SK-Tree: a systematic malware detection algorithm on streaming trees via the signature kernel

Thomas Cochrane, Peter Foster, and Varun Chhabra
The Alan Turing Institute
{thomasc,pfoster,vchhabra}@turing.ac.uk

Maud Lemercier
University of Warwick
The Alan Turing Institute
maud.lemercier@warwick.ac.uk

Terry Lyons and Cristopher Salvi
University of Oxford
The Alan Turing Institute
{tlyons,salvi}@maths.ox.ac.uk

***Abstract*—The development of machine learning algorithms in the cyber security domain has been impeded by the complex, hierarchical, sequential and multimodal nature of the data involved. In this paper we introduce the notion of a *streaming tree* as a generic data structure encompassing a large portion of real-world cyber security data. Starting from host-based event logs we represent computer processes as streaming trees that evolve in continuous time. Leveraging the properties of the *signature kernel*, a machine learning tool that recently emerged as a leading technology for learning with complex sequences of data, we develop the *SK-Tree algorithm*. SK-Tree is a supervised learning method for systematic malware detection on streaming trees that is robust to irregular sampling and high dimensionality of the underlying streams. We demonstrate the effectiveness of SK-Tree to detect malicious events on a portion of the publicly available DARPA OpTC dataset, achieving an AUROC score of 98%.**

***Keywords*—** cyber security, path signature, kernel method, sequential data, tree data-structure, process tree

I. Introduction

The design and deployment of sophisticated cyber-attacks such as advanced persistent threats [1] has grown dramatically over the last few years. This has been facilitated by the appearance of new varieties of malware, and new adversary tactics and techniques, which are designed to evade existing defensive products used by enterprises such as anti-virus, firewalls and intrusion detection systems. Modern cyber security systems are generally rule-based and rely on a team of security analysts monitoring network activities and manually investigating suspected malicious activities, to determine the scope of the potential threats. This investigation phase is particularly labour-intensive. In order to defend against the influx of new malware variants and increasingly sophisticated attacks, it is imperative to develop systematic mechanisms to detect them.
One of the main challenges in creating effective and systematic malware detection systems is the complex nature of the data. The relevant datasets consist of *multimodal streams* of information, i.e. sequences of data generated by a large set of heterogeneous sources (servers, routers, workstations etc.), and representing various types of activity such as logons, file accesses, and network connections. Such data streams are often recorded at irregular intervals, span different time periods and exhibit missing observations, all aspects that make the design of systematic methods even more complicated.
A common characteristic of this data that cannot be ignored in the analysis is their *hierarchical structure*. For example, a given process may set off child processes, which themselves may spawn more children, producing a *process tree*. In this paper we account for how activity occurs within this tree-based structure over time. Specifically we represent the behaviour of computer processes, as observed in host-based event logs, as *streaming trees* evolving in continuous time. A representation of some selected channels of a single streaming tree is depicted in Fig. 1. The notion of a streaming tree we introduce is a fairly generic data structure encompassing a large portion of real-world sequential data encountered in the cyber security domain. The structure of a streaming tree is significantly more complex than that of a multivariate time series, as it accounts for the hierarchy of the data; to our knowledge no systematic method has been proposed in the literature (summarised in Sec. II) to deal with such a data structure.
Our main contribution (Sec. III) is a systematic malware detection algorithm on streaming trees that we call *SK-Tree*. The core component of SK-Tree is the *signature kernel*, a machine learning tool introduced in [2] that can process complex sequences of data. The signature kernel is based on the *path-signature* [3], [4] which is a well-known transform from stochastic analysis that recently emerged as a leading technology for learning with time series data. SK-Tree is to our knowledge the first systematic malware classification algorithm on streaming trees that is robust to irregular sampling and missing data. We test SK-Tree (Sec. V) on a portion of the DARPA's Operationally Transparent Cyber dataset [5], which is available openly and one of the largest datasets released to date, achieving an AUROC score of 98%. We make our python implementation of SK-Tree publicly available at https://github.com/crispitagorico/SK-Tree.

II. Related work

Many machine learning methods developed in the cyber security literature have focused on the processing of network data, with only few techniques having been designed to consume host event data as we do in this work. Among publicly-available datasets, two released by Los Alamos National Laboratory (LANL) are the most widely used [6], [7]. As far as

