Malicious Cyber Activity Detection using Zigzag Persistence

Audun Myers
Mathematics, Statistics, and Data Science
Pacific Northwest National Laboratory
Seattle, WA, United States
audun.myers@pnnl.gov

Alyson Bittner
Mathematics, Statistics, and Data Science
Pacific Northwest National Laboratory
Seattle, WA, United States
alyson.bittner@pnnl.gov

Sinan Aksoy
Foundational Data Science
Pacific Northwest National Laboratory
Seattle, WA, United States
sinan.aksoy@pnnl.gov

Dan Best
Cyber Security
Pacific Northwest National Laboratory
Seattle, WA, United States
daniel.best@pnnl.gov

Gregory Henselman-Petrusek
Mathematics, Statistics, and Data Science
Pacific Northwest National Laboratory
Seattle, WA, United States
gregory.roek@pnnl.gov

Helen Jenne
Mathematics, Statistics, and Data Science
Pacific Northwest National Laboratory
Seattle, WA, United States
helen.jenne@pnnl.gov

Cliff Joslyn
Mathematics, Statistics, and Data Science
Pacific Northwest National Laboratory
Seattle, WA, United States
cliff.joslyn@pnnl.gov

Bill Kay
Computational Mathematics
Pacific Northwest National Laboratory
Seattle, WA, United States
william.kay@pnnl.gov

Garret Seppala
Cyber Resilience Foundations
Pacific Northwest National Laboratory
Seattle, WA, United States
garret.seppala@pnnl.gov

Stephen J. Young
Computational Mathematics
Pacific Northwest National Laboratory
Seattle, WA, United States
stephen.young@pnnl.gov

Emilie Purvine
Mathematics, Statistics, and Data Science
Pacific Northwest National Laboratory
Seattle, WA, United States
emilie.purvine@pnnl.gov

2023 IEEE Conference on Dependable and Secure Computing (DSC) | 979-8-3503-8211-2/23/$31.00 ©2023 IEEE | DOI: 10.1109/DSC61021.2023.10354204

***Abstract*—**

In this study we synthesize zigzag persistence from topological data analysis with autoencoder-based approaches to detect malicious cyber activity, and derive analytic insights. Cybersecurity aims to safeguard computers, networks, and servers from various forms of malicious attacks, including network damage, data theft, and activity monitoring. We focus on the cybersecurity domain and investigate the detection of malicious activity using log data. We consider the dynamics of the log data and explore the changing topology of a hypergraph representation of this data to gain insights into the underlying activity. These hypergraphs capture complex interactions between processes, together with their temporal information. To study the changing topology we use zigzag persistence, which captures how topological features persist at multiple dimensions over time. We observe that this detects malicious activity in a cyber data set. To automate this detection we implement an autoencoder trained on a vectorization of the resulting zigzag persistence barcodes. Our experimental results demonstrate the effectiveness of the autoencoder in detecting malicious activity. Overall, this study highlights the potential of zigzag persistence and its combination with temporal hypergraphs for analyzing cybersecurity log data and detecting malicious behavior.

This work was supported by the Pacific Northwest National Laboratory operated for the U. S. Department of Energy (DOE) by Battelle under Contract DE-AC06-76RL01830

I. Introduction

In this study, we leverage zigzag persistence [9], a concept from Topological Data Analysis (TDA) [10], [24], to delve into the temporal activity of cyber data and effectively detect malicious behavior.

Cybersecurity aims to safeguard computers, networks, and servers from various forms of malicious attacks including network damage, data theft, and activity monitoring. These attacks are typically carried out by gaining unauthorized access, and there is often evidence of these attacks in the underlying log data which captures information such as timestamps, IP addresses, ports, and executables. However, detecting malicious activity in the log data is challenging due to its size and complexity.

One common approach to finding malicious activity in cyber logs involves constructing graph representations of the data representations, such as process trees [16] or flow networks [4], that model dyadic relations between entities. However, standard graphs cannot capture multi-way interactions that are common in cyber data. Instead, using higher dimensional graphs, known as hypergraphs [7], to model cyber logs effectively captures the complex interactions present between users, processes, ports, and other resources. Hypergraphs have proven valuable in diverse branches of data science, including machine learning, biology, and social networks [13], [15], [23].

While hypergraphs capture the complex multi-way relationships, traditional *static* hypergraphs may fail to additionally represent the dynamic nature of cyber systems. However, by incorporating temporal information on nodes, hyperedges, or incidences, *temporal hypergraphs* [5], [11], [20] offer a solution. By allowing nodes and edges to appear and disappear over time and connect different sets of nodes at different points in time, temporal hypergraphs provide a suitable framework for studying dynamical systems of complex relations.

One approach to studying temporal hypergraphs that we implement in this work is to represent the temporal attributed hypergraph as a sequence - one hypergraph per sliding time window representing the state of the system. This sequential representation of the temporal hypergraph allows one to treat the sequence as a dynamical process $H_t \mapsto H_{t+1}$ gaining a dynamical systems perspective. For the cyber data we study each hypergraph in the sequence as a set of nodes and a set of hyperedges, where each hyperedge represents a distinct named entity (e.g., a program executable, port, or user) and each hyperedge can include different sets of nodes at different times.

