Computers & Security 98 (2020) 102002

Contents lists available at ScienceDirect

Computers & Security

journal homepage: www.elsevier.com/locate/cose

procmonML: Generating evasion resilient host-based behavioral analytics from tree ensembles

Joseph W. Mikhail*, Jamie C. Williams, George R. Roelke

The MITRE Corporation, United States

ARTICLE INFO

Article history:
Received 4 June 2020
Revised 22 July 2020
Accepted 12 August 2020
Available online 19 August 2020

Keywords:
Cyber analytics
Tree ensemble
Host process data
Evasion
Advanced adversary threats

ABSTRACT

Host-based analytics are useful for identifying nefarious activity and limiting the impact of an adversary's cyber attack on an endpoint. The majority of open-source host-based analytics are heuristic in nature and often rely on matching combinations of strings to produce an alert. Recent threat reports demonstrate that threat actors are able to easily evade these types of analytics via variances in attack techniques, implementation differences, or simple string/parameter modifications. This work introduces a novel machine learning-based approach (procmonML) to generate true behavioral host-based analytics that are more resilient to adversary evasion, thus imparting more workload on the adversary to successfully evade detection. This is accomplished by consolidating multiple system events into a single process event. Analytics are generated from a tree ensemble model using labeled host data from a lab environment and are validated on production enterprise endpoints. This approach can detect multiple variations of a single attack technique by capturing and generalizing system behaviors. The results demonstrate that the procmonML approach is able to effectively generate host-based analytics that are applicable to new environments and more resilient to adversary evasion.

1. Introduction

Intrusion detection for computer endpoints, also known as host-based detection, is a key capability that helps disrupt an adversary's cyber kill chain. Malware identification and classification is one approach to detecting adversary activity on the endpoint. However, in recent years, defensive efforts have also focused on understanding adversary behaviors. The MITRE ATT&CK® Framework is a globally accessible knowledge base of offensive behaviors used by real adversaries and provides a common language to describe and reference adversary tactics, techniques, and procedures (TTPs) (Strom et al., 2017). A tactic is the adversary's goal for performing an action, while a technique represents how the adversary is able to achieve the goal. A procedure represents the actions that the adversary performs, or how a given technique is implemented.

Recent reports show there are a growing number of fileless attacks on computer systems that utilize common benign system processes to facilitate malicious activity (Baldin, 2019). These attacks also feature "living off the land" techniques involving the abuse of an otherwise benign system asset or capability, rather than the historical attack vector that uses a malicious portable executable (PE) file. There are numerous instances of fileless malware that use legitimate system commands for underlying malicious purposes (Kumar, 2020). There is a significant challenge in detecting these types of attacks since legitimate system resources are used. For example, while PowerShell makes a system administrator's job easier, malicious scripting via PowerShell is a popular technique amongst attackers since it provides access to many core kernel resources (Baldin, 2019). With fileless attacks, there is also less chance of being detected by a network sensor (Mansfield-Devine, 2017), as this reduces an attacker's network footprint to solely command and control (C2) traffic. While fileless cyber attacks still require initial access to get into the system, with social engineering or an exploit on external facing infrastructure, network sensing tools are often blind to an attacker operating on an endpoint. Once an attacker is able infiltrate a host, attackers can then employ persistence techniques to ensure that they can remain in the system over a period of time.

State-of-the-art host intrusion detection systems (HIDS) aim to detect malware, adversary TTPs, and malicious system behaviors. Two openly available host-based data sources used to aid in the detection of endpoint attacks are Windows Security Event Logs and Microsoft (MS) Sysmon (Russinovich, 2009). These data sources are typically ingested into an organization's enterprise security information and event management (SIEM) system where cyber defenders can run text-based analytics to search for attacks across multiple endpoints (Mavroeidis and Jøsang, 2018). Anomalies can

* Corresponding author.
E-mail address: jmikhail@mitre.org (J.W. Mikhail).

https://doi.org/10.1016/j.cose.2020.102002
0167-4048/

```
EventCode=1 regsvr32.exe | search ParentImage="*regsvr32.exe" AND
Image!="*regsvr32.exe*"
```

Fig. 1. Typical host-based analytic comprised of an event id and a string.

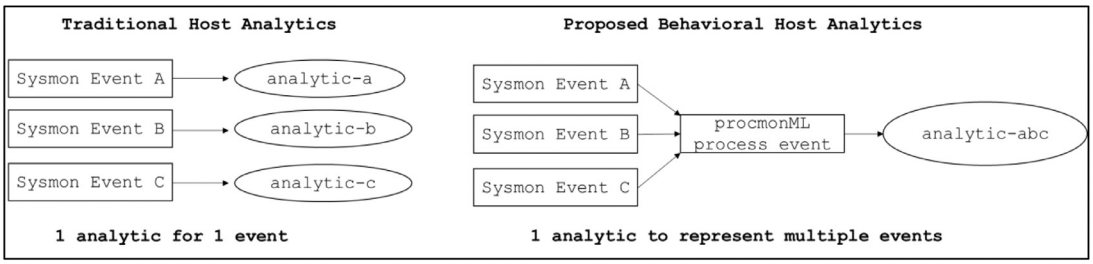

Fig. 2. Comparison of traditional analytics to procmonML analytics which aggregate multiple system events/behaviors into a single process event.

be flagged and manually investigated to determine if they are benign or part of a cyberattack. A typical host analytic is comprised of an event code and combined with one or more strings to match a field (Roth and Patzke, 2018). An example is provided in Fig. 1. In this case the analytic matches event code 1 (corresponding to a new process start) where the new process file string is `regsvr32.exe`. An adversary is easily able to rename `regsvr32.exe` to `regsvr33.exe` to avoid detection in this case. While there are methods to identify renamed binaries such as comparing the executable name to the `ORIGINALFILENAME` field of the binary, this requires extra effort on behalf of the cyber defender. In addition, there are attack mechanisms to compromise the native Windows binary signature validation process by simply modifying the registry (Graeber, 2017).