978-1-7281-5684-2/20/$31.00 ©2021 IEEE

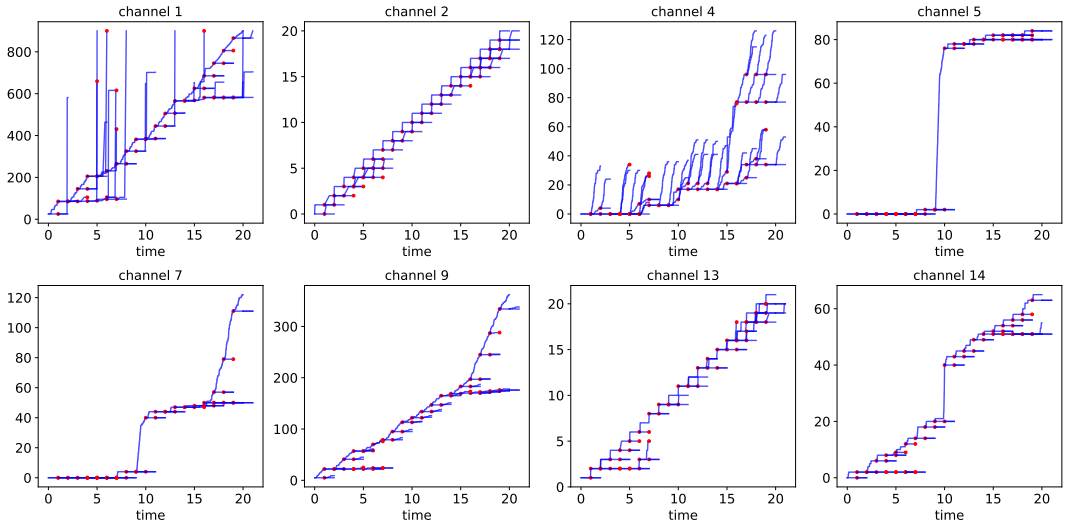

Fig. 1. Visual representation of selected channels of one single streaming tree. Each plot represents the evolution in time of the value of a given channel of the streaming tree, on its various branches. A red dot indicates a point where the currently-tracked process sets off a child process, causing the tree to branch.

we are aware, our work is the first to make use of the DARPA OpTC dataset [5]. Compared to earlier datasets, events in OpTC give a much more detailed summary of host activity, and contain more varied red-team behaviour. The heterogeneity of the data is one of the major bottlenecks.

On the one hand, some of the existing techniques aim to detect malicious activity from streams of events without taking into account any hierarchical structure. There has been a proliferation of work in anomaly detection for the purposes of finding malicious activity [8]–[14]. In particular [10] took a host-based approach and considered the authentication events in the 2015 LANL dataset using a RNN language model. Auto-encoders were used in [15] to recognise lateral movement through a network, using manually engineered features based on conditional probabilities defined by event counts.

On the other hand, another line of thought tries to detect malware or suspicious behaviour using the graph-like structure of the data, but without taking into account the sequential nature of the events. Utilised features include connections between computers on a network or relations of processes to those that they spawn. Tree and graph kernels were first introduced for malware analysis in [16], comparing the similarity of subtrees and subgraphs respectively, and achieving promising results on data obtained from a honeypot. Process trees were further studied in [17], where structures from process trees were compared against reference trees known to be non-malicious. A similar technique was used by [18], comparing star structures with templates extracted from uninfected trees.

However, none of the aforementioned approaches are designed to process complex data structures that are hierarchical and sequential simultaneously. The main objective of this paper is to provide a general procedure flexible enough to handle such complex data structures, formalised in the next section.

III. Method

A. Streaming trees

A *multivariate time series* $\mathbf{x}$ *of dimension* $d \in \mathbb{N}$ is a collection of points $x_i \in \mathbb{R}^{d-1}$ with corresponding time-stamps $t_i \in \mathbb{R}$ such that $t_0 < ... < t_n$ and defined as follows

$$\mathbf{x} = ((t_0, x_0), (t_1, x_1), ..., (t_n, x_n)) \tag{1}$$

A *tree* τ is defined recursively as a tuple of the form $\tau = (\mathbf{x}_\tau, F_\tau)$, where $\mathbf{x}_\tau$ is a multivariate time series and $F_\tau = (\tau_1, ..., \tau_k)$ is a (possibly empty) collection of trees. Concatenating to the time series $\mathbf{x}_\tau$ of a tree τ the values of the time series $\mathbf{x}_{\tau_1}, ..., \mathbf{x}_{\tau_k}$ of its children subtrees $\tau_1, ..., \tau_k$, and repeating this operation recursively for each children subtree τ_i, we end up with an equivalent representation of the tree τ as a collection of branches $\tau = (\mathbf{x}_\tau^1, ..., \mathbf{x}_\tau^n)$, where each branch $\mathbf{x}_\tau^i$ is a multivariate time series starting at t_0. For the sequel, it is important to keep both representations in mind.

Example 1. Consider the tree $\tau = (\mathbf{x}_\tau, ((\mathbf{x}_{\tau_1}, \varnothing), (\mathbf{x}_{\tau_2}, \varnothing)))$, where $\varnothing$ denotes the empty list and where $\mathbf{x}_\tau, \mathbf{x}_{\tau_1}, \mathbf{x}_{\tau_2}$ are the following multivariate time series of dimension d

$$\begin{aligned}
\mathbf{x}_\tau &= ((t_0, x_0), (t_1, x_1), ..., (t_n, x_n)) \\
\mathbf{x}_{\tau_1} &= ((t_{n+1}, y_0), (t_{n+2}, y_1), ..., (t_{n+i}, y_{i-1})) \\
\mathbf{x}_{\tau_2} &= ((t_{n+1}, z_0), (t_{n+2}, z_1), ..., (t_{n+j}, z_{j-1}))
\end{aligned}$$

