peers and connections to known peers that exhibit unusual process dynamics. We consider each in turn. We assume that the adversary has compromised the subject system and wishes to move laterally. Before the adversary can know whether a planned connection will be to a peer with a novel role, they must first know the roles of each of the subject system's peers, as well as the role of the system to which they'd like to move. If the adversary has the time, they could passively monitor all connections between the subject and its peers; however, without access to these systems or their network connection logs, they will be unable to ascertain their roles. On some networks, though, hostnames can be suggestive of role (*e.g.* Domain Controllers with "dc" in their names) and in this case it might be possible for the adversary to gain some knowledge of system role simply by hostname or other indicator. Assuming they do succeed in accurately assigning each peer to a role and determine that the planned lateral move will be to a system with a novel type, they could 1) perform some activities that would make this peer's role known to the subject but without tipping off network defenders, 2) attempt to access the target system from a different subject for which the target's role is known. With the first option, any connection to the novel peer, however benign, will generate an alert; but, lacking further indicators of compromise it is possible the alert won't be considered high risk. After some time, the adversary could then perform the lateral move or other exploitation against the target, which is now of a known role. The second option is perhaps the safest, but requires access to additional systems that have a decent chance of knowing the target's role. Ultimately, it appears difficult for the adversary to prepare the environment for an illegal lateral move; the best they can likely do is learn enough about the system and its peers to make connections within the bounds of the detector, but there is no reason *a priori* to suppose that this strategy will pay off.

Next, we consider adversarial connections from a compromised subject to a system with a known role. In this case, it is no longer necessary to learn the roles of all the subject's peers (because the novelty of the target system's role is not in question), only the role of the target system. With this information, the adversary can hope to "shape" the communication to align with the process patterns expected of connections to systems of the target's role. For example, if the adversary wishes to inject code into `lsass.exe` and use it to move laterally to a file server, they must know in what contexts `lsass.exe` would be establishing connections to file servers under normal operations, including what other processes would execute as part of this event. The easiest way to do this is to simply wait for the process in question to open a normal connection to the target, and to inject code at that time (via so-called hooks, mapping the DLL to the process via `CreateRemoteThread`, or even writing the code directly into process memory). If the adversary has the time, and if connections to the target system are not unduly rare, the adversary could rather passively "ride" a normal connection in this way. But, if the adversary must actively initiate the connection, then things become more complicated if additional processes must also be manipulated so as to recreate a normal-presenting process cluster. The success of this maneuver is likely if the adversary has SYSTEM-level access to the subject.

In summary, it is difficult to evade detection of connections to systems with novel roles, and feasible to make malicious connections to systems of known roles assuming the adversary has sufficient time, access, and the knowledge required to craft benign-looking process sequences within which to hide their communications. The lesson here is that no single sensor can protect networks and systems against a determined and well-resourced adversary.

We next discuss various limitations of this method. The anomalies detected by this method correspond to temporal groupings of processes, with no further insight into the user or thread responsible for the connection. On systems with multiple users or threads, it is possible that simultaneous connections are opened between two systems, with the result that process time series like Fig.s 6 and 8 are a mix of two or more separate process sequences. Absent user- or thread-level information, these must be analyzed together, as is. If a common occurrence, simultaneous connections could contribute significantly to cluster novelty

within certain roles and possibly generate an excessive number of false positives.

Role-based detection was developed for, and tested on, a traditional network with physical routers and switches. Meanwhile, software-defined networking (SDN) enables the configuration of virtual networks, which can be modified with relative ease at will. As with other innovations, like zero-trust networking, such reconfigurations could alter a system's identifiers, like IP address and hostname, or even its port profile. Since role-based lateral movement detection models systems by role (not identifier), it would be unaffected by modifications of the first kind, but would be susceptible to changes in port profile. For example, if different Domain Controllers on the network were configured to use different ports for the same process (*e.g.* Kerberos authentication), the clustering process described here would obviously fail to group these systems together into the same function role. SDN has additional implications for this methodology with regard to data collection: while this study has made use of centrally-deposited network connection logs collected from end-point sensors, the administrative interface of many SDN deployments provide considerable insight into network flows, including port and protocol use. It is possible that SDN-collected networking data could be used as the primary data source for performing role analysis, though would be unlikely to easily support the modeling the process dynamics, which would require payload inspection of traffic.

A further limitation of this approach involves the timeliness of alerts: as presented, this framework is batch-oriented, in that historical data must be analyzed along with daily activities. While in principle new connections could be tested as they occur, this would require in practice a substantial speed-up in computations, making use of on-line or incremental clustering and categorical outlier detection. As presented, there is at most a 36-hour incident response turn around time which limits how quickly alerts can be addressed.