Our primary objective is to analyze temporal hypergraph representations of cyber log data to effectively detect malicious activity. Our claim is that malicious cyber activity will often exhibit unique attack patterns in the log data, resulting in topological changes in the representations over time. Specifically, we investigate a hypergraph representation constructed with executables as nodes and the destination ports as hyperedges, which should have a changing topology due to malicious activity often having more executables at different time scales compared to benign activity. This intuition will be illustrated in Section III-C.

The evolution of hypergraph structure and topology over time naturally fits into a use case of zigzag persistence, a tool from the field of Topological Data Analysis (TDA). With temporal hypergraphs providing a valuable framework for capturing complex dynamical systems we need to build an understanding of the complex patterns and structural changes in these temporal hypergraphs, and this is where zigzag persistence comes into play. Zigzag persistence captures how, when, and for how long topological features at multiple dimensions persist. For example, is a component lasting for a long time and always present or does it intermittently appear?

Zigzag persistence has been previously used for studying temporal graph models [19] of transportation networks and for intermittency detection. This method has also been recently extended to study temporal hypergraphs for both cyber and social network data [18]. By leveraging the power of zigzag persistence, one is able to delve deep into the intricate temporal dynamics of (hyper)graphs, unveiling hidden trends, detecting critical events, and revealing the underlying structural transformations that shape the system's behavior.

To determine the viability of this approach we will implement an autoencoder as a form of anomaly detection on a vectorization of the resulting zigzag persistence barcodes. We will train the model to detect suspicious activity by investigating vectors that have high reconstruction loss.

We begin in Section II by introducing notation and definitions for temporal hypergraphs, zigzag persistence, and how we use zigzag persistence to study temporal hypergraphs. We also introduce the concept of an autoencoder. In Section III we describe the cyber data, our experimental design, and some intuition behind using dynamic topology to identify anomalous behavior. We then demonstrate the ability of the pipeline to detect malicious activity in IV. We provide future goals and conclusions on this work in Section V.

II. Computational Tools

The process of computing zigzag persistence for a temporal hypergraph begins with a sequence of representative hypergraphs. We then transform each hypergraph into an abstract simplicial complex and examine the appearance and disappearance of topological features across multiple dimensions in this sequence using zigzag persistence. In the final step of our pipeline we vectorize the zigzag persistence barcode and use an autoencoder to identify anomalous barcodes. In order to describe our experimental design in the context of cyber log data in Section III-B, we first introduce the necessary definitions and background in a general setting.

A. Hypergraphs and Abstract Simplicial Complexes

A hypergraph, $H = (V, E)$, analogous to a graph, is represented by a set of vertices, V and a set of (hyper)edges E. The main difference between a hypergraph and a classical graph is that an edge $e \in E$ can be an arbitrary subset of vertices $e \subseteq V$ as opposed to a pair. If $|e| = k$ then we say that e is a k-edge. A temporal hypergraph is a sequence of n hypergraphs, denoted as $\mathcal{H} = H_0, H_1, H_2, \ldots, H_{n-1}$. The sequence can be viewed as a discrete dynamical process, where H_t transitions to H_{t+1}, enabling us to gain insights into the dynamics of the underlying system.

An abstract simplicial complex (ASC), denoted as K, is a collection of sets that is closed under taking subsets. Formally, $K = \{\sigma\}$ is an ASC if whenever $\tau \subset \sigma \in K$ then $\tau \in K$. Each set, σ, is called a simplex, and if $|\sigma| = k$ then σ has dimension $k-1$ and is called a $(k-1)$-simplex. Geometrically, 0-simplices represent points or vertices, 1-simplices represent lines or edges, 2-simplices represent filled-in triangles, and so on. For $\tau, \sigma \in K$ we say that $\tau \neq \emptyset$ is a face of σ if $\tau \subseteq \sigma$. The definition of an ASC implies that every simplex is closed under the face relation, meaning it includes all of its faces (except for the empty set) as defined by the power set of the simplex.

Note that an ASC can be thought of as a hypergraph with an extra requirement on the edges, but the reverse is not true; a general hypergraph need not be an ASC. Although various methods exist for constructing an ASC from a hypergraph [12] in this paper we consider the associated ASC of a hypergraph [22]. The associated ASC consists of a simplex for each hyperedge. In other words, the associated ASC of a hypergraph contains all subsets of all hyperedges:

$$K(H) = \{\sigma \subseteq V : \exists e \in E, \sigma \subseteq e\}.$$

As many real-world hypergraphs have some large hyperedges this can become costly, and unnecessary if computing only low dimensional homology. In practice, to reduce computational complexity, we keep only those simplicies up to a small maximum dimension, $p = 2$ or 3.

B. Simplicial Homology

Simplicial homology is an algebraic approach to analyze the structure of an ASC by quantifying the number of p-dimensional features. 0-dimensional features are connected components, 1-dimensional features are graph cycles, 2-dimensional features are three-dimensional hollow polyhedra, and so on. The p-dimensional simplicial homology of an ASC, K, denoted $H_p(K)$, is a vector space whose basis represents the p-dimensional features of K. The rank of $H_p(K)$ then counts the number of p-dimensional features. This rank is denoted β_p and called the p^{th} Betti number of K. The algebraic details of simplicial homology computations and Betti numbers can be found in [14].