It is easy to see that τ has two branches. Indeed, consider a first multivariate time series of dimension d

$$\mathbf{x}_\tau^1 = ((t_0, x_0), (t_1, x_1), ..., (t_{n+i}, x_{n+i}))$$

where

$$x_k = \begin{cases} x_k, & \text{if } k \leq n, \\ y_{k-n-1}, & \text{if } n+1 \leq k \leq n+i \end{cases}$$

and a second multivariate time series of dimension d

$$\mathbf{x}_\tau^2 = ((t_0, x_0), (t_1, x_1), ..., (t_{n+j}, x_{n+j}))$$

where

$$x_k = \begin{cases} x_k, & \text{if } k \leq n, \\ z_{k-n-1}, & \text{if } n+1 \leq k \leq n+j \end{cases}$$

Then, τ has the equivalent representation $\tau = (\mathbf{x}_\tau^1, \mathbf{x}_\tau^2)$ in terms of its two branches $\mathbf{x}_\tau^1, \mathbf{x}_\tau^2$.

Given a multivariate time series $\mathbf{x}$ of the form of eq. (1), define the *path* $X : [t_0, t_n] \to \mathbb{R}^d$ as the the continous piecewise linear interpolation of the data such that $X_{t_i} = (t_i, x_i)$. A *streaming tree* $\mathcal{T}$ is the data structure obtained by replacing all the time series appearing in the definition of a tree τ (and in all its children subtrees) by their continuous piecewise linear interpolation. Analogously to the equivalent representation of a tree in terms of its branches that we discussed above, a streaming tree $\mathcal{T}$ can also be represented as a collection of branches $\mathcal{T} = (X_\mathcal{T}^1, ..., X_\mathcal{T}^n)$ where each branch $X_\mathcal{T}^i$ is the continuous piecewise linear interpolation of the i^{th} branch $\mathbf{x}_\tau^i$ of the tree τ. The resulting branches $X_\mathcal{T}^1, ..., X_\mathcal{T}^n$ form a collection of paths with common history.

Example 2. Consider again the same tree as in Example 1, i.e. $\tau = (\mathbf{x}_\tau, ((\mathbf{x}_{\tau_1}, \varnothing), (\mathbf{x}_{\tau_2}, \varnothing)))$. We saw that τ can be represented in terms of its two branches as $\tau = (\mathbf{x}_\tau^1, \mathbf{x}_\tau^2)$. Let $X_\mathcal{T}^1 : [t_0, t_{n+i}] \to \mathbb{R}^d$ and $X_\mathcal{T}^2 : [t_0, t_{n+j}] \to \mathbb{R}^d$ be the continuous piecewise linear interpolation of $\mathbf{x}_\tau^1, \mathbf{x}_\tau^2$ respectively. Then the streaming tree $\mathcal{T}$ can be represented in terms of its two branches as $\mathcal{T} = (X_\mathcal{T}^1, X_\mathcal{T}^2)$.

B. The signature

Here we describe a well-known path-transform called the *signature* that allows us to extract meaningful features from a multivariate time series in a systematic way. The signature has been deployed as a machine learning tool in many data science applications dealing with sequential data [19]–[21].

For any coordinate $\alpha \in \{1, ..., d\}$ and any continuous piecewise linear path $X : [0, T] \to \mathbb{R}^d$ we denote its α^{th} *channel* by $X^{(\alpha)}$ so that at any time $t \in [0, T]$

$$X(t) = (X^{(1)}(t), ..., X^{(d)}(t)). \tag{2}$$

We denote by $\mathcal{X}$ the set of all continuous piecewise linear paths with values in $\mathbb{R}^d$. The signature $S : \mathcal{X} \to H$ is a *feature map* defined for any path $X \in \mathcal{X}$ as the following infinite collection of features [3]

$$S(X) = \Big(1, \Big\{S(X)^{(\alpha_1)}\Big\}_{\alpha_1=1}^d, \Big\{S(X)^{(\alpha_1,\alpha_2)}\Big\}_{\alpha_1,\alpha_2=1}^d, \Big\{S(X)^{(\alpha_1,\alpha_2,\alpha_3)}\Big\}_{\alpha_1,\alpha_2,\alpha_3=1}^d, ...\Big) \tag{3}$$

where every term is a scalar defined as the iterated integral

$$S(X)^{(\alpha_1,...,\alpha_j)} = \int\limits_{0<s_1<...<s_j<T} ... \int dX^{(\alpha_1)}(s_1)...dX^{(\alpha_j)}(s_j) \tag{4}$$

The *feature space* H associated to the signature is a Hilbert space defined as the direct sum of tensor powers of $\mathbb{R}^d$

$$H = \bigoplus_{k=0}^{\infty} (\mathbb{R}^d)^{\otimes k} = \mathbb{R} \oplus \mathbb{R}^d \oplus (\mathbb{R}^d)^{\otimes 2} \oplus ... \tag{5}$$

where $\otimes$ denotes the outer product [4].

C. The expected signature

Here we describe another transform for sequential data called the *expected signature* that generalises the signature in the sense that it allows to extract useful features from an ensemble of time series. The sequence of moments $(\mathbb{E}[Z^{\otimes m}])_{m\geq 0}$ of any finite dimensional random variable Z is classically known to characterize its law $\mu_Z = \mathbb{P} \circ Z^{-1}$. It turns out that in the infinite dimensional case of path-valued random variables an analogous result holds [22]; it says that one can fully characterise such path-valued random variables by replacing moments by the expected signature, that we define next.