8. Conclusions

In this paper, we present an unsupervised approach to lateral movement detection on enterprise networks that makes essential use of the concept of system *role*. Over the course of normal operations, systems tend to make connections to remote systems of a small and stable set of roles, such that connections to systems of novel roles can be identified and investigated as potential lateral movement. Furthermore, the processes that underlie these connections follow temporal patterns based on the roles of the systems involved in the connection, such that deviations from these expected patterns signal anomalous activities.

This approach shows promise on large, operational enterprise networks. When tested on a sample of 125 test systems, the number of connections with anomalous process patterns is on the order of several thousand for detection rates of around 70%. This volume is sufficiently high to necessitate the use of correlation analysis that only promotes anomalous connections to "alert status" if additional sensors implicate relevant indicators. The specific prototype presented here was intended to work on a small group of high-valued systems, but it can in principle be extended to entire networks. In this event, the various improvements listed in Section 6.1, like parallelization and algorithmic adjustments, will be necessary.

This work might be extended in a number of ways. As presented here, it is entirely unsupervised; however, it might be possible to incorporate one-class learning into role-based anomaly detection by training a model on each role just once. There would be a single model for each role across the entire watch list (rather than roles being found anew for each subject during each test period). By foregoing the unsupervised clustering step, this method could be considerably sped up. The model could also be matured and updated over time, potentially improving accuracy over unsupervised methods.

Role-based anomaly detection could be extended by incorporating additional traffic characteristics, like bytes transferred per connection. Thus, in addition to process patterns, normal system functions could be characterized by typical data transfer rates. This would require correlating a data source with this traffic information, like Netflow, with process data.

Lateral movement via authorized means is a common tactic of advanced threats, and its detection remains a great challenge to the organizations they target. Cyber defense must move beyond rule sets and signature-based detection in order to resist these threats, and we hope this framework, which leverages common data and uses standard algorithms, can be incorporated into the defense-in-depth of the vulnerable networks in the cross-hairs of the relentless and worthy adversary.

Author Statement for submission JISAS-D-21-00758

As sole author, Brian A. Powell was responsible for all aspects/roles of the manuscript "Role-based lateral movement detection with unsupervised learning"

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

Aggarwal, C. C., Bhuiyan, M. A., & Hasan, M. A. (2014). *Frequent pattern mining algorithms: A survey* (pp. 19–64)). Cham: Springer International Publishing.

Ahmad, Z., Shahid Khan, A., Wai Shiang, C., Abdullah, J., & Ahmad, F. (2021). Network intrusion detection system: A systematic study of machine learning and deep learning approaches. *Transactions on Emerging Telecommunications Technologies, 32* (1), e4150. https://doi.org/10.1002/ett.4150

Ahmed, U., & Masood, A. (2009). Host based intrusion detection using rbf neural networks. *2009 international conference on emerging technologies* (pp. 48–51). https://doi.org/10.1109/ICET.2009.5353204

Akoglu, L., Tong, H., Vreeken, J., & Faloutsos, C. (2012). *Fast and reliable anomaly detection in categorical data* (pp. 415–424). https://doi.org/10.1145/2396761.2396816

Apruzzese, G., Pierazzi, F., Colajanni, M., & Marchetti, M. (2020). Detection and threat prioritization of pivoting attacks in large networks. *IEEE Transactions on Emerging Topics in Computing, 8*(2), 404–415. https://doi.org/10.1109/TETC.2017.2764885

ATT&CK, M. (2019). Lateral movement. https://attack.mitre.org/tactics/TA0008/.

ATT&CK, M. (2021). Process injection. https://attack.mitre.org/techniques/T1055/.

Bai, T., Bian, H., Daya, A. A., Salahuddin, M. A., Limam, N., & Boutaba, R. (2019). A machine learning approach for rdp-based lateral movement detection. *2019 ieee 44th conference on local computer networks (lcn)* (pp. 242–245). https://doi.org/10.1109/LCN44214.2019.8990853

Balajinath, B., & Raghavan, S. (2001). Intrusion detection through learning behavior model. *Computer Communications, 24*(12), 1202–1212. https://doi.org/10.1016/S0140-3664(00)00364-9

Bertacchini, M., & Fierens, P. I. (2009). A survey on masquerader detection approaches. *Cibsi, retrieved august 25, 2011, from: http://www.criptored.upm.es/cibsi/cibsi2009/docs/papers/ cibsi-dia2-sesion5(2).pdf*.

Bhattacharyya, A., & Vreeken, J. (2017). Efficiently summarising event sequences with rich interleaving patterns. *Proceedings of the siam international conference on data mining (sdm)*. SIAM. https://publications.cispa.saarland/1291/

Bhuyan, M. H., Bhattacharyya, D. K., & Kalita, J. K. (2014). Network anomaly detection: Methods, systems and tools. *IEEE Communications Surveys Tutorials, 16*(1), 303–336. https://doi.org/10.1109/SURV.2013.052213.00046

Bian, H., Bai, T., Salahuddin, M. A., Limam, N., Daya, A. A., & Boutaba, R. (2019). Host in danger? detecting network intrusions from authentication logs. *2019 15th international conference on network and service management (cnsm)* (pp. 1–9).