While Betti numbers provide valuable insights into the changing topology of hypergraph snapshots, they do not capture the relationships between the topology of consecutive snapshots. In other words, Betti numbers alone do not reveal if a feature persists throughout the entire sequence. To address this limitation and track the changes in homology and their interconnections across a sequence of ASCs, we employ the technique of zigzag persistent homology.

C. Persistent and Zigzag Homology

This section provides an introduction to persistent homology (PH) [24] and how it generalizes to zigzag persistent homology. For a detailed introduction to PH we suggest [17], [21], for zigzag see [9].

PH is used to obtain a sense of the shape and size of a data set at multiple scale resolutions. To gain some intuition on what this means we describe a common setting in which PH is applied, that of a point cloud $X \subseteq \mathbb{R}^n$. At a given scale (i.e., distance value) we connect points in X within the given distance to form an ASC. As that scale increases so does the ASC and topological features are born (appear) and die (are filled in). PH tracks the birth and death of these features as the distance scale varies to form a topological fingerprint. Short-lived features may indicate noise while long-lived ones often indicate meaningful features. The birth and death thresholds provide an idea of the general size or geometry of each feature, which can in turn provide intuition and interpretation back into the data itself. For example, the presence of a 1-dimensional loop might mean that the data is cyclical or repetitive whereas the presence of multiple 0-dimensional components could indicate strong clustering of the data.

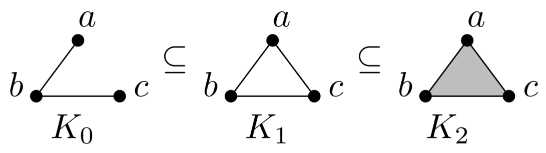


Fig. 1: Example of a 1-dimensional feature being filled in from one simplicial complex to the next in a nested sequence.

A point cloud is not the only setting for PH. In general, only a sequence of nested ASCs[1], often referred to as a filtration, is necessary:

$$K_0 \subseteq K_1 \subseteq K_2 \subseteq \ldots \subseteq K_n. \quad (1)$$

For a given dimension p we can calculate $H_p(K_i)$ for each K_i. In order to capture how the homology changes from K_i to K_{i+1} we rely on the fact that K_i is a sub-complex of K_{i+1} and so the components of the topological features found in K_i (e.g., the vertices, edges, and higher dimensional simplices) must also be found in K_{i+1}. If these components also form a topological feature in K_{i+1} then the feature persists. If they do not form a feature in K_{i+1} then the feature dies. In Figure 1 we see a 1-dimensional feature in K_1 consisting of the edges $(a, b), (a, c), (b, c)$. These edges are present in K_2 but they no longer form a 1-dimensional feature because of the presence of the triangle (a, b, c). The appearance and disappearance of p-dimensional features in the filtration is tracked in a summary known as a persistence barcode, a collection of intervals, one for each topological feature identified. Each feature has an associated interval $[b, d]$ that indicates the index of the appearance of the feature, its birth threshold b, and its disappearance, its death threshold d. We denote the barcode for dimension p of a sequence $\mathcal{K}$ as $D_p(\mathcal{K}) = \{[b_i, d_i]\}$, or simply D_p is the sequence is clear from context. In the example in Figure 1 the 1-dimensional feature is born at $b = 1$ and dies at $d = 2$. The algebraic mechanics of tracking features across spaces via their inclusions is best left to the references cited above. For the purposes of this paper only the intuition is necessary.

Given a temporal hypergraph sequence we can construct $K_i := K(H_i)$. If we are lucky enough to have a sequence in which $K_i \subseteq K_{i+1}$ for all i then we can apply PH directly. However, this is rarely the case. There are plenty of examples in which hypergraph vertices and edges are both added *and* removed over time. This is where zigzag homology, which extends the concept of PH to handle ASC sequences with addition and removal of simplices, can be applied. Given an arbitrary sequence of ASCs, $K_0, K_1, \ldots, K_n$, we can form an augmented sequence with interwoven unions[2]:

$$K_0 \subseteq K_0 \cup K_1 \supseteq K_1 \subseteq K_1 \cup K_2 \cdots K_{n-1} \cup K_n \supseteq K_n.$$

[1]In fact, persistent homology can be applied in even more general settings but for the purposes of this paper we won't consider arbitrary topological spaces or chain complexes.

[2]Intersections can also be used, just flip around the subset containments.

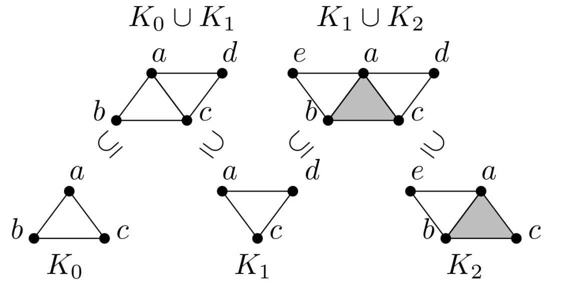


Fig. 2: Example of a zigzag sequence with interwoven unions.