Assume that $\mathcal{X}$ is compact and let μ be a probability measure supported on $\mathcal{X}$. The *expected signature* of μ is defined as the following infinite collection of statistics

$$\mathbb{E}_S(\mu) = \Big(1, \Big\{\mathbb{E}_S(\mu)^{(\alpha_1)}\Big\}_{\alpha_1=1}^d, \Big\{\mathbb{E}_S(\mu)^{(\alpha_1,\alpha_2)}\Big\}_{\alpha_1,\alpha_2=1}^d, \Big\{\mathbb{E}_S(\mu)^{(\alpha_1,\alpha_2,\alpha_3)}\Big\}_{\alpha_1,\alpha_2,\alpha_3=1}^d, ...\Big)$$

where each term is a scalar defined as the following integral

$$\mathbb{E}_S(\mu)^{(\alpha_1,...,\alpha_j)} = \int_{X\in\mathcal{X}} S(X)^{(\alpha_1,...,\alpha_k)} \mu(dX). \tag{6}$$

As explained in section III-A, a streaming tree $\mathcal{T}$ can be represented in terms of its branches as a collection of continuous piecewise linear paths $\mathcal{T} = (X_\mathcal{T}^1, ..., X_\mathcal{T}^n)$. Denoting by $\delta_{X_\mathcal{T}^i}$ the *Dirac measure* indexed on the path $X_\mathcal{T}^i$ we can represent the streaming tree $\mathcal{T}$ as the following *empirical measure*

$$\mu_\mathcal{T} = \frac{1}{n} \sum_{i=1}^n \delta_{X_\mathcal{T}^i} \tag{7}$$

We refer the reader to [23] for further details on the probabilistic setup. Therefore, any streaming tree $\mathcal{T}$ can be faithfully represented by means of its expected signature $\mathbb{E}_S(\mu_\mathcal{T})$. Given the special recursive structure of a streaming tree as collection of paths with common history, the expected signature $\mathbb{E}_S(\mu_\mathcal{T})$ can be computed via a convenient recursive formula: consider a streaming tree $\mathcal{T} = (X_\mathcal{T}, F_\mathcal{T})$, where $F_\mathcal{T} = (\mathcal{T}_1, ..., \mathcal{T}_n)$ is a (possibly empty) list of streaming trees. Then, the expected signature $\mathbb{E}_S(\mu_\mathcal{T})$ satisfies the following recursive formula

$$\mathbb{E}_S(\mu_\mathcal{T}) = \begin{cases} S(X_\mathcal{T}), & \text{if } F_\mathcal{T} = \varnothing, \\ \frac{1}{n}\sum_{i=1}^n S(X_\mathcal{T}) \otimes \mathbb{E}_S(\mu_{\mathcal{T}_i}), & \text{otherwise} \end{cases} \tag{8}$$

where $S(X_\mathcal{T})$ is the signature of the path $X_\mathcal{T}$, $\varnothing$ is the empty list, $\otimes$ is the outer product and $\mathbb{E}_S(\mu_{\mathcal{T}_i})$ is the expected signature of the streaming tree $\mathcal{T}_i$.

D. A measure of similarity between streaming trees

Consider two streaming trees $\mathcal{T}_1$ and $\mathcal{T}_2$ with corresponding probability measures $\mu_{\mathcal{T}_1}$ and $\mu_{\mathcal{T}_2}$. An appropriate measure of similarity between the trees $\mathcal{T}_1$ and $\mathcal{T}_2$ can be obtained by considering a distance between the probability measures $\mu_{\mathcal{T}_1}$ and $\mu_{\mathcal{T}_2}$. The *maximum mean discrepancy* (MMD) distance between $\mu_{\mathcal{T}_1}$ and $\mu_{\mathcal{T}_2}$ is defined as

$$d_{\text{MMD}}(\mu_{\mathcal{T}_1}, \mu_{\mathcal{T}_2}) = \sup_{f \in \mathcal{G}} \left| \mathbb{E}_{\mu_{\mathcal{T}_1}}[f(X_{\mathcal{T}_1})] - \mathbb{E}_{\mu_{\mathcal{T}_2}}[f(X_{\mathcal{T}_2})] \right| \quad (9)$$

where $X_{\mathcal{T}_i} \sim \mu_{\mathcal{T}_i}$ is a sample path from the probability measure $\mu_{\mathcal{T}_i}$ (for $i = 1, 2$), where $\mathcal{G}$ is a space of real valued functions on the path space $\mathcal{X}$ defined as the unit ball of the *reproducing kernel Hilbert space* (RKHS) $\mathcal{H}_k$ associated to an appropriate *kernel on paths* $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, in other words

$$\mathcal{G} = \{f : \mathcal{X} \to \mathbb{R} : ||f||_{\mathcal{H}_k} \leq 1\} \quad (10)$$