Bohara, A., Noureddine, M. A., Fawaz, A., & Sanders, W. H. (2017). An unsupervised multi-detector approach for identifying malicious lateral movement. *2017 ieee 36th symposium on reliable distributed systems (srds)* (pp. 224–233). https://doi.org/10.1109/SRDS.2017.31

Bowman, B., Laprade, C., Ji, Y., & Huang, H. H. (2020). Detecting lateral movement in enterprise computer networks with unsupervised graph AI. *23rd international symposium on research in attacks, intrusions and defenses (RAID 2020)* (pp. 257–268). San Sebastian: USENIX Association.https://www.usenix.org/conference/raid2020/presentation/bowman

Bridges, R. A., Glass-Vanderlan, T. R., Iannacone, M. D., Vincent, M. S., & Chen, Q. G. (2019). A survey of intrusion detection systems leveraging host data. *ACM Comput. Surv., 52*(6). https://doi.org/10.1145/3344382

Buczak, A. L., & Guven, E. (2016). A survey of data mining and machine learning methods for cyber security intrusion detection. *IEEE Communications Surveys Tutorials, 18*(2), 1153–1176. https://doi.org/10.1109/COMST.2015.2494502

Cabrera, J. A. B. D., Lewis, L., & Mehra, R. K. (2001). Detection and classification of intrusions and faults using sequences of system calls. *SIGMOD Rec., 30*(4), 25–34.10.1145/604264.604269

Carbon Black 2019 Global Threat Report. (2019).

Carrington, A. M., Manuel, D. G., Fieguth, P. W., Ramsay, T., Osmani, V., Wernly, B., Bennett, C., Hawken, S., McInnes, M., Magwood, O., Sheikh, Y., & Holzinger, A. (2021). Deep roc analysis and auc as balanced average accuracy to improve model selection, understanding and interpretation.

Chandola, V., Banerjee, A., & Kumar, V. (2012). Anomaly detection for discrete sequences: A survey. *IEEE Transactions on Knowledge and Data Engineering, 24*(5), 823–839. https://doi.org/10.1109/TKDE.2010.235

Charikar, M., Chekuri, C., Feder, T., & Motwani, R. (2004). Incremental clustering and dynamic information retrieval. *SIAM Journal of Computing, 33*(6), 1417–1440.

Chen, L., & Dong, G. (2006). Masquerader detection using oclep: One-class classification using length statistics of emerging patterns. *2006 seventh international conference on web-age information management workshops*. https://doi.org/10.1109/WAIMW.2006.195–5

Chen, M., Yao, Y., Liu, J., Jiang, B., Su, L., & Lu, Z. (2018). A novel approach for identifying lateral movement attacks based on network embedding. *2018 ieee intl conf on parallel distributed processing with applications, ubiquitous computing communications, big data cloud computing, social computing networking, sustainable computing communications (ispa/iucc/bdcloud/socialcom/sustaincom)* (pp. 708–715). https://doi.org/10.1109/BDCloud.2018.00107

Chen, W.-H., Hsu, S.-H., & Shen, H.-P. (2005). Application of svm and ann for intrusion detection. *Computers & Operations Research, 32*(10), 2617–2634. https://doi.org/10.1016/j.cor.2004.03.019Applications of Neural Networks

Chen, Y., Nyemba, S., Zhang, W., & Malin, B. (2012). Specializing network analysis to detect anomalous insider actions. *Secur. Inform., 1*(5), 1–24.

Cohen, W. W. (1995). Fast effective rule induction. *Machine learning proceedings 1995* (pp. 115–123). San Francisco (CA): Morgan Kaufmann. https://doi.org/10.1016/B978-1-55860-377-6.50023-2

Creech, G., & Hu, J. (2014). A semantic approach to host-based intrusion detection systems using contiguousand discontiguous system call patterns. *IEEE Transactions on Computers, 63*(4), 807–819. https://doi.org/10.1109/TC.2013.13

Cybersecurity, & Agency, I. S. (2021). Cisa analysis: Fy2020 risk and vulnerability assessments.

Davison, B., & Hirsh, H. (1998). Predicting sequences of user actions. *Proc. 1998 AAAI/ICML Workshop on Predicting the Future : AI Approaches to Time-Series Analysis*, 5–12.

Dewaele, G., Himura, Y., Borgnat, P., Fukuda, K., Abry, P., Michel, O., Fontugne, R., Cho, K., & Esaki, H. (2010). Unsupervised host behavior classification from connection patterns. *International Journal of Network Management, 20*(5), 317–337. https://doi.org/10.1002/nem.750

Djidjev, H., Sandine, G., Storlie, C., & Wiel, S. V. (2011). Graph based statistical analysis of network traffic. In *MLG '11*.