The idea of zigzag homology is similar to PH. Even though the inclusions are not in the same direction throughout the augmented sequence their presence still allows us to track whether a feature in one ASC is the same as a feature in the next. In Figure 2 we show an example sequence of three ASCs with interwoven unions. There is a 1-dimensional feature in all three ASCs but through the use of zigzag we can see that they are all different loops. The barcode consists of three intervals: [0,1] for loop $(a,b),(a,c),(b,c)$, [0.5, 1.5] for loop $(a,c),(a,d),(c,d)$, and [1.5, 2] for loop $(a,b),(a,e),(b,e)$. If a loop is born (resp. dies) at a union step between i and $i+1$ we say that it is born (resp. dies) at the midpoint, $i+\frac{1}{2}$.

For a more detailed introduction to zigzag persistence in the context of studying temporal hypergraphs, we refer the reader to [18], which includes an example illustrating the procedure.

D. Vectorization of Persistence Barcodes

To implement an autoencoder for studying zigzag persistence barcodes we need to create a faithful vector representation of the barcode. While there are many methods for vectorizing a barcode for machine learning applications, such as persistence images [1] and persistence landscapes [8], these are often high dimensional making the autoencoder training more burdensome. In this work we use Adcock-Carlsson Coordinates (ACCs) [2] as they are computationally and storage efficient and have been shown to provide comparable performance to the more advanced vectorization methods for classification tasks [6]. The ACCs are calculated as

$$\begin{aligned} ACC(D_p) = \Big[&\sum_i b_i(d_i - b_i), \\ &\sum_i (d_{\max} - d_i)(d_i - b_i), \\ &\sum_i b_i^2 (d_i - b_i)^4, \\ &\sum_i (d_{\max} - d_i)^2 (d_i - b_i)^4 \Big]. \end{aligned} \tag{2}$$

We then stacked the ACCs for each dimension $p \in [1,2]$ into a single eight-dimensional vector.

E. Autoencoder

One of the ways to leverage the power of neural networks to perform anomaly detection on a dataset is through the use of autoencoders. An autoencoder is a particular kind of feed-forward neural network that takes in data, compresses it via encoding layers, and then attempts to reconstruct the original representation from the compressed form through decoding layers as shown in Fig. 3. The metric used to quantify the difference between the reconstructed version and the original data is called the reconstruction loss.

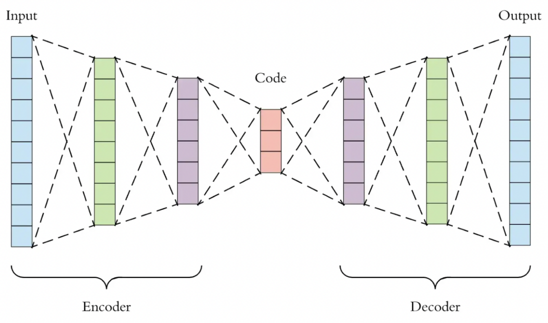


Fig. 3: Autoencoder schema.

If an autoencoder is trained on "typical" data, then the reconstruction loss for unseen typical data should be low whereas the reconstruction loss for "atypical" data will be much greater. This is the motivation for utilizing autoencoders to detect anomalies in data. More precisely, if the reconstruction loss of unseen data is above a chosen threshold then the unseen data is considered anomalous.

III. Methodology

A. Data and Data Preparation

The Operationally Transparent Cyber (OpTC) dataset [3] used in our experiments was created by the Defense Advanced Research Projects Agency (DARPA) as part of a mission to test scaling of cyber attack detection. The data consists of log records of both benign and malicious activity, with an associated ground truth document describing the attack events. The attack events include downloading malicious PowerShell Empire, privilege escalation, credential theft, network scanning, and lateral movement. The data contains both flow and host logs. The elements of each record vary depending on the type of log but the format is standardized allowing for easy analysis across log-types. In this paper we consider only the flow subset of records but future work could include a more comprehensive analysis.

We focus our analysis of the data on the first day, September 23, on a sampling of both benign and malicious hosts, see Table I. For the benign set, we selected hosts that did not appear in the ground truth document and chose a subset of hosts

Benign (Training) Hosts:	0005, 0006, 0010, 0012, 0071, 0162 0213, 0222, 0274, 0304, 0461, 0906
Evaluation (Testing) Hosts:	0201, 0402, 0660

TABLE I: Subset of hosts used for training and testing

with varying levels of activity relative to the malicious hosts. In particular, hosts 0005, 0006, 0010, 0012 had significantly less (approximately half as much) activity, hosts 0162, 0304, 0461, 0906 had comparable amounts of activity, and hosts 0071, 0213, 0222, 0274 had more activity.

We performed selective filtering of the data as an initial preprocessing step. In particular, we filtered out actions where the image path (executable) or source IP address were missing and where the source IP address corresponded to localhost activity. Since the network traffic data in the dataset is bidirectional, we also filtered out actions where the destination port was ephemeral.

B. *Experimental Design*

We designed an experiment with the aim to identify source IPs that are responsible for malicious activity, and the particular time in which the malicious activity occurs, by using the topology of their network interactions. We create hypergraphs for a given source IP and sequence of timeframes, and then vectorize the hypergraph sequences in two ways: 1) using zigzag persistence and 2) a more naive hypergraph property embedding. In order to understand the viability of zigzag persistence diagrams to encode differences in the topological dynamics of benign and malicious activity we train two autoencoders, one on the vectors derived from zigzag persistence and a second on the hypergraph property vectors. We chose to use an autoencoder based on the assumption that a large proportion of traffic on the network is typical benign activity, whereas malicious activity is fairly uncommon. We then perform autoencoder-based anomaly detection separately on the two vectorizations and examine how the anomalies align with the ground truth document. If our zigzag autoencoder successfully identifies malicious activity on the network, this provides evidence that the topological information encoded by the zigzag persistence barcodes can aid in cybersecurity efforts. We use the autoencoder trained on hypergraph property vectors as a comparison.