For a detailed account of RKHSs and MMD distance we refer the interested reader to [24]. Thanks to the main result in [24], the MMD distance of eq. (9) can be expressed as the sum of the following three terms

$$\begin{aligned} d_{\text{MMD}}(\mu_{\mathcal{T}_1}, \mu_{\mathcal{T}_2})^2 &= \mathbb{E}_{(\mu_{\mathcal{T}_1}, \mu_{\mathcal{T}_1})}[k(X_{\mathcal{T}_1}, \widetilde{X}_{\mathcal{T}_1})] \\ &+ \mathbb{E}_{(\mu_{\mathcal{T}_2}, \mu_{\mathcal{T}_2})}[k(X_{\mathcal{T}_2}, \widetilde{X}_{\mathcal{T}_2})] \\ &- 2\mathbb{E}_{(\mu_{\mathcal{T}_1}, \mu_{\mathcal{T}_2})}[k(X_{\mathcal{T}_1}, X_{\mathcal{T}_2})] \end{aligned} \quad (11)$$

where each term is expressed in terms of sample paths from the underlying measures $\mu_{\mathcal{T}_1}, \mu_{\mathcal{T}_2}$ and kernel evaluations on those samples. If the streaming tree $\mathcal{T}_1 = (X^1_{\mathcal{T}_1}, ..., X^m_{\mathcal{T}_1})$ has m branches and the streaming tree $\mathcal{T}_2 = (X^1_{\mathcal{T}_2}, ..., X^n_{\mathcal{T}_2})$ has n branches, the MMD distance can be computed explicitly as

$$\begin{aligned} d_{\text{MMD}}(\mu_{\mathcal{T}_1}, \mu_{\mathcal{T}_2})^2 &= \frac{1}{m(m-1)} \sum_{i=1}^{m} \sum_{j \neq i}^{m} k(X^i_{\mathcal{T}_1}, X^j_{\mathcal{T}_1}) \\ &+ \frac{1}{n(n-1)} \sum_{i=1}^{n} \sum_{j \neq i}^{n} k(X^i_{\mathcal{T}_2}, X^j_{\mathcal{T}_2}) \\ &- \frac{2}{mn} \sum_{i=1}^{m} \sum_{j=1}^{n} k(X^i_{\mathcal{T}_1}, X^j_{\mathcal{T}_2}) \end{aligned} \quad (12)$$

Therefore, in order to quantify the similarity between two streaming trees $\mathcal{T}_1, \mathcal{T}_2$ using (12) it is paramount to specify an appropriate kernel k acting on paths from $\mathcal{X}$.

E. The signature kernel

The *signature kernel* $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a reproducing kernel associated to the signature feature map S and defined for any pair of paths $X : [0, T] \to \mathbb{R}^d$ and $Y : [0, T] \to \mathbb{R}^d$ as the following inner product [22]

$$k(X, Y) = \langle S(X), S(Y) \rangle_H \quad (13)$$

In the recent article [25] the authors show that the signature kernel can be computed explicitly; they provide a *kernel trick* for the signature kernel by proving the following relation

$$k(X, Y) = U(T, T) \quad (14)$$

where the function $U : [0, T] \times [0, T] \to \mathbb{R}$ is the solution of the following *partial differential equation* (PDE)

$$\frac{\partial^2 U}{\partial s \partial t} = \frac{\partial^2 \kappa(X(s), Y(t))}{\partial s \partial t} U \quad (15)$$

with boundary conditions $U(0, \cdot) = 1$ and $U(\cdot, 0) = 1$, where $\kappa : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is any base kernel on $\mathbb{R}^d$. In this paper we will be using the *Gaussian RBF kernel* as base kernel κ. Therefore, the MMD distance between two streaming trees $\mathcal{T}_1, \mathcal{T}_2$ can be computed explicitly via eq. (12). We will make use of this distance in our malware detection algorithm SK-Tree that we introduce next.

IV. SK-Tree: A kernel based malware detection algorithm

In [23] the authors construct a universal kernel indexed on probability measures on paths defined as a combination of the signature kernel, MMD distance of eq. (9), and a Gaussian kernel. The formulation of this kernel is given in eq. (17) below. Our final algorithm (SK-Tree) will simply consist of a *support vector machine* (SVM) classifier [26] equipped with this kernel, as we shall outline next.

Given the representation of a streaming tree as a probability measure on paths, we can leverage their construction and propose the following kernel for streaming trees: let $\mathcal{T}_1 = (X^1_{\mathcal{T}_1}, ..., X^m_{\mathcal{T}_1})$ and $\mathcal{T}_2 = (X^1_{\mathcal{T}_1}, ..., X^n_{\mathcal{T}_1})$ be two streaming trees with associated probability measures $\mu_{\mathcal{T}_1}, \mu_{\mathcal{T}_2}$, respectively. We define the kernel k_σ on streaming trees as

$$k_\sigma(\mu_{\mathcal{T}_1}, \mu_{\mathcal{T}_2}) = \exp\left(-\sigma^2 d_{\text{MMD}}(\mu_{\mathcal{T}_1}, \mu_{\mathcal{T}_2})^2\right) \quad (16)$$