Domingues, R., Michiardi, P., Barlet, J., & Filippone, M. (2020). A comparative evaluation of novelty detection algorithms for discrete sequences. *Artif Intell Rev, 53*, 3787–3812.

Drašar, M., Vizváry, M., & Vykopal, J. (2014). Similarity as a central approach to flow-based anomaly detection. *Netw., 24*(4), 318–336. https://doi.org/10.1002/nem.1867

Eberle, W., & Holder, L. (2009). Graph-based approaches to insider threat detection, . In *CSIIRW '09Proceedings of the 5th annual workshop on cyber security and information intelligence research: Cyber security and information intelligence challenges and strategies*. New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/1558607.1558658

Eberle, W., Holder, L., & Graves, J. (2010). Insider threat detection using a graph-based approach. *Journal of Applied Security Research, 6*. https://doi.org/10.1080/19361610.2011.529413

El Masri, A., Wechsler, H., Likarish, P., & Kang, B. B. (2014). Identifying users with application-specific command streams. *2014 twelfth annual international conference on privacy, security and trust* (pp. 232–238). https://doi.org/10.1109/PST.2014.6890944

Erman, J., Arlitt, M., & Mahanti, A. (2006). Traffic classification using clustering algorithms, . In *MineNet '06Proceedings of the 2006 sigcomm workshop on mining network data* (pp. 281–286). New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/1162678.1162679

ESKIN, E. (2001). Modeling system calls for intrusion detection with dynamic window sizes. *Proc. DARPA Information Survivability Conference and Exposition (DISCEX 2001), Anaheim, USA*.https://ci.nii.ac.jp/naid/10019460066/en/

Ester, M., Kriegel, H.-P., Sander, J., & Xu, X. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise, . In *KDD'96Proceedings of the second international conference on knowledge discovery and data mining* (pp. 226–231). AAAI Press.

Fawaz, A., Bohara, A., Cheh, C., & Sanders, W. H. (2016). Lateral movement detection using distributed data fusion. *2016 ieee 35th symposium on reliable distributed systems (srds)* (pp. 21–30). https://doi.org/10.1109/SRDS.2016.014

Forrest, S., Hofmeyr, S. A., Somayaji, A., & Longstaff, T. A. (1996). A sense of self for unix processes, . In *SP '96Proceedings of the 1996 ieee symposium on security and privacy* (p. 120). USA: IEEE Computer Society.

Fowkes, J., & Sutton, C. (2016). A subsequence interleaving model for sequential pattern mining, . In *KDD '16Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining* (pp. 835–844). New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/2939672.2939787

Galbrun, E. (2021). The minimum description length principle for pattern mining: A survey.

Gamachchi, A., & Boztas, S. (2017). Insider threat detection through attributed graph clustering. *2017 ieee trustcom/bigdatase/icess* (pp. 112–119).

Gamachchi, A., Sun, L., & Boztas, S. (2017). A graph based framework for malicious insider threat detection. *50th hawaii international conference on system sciences (hicss)*.

Garg, Rahalkar, Upadhyaya, & Kwiat. (2006). Profiling users in gui based systems for masquerade detection. *2006 ieee information assurance workshop* (pp. 48–54). https://doi.org/10.1109/IAW.2006.1652076

Ghosh, A., Schwartzbard, A., & Schatz, M. (1999). Learning program behavior profiles for intrusion detection. *1st workshop on intrusion detection and network monitoring (ID 99)*. Santa Clara, CA: USENIX Association. https://www.usenix.org/conference/id-99/learning-program-behavior-profiles-intrusion-detection

Goodman, E., Ingram, J., Martin, S., & Grunwald, D. (2015). Using bipartite anomaly features for cyber security applications. *2015 ieee 14th international conference on machine learning and applications (icmla)* (pp. 301–306). https://doi.org/10.1109/ICMLA.2015.69

Hagemann, T., & Katsarou, K. (2020). *A systematic review on anomaly detection for cloud computing environments* (pp. 83–96)). New York, NY, USA: Association for Computing Machinery.https://doi.org/10.1145/3442536.3442550

Han, S.-J., Kim, K.-J., & Cho, S.-B. (2004). Evolutionary learning program's behavior in neural networks for anomaly detection. *Neural information processing* (pp. 236–241). Berlin, Heidelberg: Springer Berlin Heidelberg.

Hausknecht, R. (2019). Offensive lateral movement. https://posts.specterops.io/offensive-lateral-movement-1744ae62b14f.