The details and pipeline of these experiments are illustrated in Fig. 4. Our experimental design begins with the log data, see the box labeled *Log Data* in Fig. 4. We show a small set of the OpTC log data including the specific columns needed: timestamp, source IP, destination port, and image path (executable). Using the timestamps, we break this log data into 10-minute windows that overlap by 5 minutes (see Fig. 4 box with label *Windows of Data*). From each of these windows we construct a collection of hypergraphs. For each source IP we collect all their records. Then we create a hypergraph with vertices as the executable files and hyperedges as the destination ports. For the hypergraph pertaining to source IP X the vertex for executable t is contained in the destination port edge r if there is a record with the (source IP, destination port, executable) tuple (X, t, r). In completing this research, we considered many combinations of hyperedges and nodes for constructing hypergraphs and found the clearest malicious activity detection on the OpTC dataset when using the image path/destination port construction.

For each source IP we apply zigzag persistence to the temporal sequence of hypergraph snapshots, as shown in Fig. 4 in the box labeled *Zigzag Persistence*, resulting in a barcode for each dimension (0 and 1). This full time barcode is further broken into sub-barcodes over 1 hour sub-windows. Each of these sub-barcodes are vectorized using the ACCs described in Section II-D. We trained the zigzag autoencoder on these ACC vectors from IPs in the benign host list from Table I and tested on those from the evaluation hosts. For each source IP we calculated the time series of mean squared error reconstruction loss as an indicator of abnormal or malicious activity. This is shown in Fig. 4, in the box labeled *Autoencoder*.

The zigzag autoencoder contains one fully-connected neural layer as the encoder and decoder. The input zigzag vectors are 8-dimensional, the autoencoder compresses the data into 2-dimensional vectors, and decompresses them back into 8-dimensions. The encoder and decoder of the model learn by minimizing the mean squared error between the original zigzag vector and the reconstructed 8-dimensional vector.

We trained a second autoencoder on some naive summary statistics of the hypergraphs as a feature vector on the collection of hypergraphs that occurred during the 1 hour sub-windows. For each of the hypergraphs during each sub-window we calculated the number of edges, number of nodes, number of components, and diameter of the largest component and then concatenated them together. This results in a 48-dimensional feature vector for each 1 hour window. The autoencoder again had a latent space of 2-dimensions to make a fair comparison to the first autoencoder.

By analyzing the reconstruction loss of the two autoencoders, we can compare the ability of the zigzag persistence barcodes and standard summary statistics to detect malicious activity.

C. *Intuition*

Before we transition to the results of our experiment we provide some intuition, through an example, for why the *dynamics* of the hypergraph topology, and not just the static topology of each snapshot, are important to detecting malicious activity. Fig. 5 shows hypergraphs from one benign and one malicious time period for source IP 142.20.56.202 on Host 201. It is apparent that the structural configuration of the hypergraph during benign activity differs from that during malicious activity, but from a topological perspective, the two snapshots are equivalent. Both hypergraphs exhibit two components and no higher-dimensional homology, indicating a similarity in their topological properties.

However, the two isolated snapshots do not tell the entire story of the topology. While the snapshots are topologically equivalent they do not account for the underlying dynamics of the topology (e.g., do these two components persist for long periods of time or do they quickly evolve?). By looking beyond the isolated snapshots we can gain a quick insight that, in fact, the dynamics of the malicious activity change at a much higher rate than the dynamics of the benign activity. Figure 6 shows the sequence of image path executables for

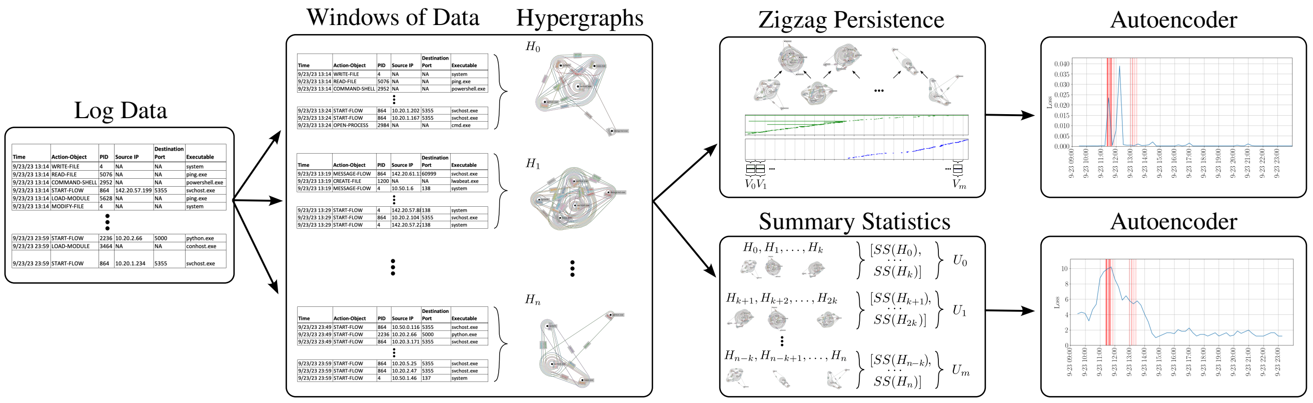
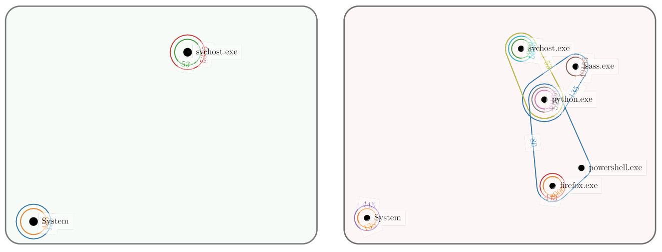
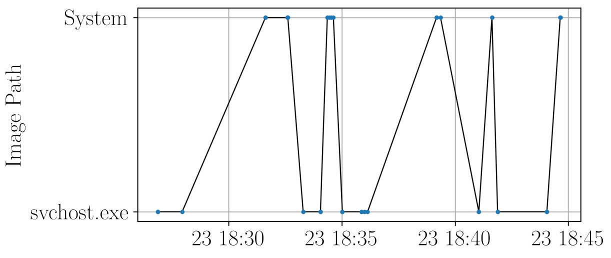