where $\sigma > 0$ is a scalar hyperparameter. By eq. (12), the kernel k_σ can be explicitly computed by evaluating the signature kernel k at the branches of $\mathcal{T}_1, \mathcal{T}_2$ as follows

$$\begin{aligned} k_\sigma(\mu_{\mathcal{T}_1}, \mu_{\mathcal{T}_2}) = \exp\Big(&- \frac{\sigma^2}{m(m-1)} \sum_{i=1}^{m} \sum_{j \neq i}^{m} k(X^i_{\mathcal{T}_1}, X^j_{\mathcal{T}_1}) \\ &- \frac{\sigma^2}{n(n-1)} \sum_{i=1}^{n} \sum_{j \neq i}^{n} k(X^i_{\mathcal{T}_2}, X^j_{\mathcal{T}_2}) \\ &+ \frac{2\sigma^2}{mn} \sum_{i=1}^{m} \sum_{j=1}^{n} k(X^i_{\mathcal{T}_1}, X^j_{\mathcal{T}_2}) \Big) \end{aligned} \quad (17)$$

Performing classification (or regression) tasks on streaming trees is easily achieved thanks to the explicit expression of the kernel k_σ as per eq. (17). In this paper we are interested in binary classification of streaming trees as non-malicious (class 0) or malicious (class 1) events. We are given a dataset of input-output pairs $\{\mathcal{T}_i, y_i\}_{i=1}^M$ where the inputs $\mathcal{T}_i$ are streaming trees and the outputs y_i are in $\{0, 1\}$. We note that it is straightforward to extend the binary SVM classifier (SK-Tree) to a multi-class SVM to classify different types of malware[1]. To carry out the classification we make use of a *support vector machine* (SVM) classifier [26] equipped with

[1] See https://scikit-learn.org/stable/modules/svm.html for more details

the kernel k_σ on streaming trees of eq. (17). The binary SVM classification algorithm aims at solving the minimisation

$$\min_{f\in\mathcal{H}_{k_\sigma}} \sum_{i=1}^{M} L(y_i, f(\mathcal{T}_i)) + \lambda||f||_{\mathcal{H}_{k_\sigma}} \tag{18}$$

where $L(y_i, f(x_i)) = \max(0, 1 - y_i f(x_i))$, and λ is the penalty hyperparameter. Following [27], the optimal solution to this minimisation can be expressed in terms of the kernel k_σ as follows: for any streaming tree $\mathcal{T}$

$$f^*(\mathcal{T}) = sgn\Big(\alpha_0 + \sum_{i=1}^{M} y_i \alpha_i k_\sigma(\mathcal{T}, \mathcal{T}_i)\Big) \tag{19}$$

where α_i are scalar coefficients computed from solving a quadratic programming problem. Our algorithm, that we call SK-Tree, consists of an SVM classifier that uses the Gram matrix associated to the kernel k_σ and computed via Algorithm 1. In the latter, PDESolve stands for a call to a PDE solver to evaluate the signature kernel k (see Sec. III-E) and the notation $0_{m\times n}$ indicates the zero-matrix in $\mathbb{R}^{m\times n}$. We implemented the full SK-Tree algorithm as a ready-to-use estimator using the popular python library scikit-learn [28] and we make it publicly available at https://github.com/crispitagorico/SK-Tree.

Algorithm 1 SK-Tree Gram matrix

Input: M str. trees $\left\{\mathcal{T}_i = \left(X^1_{\mathcal{T}_i}, ..., X^{k_i}_{\mathcal{T}_i}\right)\right\}_{i=1}^{M}$, $\sigma > 0$
Initialize $G \leftarrow 0_{M\times M}$
for each pair $(i, j) \in \{1, ..., M\}^2$ **do**
  Initialize $K_1 \leftarrow 0_{k_i\times k_i}, K_2 \leftarrow 0_{k_j\times k_j}, K_3 \leftarrow 0_{k_i\times k_j}$
  for $(p, q) \in \{1, ..., k_i\}^2$ **do**
    $K_1[p, q] \leftarrow \text{PDESolve}(X^p_{\mathcal{T}_i}, X^q_{\mathcal{T}_i})$
  for $(p, q) \in \{1, ..., k_j\}^2$ **do**
    $K_2[p, q] \leftarrow \text{PDESolve}(X^p_{\mathcal{T}_j}, X^q_{\mathcal{T}_j})$
  for $(p, q) \in \{1, ..., k_i\} \times \{1, ..., k_j\}$ **do**
    $K_3[p, q] \leftarrow \text{PDESolve}(X^p_{\mathcal{T}_i}, X^q_{\mathcal{T}_j})$
  $G[i, j] \leftarrow \frac{\text{sum}(K_1)}{k_i(k_i-1)} + \frac{\text{sum}(K_2)}{k_j(k_j-1)} - 2 \times \frac{\text{sum}(K_3)}{k_i k_j}$
$G \leftarrow \exp(-\sigma^2 G)$
Output: The Gram matrix G.