He, Z., Xu, X., Huang, J., & Deng, S. (2005). Fp-outlier: Frequent pattern based outlier detection. *Comput. Sci. Inf. Syst., 2*, 103–118. https://doi.org/10.2298/CSIS0501103H

Himura, Y., Fukuda, K., Cho, K., Borgnat, P., Abry, P., & Esaki, H. (2013). Synoptic graphlet: Bridging the gap between supervised and unsupervised profiling of host-level network traffic. *IEEE/ACM Trans. Netw., 21*(4), 1284–1297. https://doi.org/10.1109/TNET.2012.2226603

Hoang, T., Mörchen, F., Fradkin, D., & Calders, T. (2012). Mining compressing sequential problems. *Proceedings of the twelfth siam international conference on data mining (sdm 2012, anaheim ca, usa, april 26-28, 2012)* (pp. 319–330). United States: Society for Industrial and Applied Mathematics (SIAM).

Hoang, X. D., & Hu, J. (2004). An efficient hidden markov model training scheme for anomaly intrusion detection of server applications based on system calls. *in ieee intl. conf. on networks* (pp. 470–474).

Hofmeyr, S., Forrest, S., & Somayaji, A. (1998). Intrusion detection using sequences of system calls. *J. of Comp. Sec., 6*, 151–180.

Holt, R., Aubrey, S., DeVille, A., Haight, W., Gary, T., & Wang, Q. (2019). Deep autoencoder neural networks for detecting lateral movement in computer networks. *Proceddings of the 21st international conference on artificial intelligence (icai'19)*.

Hornik, K., Feinerer, I., Kober, M., & Buchta, C. (2012). Spherical k-means clustering. *Journal of Statistical Software, 050*(i10).https://EconPapers.repec.org/RePEc:jss:jstsof:v:050:i10

Hu, J., Yu, X., Qiu, D., & Chen, H.-H. (2009). A simple and efficient hidden markov model scheme for host-based anomaly intrusion detection. *IEEE Network, 23*(1), 42–47. https://doi.org/10.1109/MNET.2009.4804323

Husàk, M., Apruzzese, G., Yang, S., & Werner, G. (2021). Towards an efficient detection of pivoting activity. *Ifip/ieee international symposium on integrated network management (im)* (pp. 1–6).

Iglesias, J. A., Angelov, P., Ledezma, A., & Sanchis, A. (2009a). Modelling evolving user behaviours. *2009 ieee workshop on evolving and self-developing intelligent systems* (pp. 16–23). https://doi.org/10.1109/ESDIS.2009.4938994

Iglesias, J. A., Ledezma, A., & Sanchis, A. (2009b). Creating user profiles from a command-line interface: A statistical approach. *User modeling, adaptation, and personalization* (pp. 90–101). Berlin, Heidelberg: Springer Berlin Heidelberg.

Kaiafas, G., Varisteas, G., Lagraa, S., State, R., Nguyen, C., Ries, T., & Ourdane, M. (2018). Detecting malicious authentication events trustfully. *Noms 2018 - 2018 ieee/ifip network operations and management symposium* (pp. 1–6).

Kang, D.-K., Fuller, D., & Honavar, V. (2005). Learning classifiers for misuse and anomaly detection using a bag of system calls representation. *Proceedings from the sixth annual ieee smc information assurance workshop* (pp. 118–125). https://doi.org/10.1109/IAW.2005.1495942

Karagiannis, T., Papagiannaki, K., & Faloutsos, M. (2005). Blinc: Multilevel traffic classification in the dark. *SIGCOMM Comput. Commun. Rev., 35*(4), 229–240. https://doi.org/10.1145/1090191.1080119

Kent, A., Liebrock, L., & Neil, J. (2015). Authentication graphs: Analyzing user behavior within an enterprise network. *Comp. Sec., 48*, 150–166.

Kim, H., Claffy, K., Fomenkov, M., Barman, D., Faloutsos, M., & Lee, K. (2008). Internet traffic classification demystified: Myths, caveats, and the best practices, . In *CoNEXT '08Proceedings of the 2008 acm conext conference*. New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/1544012.1544023

Kosoresow, A., & Hofmeyer, S. (1997). Intrusion detection via system call traces. *IEEE Software, 14*(5), 35–42. https://doi.org/10.1109/52.605929

Kruegel, C., Mutz, D., Valeur, F., & Vigna, G. (2003). On the detection of anomalous system call arguments. *Computer security – esorics 2003* (pp. 326–343). Berlin, Heidelberg: Springer Berlin Heidelberg.

Lakhina, A., Crovella, M., & Diot, C. (2005). Mining anomalies using traffic feature distributions. *SIGCOMM Comput. Commun. Rev., 35*(4), 217–228. https://doi.org/10.1145/1090191.1080118

Lam, H. T., Mörchen, F., Fradkin, D., & Calders, T. (2014). Mining compressing sequential patterns. *Statistical Analysis and Data Mining: The ASA Data Science Journal, 7*(1), 34–52. https://doi.org/10.1002/sam.11192

Lane, T. (1999). Hidden markov models for human / computer interface modeling. *Proc. the IJCAI-99 Workshop on Learning About Users*, 35–44.