Fig. 4: Experimental design pipeline for study OpTC log data with autoencoders trained on the ACC vectors of the subwindowed zigzag persistence barcodes compared to the autoencoder trained on the concatenated summary statistics of hypergraphs during the same subwindows.

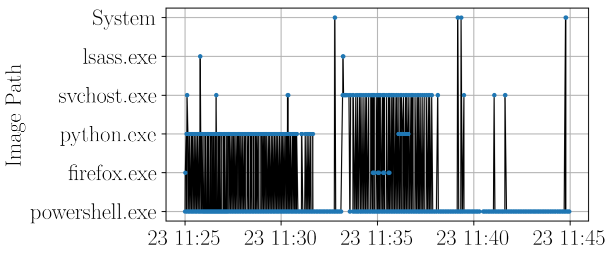


(a) Benign Activity (b) Malicious Activity.

Fig. 5: Hypergraphs formed during malicious and benign activity for source IP 142.20.56.202 on host 201 using destination ports as hyperedges and image path executables as nodes.



(a) Benign Activity



(b) Malicious Activity.

Fig. 6: Sequence of image path executables for a source IP 142.20.56.202 during 20 minute benign and malicious activity windows on host 201 (same windows as in Fig. 5).

source IP 142.20.56.202 on host 201 during the same 20 minute benign and malicious activity windows associated to the hypergraphs in Fig. 5. As shown, during the benign activity the only executables used were System and svchost.exe and tend to be executed every few minutes, while in the malicious activity many executables are used and are executed much more frequently. It is clear from this example that benign and malicious activity show changes at different time scales. As we will see in the results, the dynamics of the topology as shown by the zigzag barcode vectorizations allow us to detect a difference between benign and malicious activity patterns.

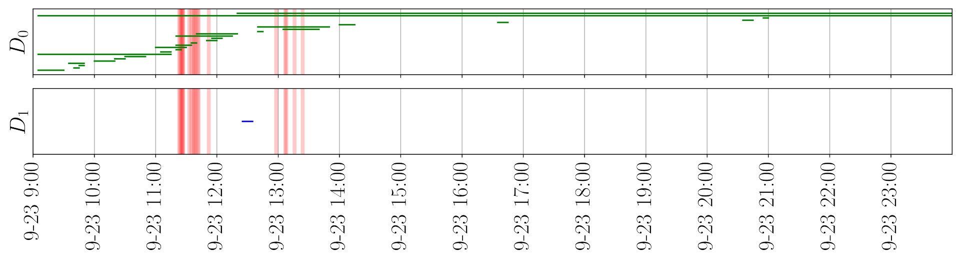
IV. Results

Here we demonstrate the ability of both the zigzag persistence and summary statistics to detect malicious activity for an example source IP. Namely, we demonstrate these results for source IP 142.20.56.202 for malicious activity and source IP 142.20.56.175 for benign activity on host 201 on September 23, 2019. We chose this malicious source IP and host to demonstrate the effectiveness of this autoencoder due to the variety of attacks during this time as shown in the ground truth data provided in the GitHub repository[3]. While there are limited malicious source IPs during this time window there are a very large number of benign source IPs. We chose source IP 142.20.56.175 as an exemplary benign source IP, but we found very similar dynamics and reconstruction loss values for other benign source IPs.