V. Data and experiments

The Operationally Transparent Cyber (OpTC) dataset [5] was released by DARPA in June 2020. It consists of data collected from an isolated network of 1000 hosts over a multi-day period. We use OpTC's *ecar* data, which records endpoint activity in an extended CAR format based on MITRE's CAR data model [29]. For example, a process creation event appears as follows (some fields omitted for clarity):

```
{"action":"CREATE",
"actorID":"437acfc7-d9ef-4c60-a108-...",
"hostname":"SysClient0201.systemia.com",
"object":"PROCESS",
"objectID":"b9d06a48-0968-4bda-b743-...",
"properties":...,
"timestamp":1569245579591}
```

Here `actorID` and `objectID` uniquely identify the parent and child processes. In events of other types, such as file and network activity, the `actorID` similarly provides a unique identifier of the process responsible for the event.

A. Modelling the data as streaming trees

These events can be interpreted as trees in a natural way, where each branch follows the events that are associated with a particular process. At a process creation event, the tree splits into two branches: one following the continuing parent process, the other following the new child process. Specifically, we produce a tree in 23 dimensions:

- The first dimension represents time.
- The next two dimensions encode the structure of the process tree: they represent the depth in the tree, and the number of processes that this branch has set off. At a process creation event, the child branch and parent branch immediately increment in these dimensions respectively.
- We take 20 other types of events defined by (object, action) pairs, eg (MODULE, LOAD) or (FILE, DELETE). Each event type is associated a dimension. Whenever an event of this type is observed along a branch, the branch increments in the appropriate dimension.

The method easily extends to other sources of host data: the only requirement is that process creation events are logged (including the identity of the parent and child processes), enabling a process tree to be built. Even in the OpTC *ecar* data there is much more information available in the `properties` of each event: e.g. the identities of processes and files; source and destination ports of network connections. We currently ignore these properties, but they could be used to select a set of event types that better summarises the data.

B. Results

The vast majority of activity in the OpTC dataset is benign: this activity continues throughout the period. However on three days, red-team attackers are active on the network, providing a source of known malicious activity that we aim to detect. The attackers' actions are detailed in a PDF document that accompanies the OpTC dataset. We take all processes that correspond to activity listed in the document, together with their descendants, and mark these processes as malicious.

To form the trees, we take all activity on the first day of malicious activity from host 0201 (since this host has the greatest proportion of malicious activity). We split each process into 15-minute windows of activity and create a streaming tree for each. The associated label is 1 if the root process is malicious, 0 otherwise. All branches are scaled to have mean 0, sd 1. We discard any tree with fewer than 2 or more than 200 events. As a result, the final dataset contains $4\,199$ streaming trees. The SK-Tree binary classifier is run on random 5 fold train-test partitions and we report the mean and standard deviation of the AUROC score on Fig. 2. The hyperparameters of SK-Tree are selected by cross-validation via a grid search on the training set of each fold. As it can be observed, we achieve a 98% AUROC score. The high AUROC rate implies a good

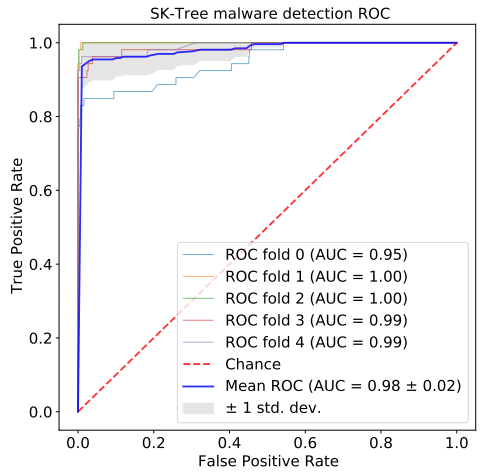

Fig. 2. ROC evaluation of the SK-Tree binary classifier on the OpTC data

performance both in terms of absolute accuracy and of false positive/negative rates detection. In future work it would be interesting to extend these experiments to multi-class malware detection and to other hosts of the OpTC dataset.

VI. Conclusion

In this paper we introduced the notion of a *streaming tree* to describe computer process activity as a generic data structure that encompasses most of the complex, hierarchical, sequential and multimodal nature of the data involved. We then introduced SK-Tree, a new supervised learning method for systematic malware detection on streaming trees that is robust to the irregular sampling and high dimensionality of the underlying streams. SK-Tree is based on the signature kernel. We finally demonstrated the effectiveness of SK-Tree at detecting malicious events on a portion of the publicly available OpTC dataset [5] achieving a AUROC score of 98%.

Acknowledgement

All authors were supported by the Alan Turing Institute under the EPSRC grant EP/N510129/1 and the D&S Programme and by DataSig under the EPSRC grant EP/S026347/1.

References

[1] P. Chen, L. Desmet, and C. Huygens, "A study on advanced persistent threats," in *IFIP International Conference on Communications and Multimedia Security*. Springer, 2014, pp. 63–72.

[2] F. J. Király and H. Oberhauser, "Kernels for sequentially ordered data," *Journal of Machine Learning Research*, vol. 20, no. 31, pp. 1–45, 2019.

[3] I. Chevyrev and A. Kormilitzin, "A primer on the signature method in machine learning," *arXiv preprint arXiv:1603.03788*, 2016.

[4] T. J. Lyons, M. Caruana, and T. Lévy, *Differential equations driven by rough paths*. Springer, 2007.