Lane, T., & Brodley, C. E. (1999). Temporal sequence learning and data reduction for anomaly detection. *ACM Trans. Inf. Syst. Secur., 2*(3), 295–331. https://doi.org/10.1145/322510.322526

Lazarevic, A., Ertoz, L., Kumar, V., Ozgur, A., & Srivastava, J. (a). A comparative study of anomaly detection schemes in network intrusion detection. In Proceedings of the 2003 SIAM International Conference on Data Mining (SDM), pp. 25–36). 10.1137/1.9781611972733.3.

Lee, W., Stolfo, S., & Chan, P. K. (1997). Learning patterns from unix process execution traces for intrusion detection. *Proceedings of AAAI97 Workshop on AI Methods in Fraud and Risk Management*, 50–56.

Li, S., Lee, R., & Lang, S.-D. (2007). Mining distance-based outliers from categorical data. *Seventh ieee international conference on data mining workshops (icdmw 2007)* (pp. 225–230). https://doi.org/10.1109/ICDMW.2007.75

Li, Z., Nie, F., Chang, X., Nie, L., Zhang, H., & Yang, Y. (2018a). Rank-constrained spectral clustering with flexible embedding. *IEEE Transactions on Neural Networks and Learning Systems, 29*(12), 6073–6082. https://doi.org/10.1109/TNNLS.2018.2817538

Li, Z., Nie, F., Chang, X., Yang, Y., Zhang, C., & Sebe, N. (2018b). Dynamic affinity graph construction for spectral clustering using multiple features. *IEEE Transactions on Neural Networks and Learning Systems, 29*(12), 6323–6332. https://doi.org/10.1109/TNNLS.2018.2829867

Liao, Y., & Vemuri, V. R. (2002). Using text categorization techniques for intrusion detection. *11th USENIX security symposium (USENIX security 02)*. San Francisco, CA: USENIX Association. https://www.usenix.org/conference/11th-usenix-security-symposium/using-text-categorization-techniques-intrusion-detection

Liu, L., De Vel, O., Han, Q.-L., Zhang, J., & Xiang, Y. (2018a). Detecting and preventing cyber insider threats: A survey. *IEEE Communications Surveys Tutorials, 20*(2), 1397–1417. https://doi.org/10.1109/COMST.2018.2800740

Liu, M., Xue, Z., Xu, X., Zhong, C., & Chen, J. (2018b). Host-based intrusion detection system with system calls: Review and future trends. *ACM Comput. Surv., 51*(5). https://doi.org/10.1145/3214304

Luo, M., Chang, X., Nie, L., Yang, Y., Hauptmann, A. G., & Zheng, Q. (2018a). An adaptive semisupervised feature analysis for video semantic recognition. *IEEE Transactions on Cybernetics, 48*(2), 648–660. https://doi.org/10.1109/TCYB.2017.2647904

Luo, M., Nie, F., Chang, X., Yang, Y., Hauptmann, A. G., & Zheng, Q. (2018b). Adaptive unsupervised feature selection with structure regularization. *IEEE Transactions on Neural Networks and Learning Systems, 29*(4), 944–956. https://doi.org/10.1109/TNNLS.2017.2650978

Magán-Carrión, R., Urda, D., Díaz-Cano, I., & Dorronsoro, B. (2020). Towards a reliable comparison and evaluation of network intrusion detection systems based on machine learning approaches. *Applied Sciences, 10*(5). https://doi.org/10.3390/app10051775

Maggi, F., Matteucci, M., & Zanero, S. (2010). Detecting intrusions through system call sequence and argument analysis. *IEEE Trans. Dependable Secur. Comput., 7*(4), 381–395. https://doi.org/10.1109/TDSC.2008.69

Mandiant. (2020). *2020 Security Effectiveness: Deep Dive Into Cyber Reality*.

Maxion, R., & Townsend, T. (2002). Masquerade detection using truncated command lines. *Proceedings International Conference on Dependable Systems and Networks*, 219–228.