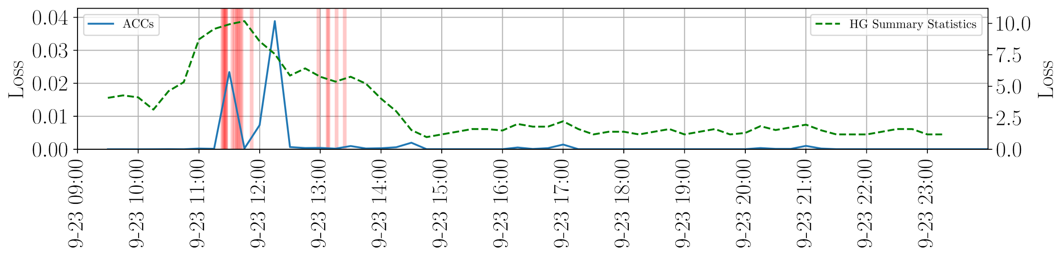
[3]See https://github.com/FiveDirections/OpTC-data for red team ground truth data

Figure 7 shows the zigzag persistence barcode (7a) and the reconstruction losses over time for the ACC vectors and the hypergraph summary statistics (7b). In both plots we have highlighted each of the ground truth malicious events from the OpTC ground truth diary as red vertical bars.

The main takeaway from Figure 7 is that while both



(a) Zigzag persistence barcodes.



(b) Autoencoder reconstruction loss (mean squared error).

Fig. 7: Autoencoder results for malicious source IP 142.20.56.202 on host 201 using the ACCs of the windowed zigzag persistence barcode compared to summary statistics with highlight red vertical lines for each malicious activity instance recorded in OpTC ground truth.

ACCs and hypergraph summary statistics seem to show an anomaly during the malicious activity, the autoencoder trained on the ACCs more precisely detects the malicious activity. The summary statistics show a broad range in time when the reconstruction loss is high (approximately 9:30 to 14:30) which is larger than the range occupied by the malicious activity. On the other hand, the autoencoder trained on the ACCs is able to accurately detect the first sequence of attacks with a spike in reconstruction loss from approximately 11:15 to 11:40, which closely correlates to when the first attack sequence occurred. However, there is a second spike from approximately 11:45 to 12:15 that does not correspond to an attack. We believe this is due to the large amount of activity that the red team agent is performing during this time even if it was not logged as malicious. Lastly, the autoencoder trained on the ACCs was not able to pick up the second attack which was dominated by a series of ARP scans and ping sweeps. Unfortunately, our hypergraph construction is not sensitive to this attack as many of the lines in the log data corresponding to ARP scans are not labeled with a source IP. And when the lines are associated with a source IP they are repetitive (e.g., the ping responses repeatedly have image path System and destination port 0) and do not show up as significant changes in the hypergraphs topology.

As a point of comparison we show the same zigzag and reconstruction loss plots for an exemplary benign source IP in Fig. 8. From the zigzag barcode (8a) we see that there are typically no 1-dimensional features, as evidenced by the empty D_1 barcode, for benign activity. Moreover, the 0-dimensional features have a predicatble, periodic behavior. This is further substantiated by the reconstruction loss for both the ACCs and summary statistics being very low (compare the y-axis scales in Fig. 8b to those in Fig. 7b).

V. Conclusion

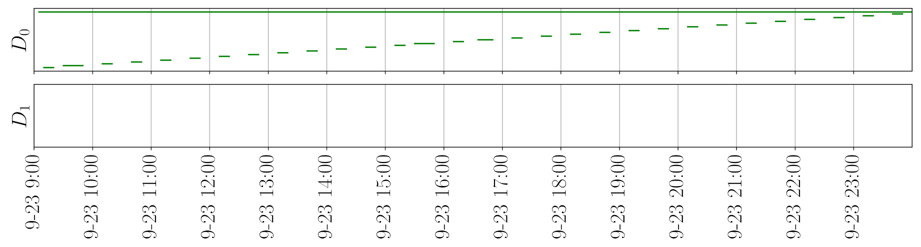
The work we present in this paper shows that the dynamics of topology of hypergraphs representing cyber log data can be effective for distinguishing malicious activity from benign. However, we have noted some limitations that we plan to explore in future work. In particular, the ACC vectorization strategy for persistence barcodes is rather coarse. We plan to evaluate more complex representations like persistence images and landscapes for this vectorization step.

Additionally, we are aware that our hypergraph construction linking executables to destination ports does not capture all types of malicious behavior. We will experiment with additional hypergraph constructions to understand how other malicious behavior can be encoded.

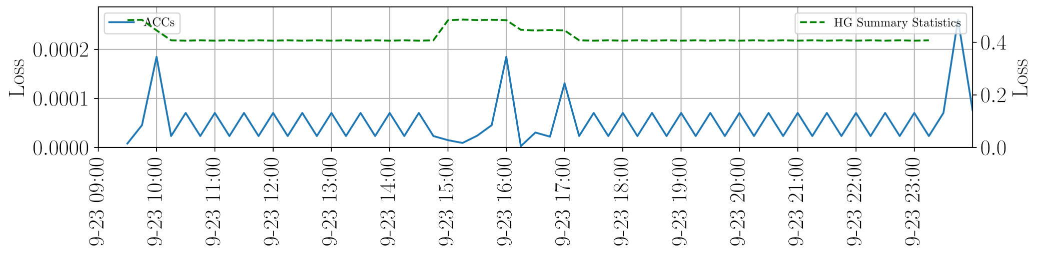
Finally, in order for cyber analysts to trust the results of our pipeline we must be able to provide some interpetation of the topological patterns in the context of the log data and ground truth malicious activity. This is ongoing work and provides an exciting opportunity for collaboration between cyber security researchers and mathematicians.

References

[1] Adams, H., Emerson, T., Kirby, M., Neville, R., Peterson, C., Shipman, P., Chepushtanova, S., Hanson, E., Motta, F., Ziegelmeier, L.:



(a) Zigzag persistence barcodes.