[5] DARPA. (2020) Operationally Transparent Cyber data release. [Online]. Available: https://github.com/FiveDirections/OpTC-data

[6] A. D. Kent, "Comprehensive, Multi-Source Cyber-Security Events," Los Alamos National Laboratory, 2015.

[7] M. J. M. Turcotte, A. D. Kent, and C. Hash, *Unified Host and Network Data Set*. World Scientific, Nov 2018, ch. 1, pp. 1–22.

[8] A. Walker and S. Sengupta, "Malware family fingerprinting through behavioral analysis," in *2020 IEEE International Conference on Intelligence and Security Informatics (ISI)*. IEEE, 2020, pp. 1–5.

[9] M. Whitehouse, M. Evangelou, and N. M. Adams, "Activity-based temporal anomaly detection in enterprise-cyber security," in *2016 IEEE Conference on Intelligence and Security Informatics (ISI)*. IEEE, 2016, pp. 248–250.

[10] A. Brown, A. Tuor, B. Hutchinson, and N. Nichols, "Recurrent neural network attention mechanisms for interpretable system log anomaly detection," in *Proceedings of the First Workshop on Machine Learning for Computing Systems*, 2018, pp. 1–8.

[11] E. Riddle-Workman, M. Evangelou, and N. M. Adams, "Adaptive anomaly detection on network data streams," in *2018 IEEE International Conference on Intelligence and Security Informatics (ISI)*. IEEE, 2018, pp. 19–24.

[12] M. E. Eren, J. S. Moore, and B. S. Alexandro, "Multi-dimensional anomalous entity detection via poisson tensor factorization," in *2020 IEEE International Conference on Intelligence and Security Informatics (ISI)*. IEEE, 2020, pp. 1–6.

[13] A. Tuor, S. Kaplan, B. Hutchinson, N. Nichols, and S. Robinson, "Deep learning for unsupervised insider threat detection in structured cybersecurity data streams," *arXiv preprint arXiv:1710.00811*, 2017.

[14] F. S. Passino and N. A. Heard, "Classification of periodic arrivals in event time data for filtering computer network traffic," *Statistics and Computing*, vol. 30, no. 5, pp. 1241–1254, 2020.

[15] R. Holt, S. Aubrey, A. DeVille, W. Haight, T. Gary, and Q. Wang, "Deep autoencoder neural networks for detecting lateral movement in computer networks," in *Proceedings on the International Conference on Artificial Intelligence (ICAI)*. The Steering Committee of The World Congress in Computer Science, Computer ..., 2019, pp. 277–283.

[16] C. Wagner, G. Wagener, R. State, and T. Engel, "Malware analysis with graph kernels and support vector machines," in *2009 4th International Conference on Malicious and Unwanted Software (Malware)*. IEEE, 2009, pp. 63–68.

[17] K. Wijands, "Detecting malware using process tree and process activity data," Master's thesis, Technische Universiteit Delft, 2015.

[18] R. Luh and S. Schrittwieser, "Advanced threat intelligence: detection and classification of anomalous behaviour in system processes," *Elektrotechnik & Informationstechnik*, vol. 137, pp. 38–44, 2020.

[19] I. P. Arribas, G. M. Goodwin, J. R. Geddes, T. Lyons, and K. E. Saunders, "A signature-based machine learning model for distinguishing bipolar disorder and borderline personality disorder," *Translational psychiatry*, vol. 8, no. 1, pp. 1–7, 2018.

[20] J. Kalsi, T. Lyons, and I. P. Arribas, "Optimal execution with rough path signatures," *SIAM Journal on Financial Mathematics*, vol. 11, no. 2, pp. 470–493, 2020.

[21] P. Moore, T. Lyons, J. Gallacher, and A. D. N. Initiative, "Using path signatures to predict a diagnosis of alzheimer's disease," *PloS one*, vol. 14, no. 9, p. e0222212, 2019.

[22] I. Chevyrev and H. Oberhauser, "Signature moments to characterize laws of stochastic processes," *arXiv preprint arXiv:1810.10971*, 2018.

[23] M. Lemercier, C. Salvi, T. Damoulas, E. V. Bonilla, and T. Lyons, "Distribution regression for continuous-time processes via the expected signature," in *Artificial Intelligence and Statistics (AISTATS)*, 2021.

[24] A. Gretton, K. M. Borgwardt, M. J. Rasch, B. Schölkopf, and A. Smola, "A kernel two-sample test," *Journal of Machine Learning Research*, vol. 13, no. Mar, pp. 723–773, 2012.

[25] T. Cass, T. Lyons, C. Salvi, and W. Yang, "The signature kernel is the solution of a goursat pde," *arXiv preprint arXiv:2006.14794*, 2020.

[26] L. Wang, *Support vector machines: theory and applications*. Springer Science & Business Media, 2005, vol. 177.

[27] B. Scholkopf and A. J. Smola, *Learning with kernels: support vector machines, regularization, optimization, and beyond*. Adaptive Computation and Machine Learning series, 2018.

[28] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[29] MITRE. CAR data model. [Online]. Available: https://car.mitre.org/data_model/