McClish, D. K. (1989). Analyzing a portion of the roc curve. *Medical Decision Making, 9*(3), 190–195. https://doi.org/10.1177/0272989X8900900307PMID: 2668680

McHugh, J., McLeod, R., & Nagaonkar, V. (2008). Passive network forensics: Behavioural classification of network hosts based on connection patterns. *SIGOPS Oper. Syst. Rev., 42*(3), 99–111. https://doi.org/10.1145/1368506.1368520

Mehnaz, S., & Bertino, E. (2017). Ghostbuster: A fine-grained approach for anomaly detection in file system accesses, . In *CODASPY '17Proceedings of the seventh acm on conference on data and application security and privacy* (pp. 3–14). New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3029806.3029809

Milajerdi, S. M., Gjomemo, R., Eshete, B., Sekar, R., & Venkatakrishnan, V. (2019). Holmes: Real-time apt detection through correlation of suspicious information flows. *2019 ieee symposium on security and privacy (sp)* (pp. 1137–1152). https://doi.org/10.1109/SP.2019.00026

Moustafa, N., & Slay, J. (2015). Unsw-nb15: a comprehensive data set for network intrusion detection systems (unsw-nb15 network data set). *2015 military communications and information systems conference (milcis)* (pp. 1–6). https://doi.org/10.1109/MilCIS.2015.7348942

Murtaza, S. S., Khreich, W., Hamou-Lhadj, A., & Couture, M. (2013). A host-based anomaly detection approach by representing system calls as states of kernel modules. *2013 ieee 24th international symposium on software reliability engineering (issre)* (pp. 431–440). https://doi.org/10.1109/ISSRE.2013.6698896

Nguyen, T. T., & Armitage, G. (2008). A survey of techniques for internet traffic classification using machine learning. *IEEE Communications Surveys Tutorials, 10*(4), 56–76. https://doi.org/10.1109/SURV.2008.080406

Powell, B. A. (2020). Detecting malicious logins as graph anomalies. *Journal of Information Security and Applications, 54*, 102557. https://doi.org/10.1016/j.jisa.2020.102557

Purvine, E., Johnson, J. R., & Lo, C. (2016). A graph-based impact metric for mitigating lateral movement cyber attacks, . In *SafeConfig '16Proceedings of the 2016 acm workshop on automated decision making for active cyber defense* (pp. 45–52). New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/2994475.2994476

Qian, Q., & Xin, M. (2007). Research on hidden markov model for system call anomaly detection. *Intelligence and security informatics* (pp. 152–159). Berlin, Heidelberg: Springer Berlin Heidelberg.

Rawat, S., Gulati, V., Pujari, A., & Vemuri, V. (2006). Intrusion detection using text processing techniques with a binary-weighted cosine metric. *Journal of Information Assurance and Security (JIAS), 1*(1), 43–50.

Russinovich, M., Solomon, D., & Ionescu, A. (2013). *Windows Internals, Part 2, 6th edition* (6th). Pearson.

Salem, M. B., Hershkop, S., & Stolfo, S. J. (2008). *A survey of insider attack detection research* (pp. 69–90)). Boston, MA: Springer US.

Salem, M. B., & Stolfo, S. J. (2011). Modeling user search behavior for masquerade detection. *Recent advances in intrusion detection* (pp. 181–200). Berlin, Heidelberg: Springer Berlin Heidelberg.

Salman, O., Elhajj, I., Kayssi, A., & Chehab, A. (2020). A review on machine learning-based approaches for internet traffic classification. *Ann. des Télécommunications, 75*, 673–710.

Schonlau, M., DuMouchel, W., Ju, W., Karr, A., Theus, M., & Vardi, Y. (2001). Computer intrusion: Detecting masquerades. *Stat. Sci., 16*, 1–17.

Sharma, A., & Paliwal, K. (2007). Detecting masquerades using a combination of naïve bayes and weighted rbf approach. *Journal in Computer Virology, 3*, 237–245.

Siadati, H., & Memon, N. (2017). Detecting structurally anomalous logins within enterprise networks, . In *CCS '17Proceedings of the 2017 acm sigsac conference on computer and communications security* (pp. 1273–1284). https://doi.org/10.1145/3133956.3134003

Siebes, A., Vreeken, J., & van Leeuwen, M. (b). Item sets that compress. In Proceedings of the 2006 SIAM International Conference on Data Mining (SDM), pp. 395–406). 10.1137/1.9781611972764.35.

Singh, M., Mehtre, B., & Sangeetha, S. (2019). User behavior profiling using ensemble approach for insider threat detection. *2019 ieee 5th international conference on identity, security, and behavior analysis (isba)* (pp. 1–8). https://doi.org/10.1109/ISBA.2019.8778466

Smets, K., & Vreeken, J. (2011). *The odd one out: Identifying and characterising anomalies* (pp. 804–815). https://doi.org/10.1137/1.9781611972818.69

Smets, K., & Vreeken, J. (2012). Slim: Directly mining descriptive patterns. *In proc of the sdm*.

Taha, A., & Hadi, A. S. (2019). Anomaly detection methods for categorical data: A review. *ACM Comput. Surv., 52*(2). https://doi.org/10.1145/3312739

Tan, G., Poletto, M., Guttag, J., & Kaashoek, F. (2003). Role classification of hosts within enterprise networks based on connection patterns, . In *ATEC '03Proceedings of the annual conference on usenix annual technical conference* (p. 2). USA: USENIX Association.