(b) Autoencoder reconstruction loss (mean squared error).

Fig. 8: Autoencoder results for benign source IP 142.20.56.175 on host 201 using the ACCs of the windowed zigzag persistence barcode compared to summary statistics with highlight red vertical lines for each malicious activity instance recorded in OpTC ground truth.

Persistence images: A stable vector representation of persistent homology. Journal of Machine Learning Research **18**(8), 1–35 (Jan 2017), `http://jmlr.org/papers/v18/16-337.html`

[2] Adcock, A., Carlsson, E., Carlsson, G.: The ring of algebraic functions on persistence bar codes. arXiv preprint arXiv:1304.0530 (2013)

[3] Agency, D.A.R.P.: Operationally transparent cyber (optc) data release (2020)

[4] Aksoy, S.G., Purvine, E., Young, S.J.: Directional laplacian centrality for cyber situational awareness. Digital Threats: Research and Practice (DTRAP) **2**(4), 1–28 (2021)

[5] Antelmi, A., Cordasco, G., Spagnuolo, C., Scarano, V.: A design-methodology for epidemic dynamics via time-varying hypergraphs. In: Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems. pp. 61–69 (2020)

[6] Barnes, D., Polanco, L., Perea, J.A.: A comparative study of machine learning methods for persistence diagrams. Frontiers in Artificial Intelligence **4**, 681174 (2021)

[7] Berge, C.: Hypergraphs: combinatorics of finite sets, vol. 45. Elsevier (1984)

[8] Bubenik, P.: Statistical topological data analysis using persistence landscapes. Journal of Machine Learning Research **16**(3), 77–102 (2015), `http://jmlr.org/papers/v16/bubenik15a.html`

[9] Carlsson, G., de Silva, V.: Zigzag persistence. Foundations of Computational Mathematics **10**(4), 367–405 (Apr 2010). https://doi.org/10.1007/s10208-010-9066-0

[10] Edelsbrunner, Letscher, Zomorodian: Topological persistence and simplification. Discrete &amp; Computational Geometry **28**(4), 511–533 (Nov 2002). https://doi.org/10.1007/s00454-002-2885-2

[11] Fischer, M.T., Arya, D., Streeb, D., Seebacher, D., Keim, D.A., Worring, M.: Visual analytics for temporal hypergraph model exploration. IEEE Transactions on Visualization and Computer Graphics **27**(2), 550–560 (2020)

[12] Gasparovic, E., Gommel, M., Purvine, E., Sazdanovic, R., Wang, B., Wang, Y., Ziegelmeier, L.: Homology of graphs and hypergraphs (May 2021), `https://www.youtube.com/watch?v=XeNBysFcwOw`

[13] Han, E.H., Karypis, G., Kumar, V., Mobasher, B.: Hypergraph based clustering in high-dimensional data sets: A summary of results. IEEE Data Eng. Bull. **21**(1), 15–22 (1998)

[14] Hatcher, A.: Algebraic Topology. Cambridge University Press, New York (2001)

[15] Klamt, S., Haus, U.U., Theis, F.: Hypergraphs and cellular networks. PLOS Computational Biology **5**(5), 1–6 (05 2009). https://doi.org/10.1371/journal.pcbi.1000385, `https://doi.org/10.1371/journal.pcbi.1000385`

[16] Mamun, M., Shi, K.: Deeptaskapt: insider apt detection using task-tree based deep learning. In: 2021 IEEE 20th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom). pp. 693–700. IEEE (2021)

[17] Munch, E.: A user's guide to topological data analysis. Journal of Learning Analytics **4**(2) (Jul 2017). https://doi.org/10.18608/jla.2017.42.6

[18] Myers, A., Joslyn, C., Kay, B., Purvine, E., Roek, G., Shapiro, M.: Topological analysis of temporal hypergraphs. In: Algorithms and Models for the Web Graph: 18th International Workshop, WAW 2023, Toronto, ON, Canada, May 23–26, 2023, Proceedings. pp. 127–146. Springer (2023)

[19] Myers, A., Muñoz, D., Khasawneh, F., Munch, E.: Temporal network analysis using zigzag persistence (2022)

[20] Neuhäuser, L., Lambiotte, R., Schaub, M.T.: Consensus dynamics on temporal hypergraphs. Physical Review E **104**(6), 064305 (2021)

[21] Otter, N., Porter, M.A., Tillmann, U., Grindrod, P., Harrington, H.A.: A roadmap for the computation of persistent homology. EPJ Data Science **6**(1) (Aug 2017). https://doi.org/10.1140/epjds/s13688-017-0109-5

[22] Ren, S.: Persistent homology for hypergraphs and computational tools — a survey for users. Journal of Knot Theory and Its Ramifications **29**(13), 2043007 (Nov 2020). https://doi.org/10.1142/s0218216520430075

[23] Zlatić, V., Ghoshal, G., Caldarelli, G.: Hypergraph topological quantities for tagged social networks. Physical Review E **80**(3), 036118 (2009)

[24] Zomorodian, A., Carlsson, G.: Computing persistent homology. Discrete &amp; Computational Geometry **33**(2), 249–274 (Nov 2004). https://doi.org/10.1007/s00454-004-1146-y