Tan, K. M. C., & Maxion, R. A. (2002). "why 6?" defining the operational limits of stide, an anomaly-based intrusion detector. *Proceedings 2002 ieee symposium on security and privacy* (pp. 188–201). https://doi.org/10.1109/SECPRI.2002.1004371

Tandon, G., & Chan, P. K. (2003). Learning rules from system call arguments and sequences for anomaly detection. *Technical Report*.

Tapiador, J. E., & Clark, J. A. (2010). Information-theoretic detection of masquerade mimicry attacks. *2010 fourth international conference on network and system security* (pp. 183–190). https://doi.org/10.1109/NSS.2010.55

Tatti, N., & Vreeken, J. (2012). The long and the short of it: Summarising event sequences with serial episodes, . In *KDD '12Proceedings of the 18th acm sigkdd international conference on knowledge discovery and data mining* (pp. 462–470). New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/2339530.2339606

Tavallaee, M., Bagheri, E., Lu, W., & Ghorbani, A. A. (2009). A detailed analysis of the kdd cup 99 data set, . In *CISDA'09Proceedings of the second ieee international conference on computational intelligence for security and defense applications* (pp. 53–58). IEEE Press.

Teng, H., Chen, K., & Lu, S. (1990). Security audit trail analysis using inductively generated predictive rules. *Sixth conference on artificial intelligence for applications*. Los Alamitos, CA, USA: IEEE Computer Society. https://doi.org/10.1109/CAIA.1990.8916724,25,26,27,28,29

Thompson, M., & Zucchini, W. (1989). On the statistical analysis of roc curves. *Statistics in medicine, 8 10*, 1277–1290.

Tian, S., Mu, S., & Yin, C. (2007). Sequence-similarity kernels for svms to detect anomalies in system calls. *Neurocomputing, 70*(4), 859–866. https://doi.org/10.1016/j.neucom.2006.10.017Advanced Neurocomputing Theory and Methodology

Uci kdd archive: Kdd cup 1999 data (1999). http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html.

van Leeuwen, M., & Vreeken, J. (2014). *Mining and using sets of patterns through compression*.

van Leeuwen, M., Vreeken, J., & Siebes, A. (2006). Compression picks item sets that matter. *Knowledge discovery in databases: Pkdd 2006* (pp. 585–592). Berlin, Heidelberg: Springer Berlin Heidelberg.

Vendramin, L., Campello, R. J. G. B., & Hruschka, E. R. (2010). Relative clustering validity criteria: A comparative overview. *Stat. Anal. Data Min., 3*, 209–235.

Warrender, C., Forrest, S., & Pearlmutter, B. (1999). Detecting intrusions using system calls: alternative data models. *Proceedings of the 1999 ieee symposium on security and privacy*.

Wei, S., Mirkovic, J., & Kissel, E. (2006). Profiling and clustering internet hosts. *Proceedings of the international conference on data mining* (pp. 1–8).

Xie, M., Hu, J., & Slay, J. (2014). Evaluating host-based anomaly detection systems: Application of the one-class svm algorithm to adfa-ld. *2014 11th international conference on fuzzy systems and knowledge discovery (fskd)* (pp. 978–982). https://doi.org/10.1109/FSKD.2014.6980972

Xu, K., Wang, F., & Gu, L. (2011). Network-aware behavior clustering of internet end hosts. *2011 proceedings ieee infocom* (pp. 2078–2086). https://doi.org/10.1109/INFCOM.2011.5935017

Xu, K., Zhang, Z.-L., & Bhattacharyya, S. (2005). Profiling internet backbone traffic: Behavior models and applications. *SIGCOMM Comput. Commun. Rev., 35*(4), 169–180. https://doi.org/10.1145/1090191.1080112

Ye, N., Li, X., Chen, Q., Emran, S., & Xu, M. (2001). Probabilistic techniques for intrusion detection based on computer audit data. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans, 31*(4), 266–274. https://doi.org/10.1109/3468.935043

Yen, T.-F., Oprea, A., Onarlioglu, K., Leetham, T., Robertson, W., Juels, A., & Kirda, E. (2013). Beehive: Large-scale log analysis for detecting suspicious activity in enterprise networks, . In *ACSAC '13Proceedings of the 29th annual computer security applications conference* (pp. 199–208). New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/2523649.2523670

Yeung, D.-Y., & Ding, Y. (2003). Host-based intrusion detection using dynamic and static behavioral models. *Pattern Recognition, 36*(1), 229–243. https://doi.org/10.1016/S0031-3203(02)00026-2

Yu, J. X., Qian, W., Lu, H., & Zhou, A. (2006). Finding centric local outliers in categorical/numerical spaces. *Knowl. Inf. Syst., 9*(3), 309–338.