This research effort utilizes machine learning techniques to develop behavioral host-based analytics that are more resilient to adversary evasion than traditional heuristic analytics, using a process monitoring "*procmon*" based approach. We considered Sysmon and Event Tracing for Windows (ETW) as possible data sources, but ruled out ETW because the quantity of events even after filtering was a challenge to manage. Whereas traditional analytics utilize a single event for analysis, we propose organizing events by individual process and aggregating multiple generated system events into a single procmonML process event. This is depicted in Fig. 2. A lightweight sensor is installed on endpoints to collect and compress the endpoint data. Each process event is comprised of features that include event quantities and process metadata. Compressing multiple security events into a single process event reduces the quantity of events prior to SIEM ingestion, and allows for the correlation of multiple system behaviors for a given process. The resulting generated analytics are comprised of behavioral features/parameters, and the parameters of the generated analytics have a corresponding numerical threshold value rather than a string value, as seen in traditional analytics.

The scope of the offensive techniques investigated in this study includes common post-exploitation techniques used by adversaries to evade detection, obtain credentials, and remain resident on a system. These techniques were chosen based on adversary popularity in recent threat reports (CrowdStrike 2019, Red Canary 2020). We also targeted attack techniques with multiple variations in an attempt to generate a single analytic for each technique and identify commonalities between variations of techniques. ATT&CK techniques T1050 (New Service Creation) and T1053 (New Scheduled Task Creation) were chosen as test cases for maintaining persistence on a system. In these attacks, adversaries can issue commands, run custom code, and utilize the Windows GUI to create new scheduled tasks and services that run after system restarts. T1003 (Credential Dumping from local security authority subsystem service (lsass) memory) was chosen as a use case for obtaining user credentials. There are numerous tools to execute this attack that reads from the `lsass.exe` process memory location which stores user credential information (Ussath et al., 2016). The credential information can be dumped to disk or directly exfiltrated. Finally, we looked at the following evasion techniques: T1117 (Regsvr32 creating a new process) and T1055 (Process Injection). Regsvr32 is a "living off the land" technique to bypass application whitelisting and open a new process for further system exploitation. Process injection is a technique where malicious code can be directly injected into a legitimate process making detection more difficult (Block and Dewald, 2019). These attacks are run in both a lab and within MITRE's production enterprise environment in order to generate the necessary labeled data for this experiment.

This work provides several novel contributions to the HIDS domain. Furthermore, it addresses many of the challenges outlined in a recent work by Liu et al. (Liu et al., 2018) including false positive results, performance issues, dataset shortcomings, and lack of robustness for detecting advanced threats. Many recent HIDS approaches focus on larger n-gram call sequences such as in Creech & Hu (Creech and Hu, 2014) however these sequences can be computationally expensive considering that the size of the feature vector grows exponentially, as n increases (Hubballi et al., 2010). Rather than implement a sequence-based approach, we propose aggregating endpoint data by individual processes. This is similar to a work by Tobiyama & Yamaguchi (Tobiyama et al., 2016) where process behavior is logged in an attempt to discriminate between benign and malicious processes. While their approach is used to detect malware processes, our approach is applied to specific adversary TTPs. In addition, our research investigates significantly more process features (40+ versus 7) as a means to improve detection performance. Similar features (file, registry, and API) to those implemented in this study have been previously implemented in various dynamic malware analysis studies (Jethva et al., 2020, Liu et al., 2011, Cabau et al., 2016, Tajoddin and Abadi, 2019), where the objective is to discriminate between benign and malicious files. This is different than our approach where we demonstrate the detection of TTPs by analyzing endpoint processes from kernel log data. While heuristic analytics are often written to detect a single variation of a technique, we also demonstrate that analytics can be generated to detect multiple variations of a single attack technique. Rather than test the proposed approach on existing outdated datasets, we collected real data from both a lab environment and MITRE's production environment. In our work, multiple variations of advanced attacks that incorporate elements of evasion are executed based on real adversary activity documented

in recent threat reporting. Furthermore, we demonstrate that we are able to generate interpretable/explainable rule-based analytics which provides an advantage for defenders compared to typical black-box models. Results from testing showed that the generated procmonML analytics are able to achieve false positive rates (FPR)s of less than 0.5% in the (5) different evaluated attack technique categories. Additionally, the results show that the approach can be implemented with minimal performance overhead on the endpoint and a comparable number of daily alerts to existing log collection baselines. Overall, the procmonML approach offers an improved means to defend an enterprise organization from advanced threats that incorporate elements of evasion.

2. Related work

A recent SANS report (Shackleford, 2016) highlighted the importance of host/endpoint threat detection. The report presented survey data concluding that endpoint detection was the top means by which organizations were alerted to breaches. This has changed from the historical reliance on network-based data for detection. One common approach to monitor multiple endpoints is to consolidate endpoint events in a central SIEM solution for threat hunting purposes and run queries to detect specific threats (Sekharan and Kandasamy, 2017). There are several open source analytics repositories that provide public access to common heuristic analytics such as MITRE's Cyber Analytic Repository (CAR) (The MITRE Corporation 2020), Sigma Analytics Repository (7), and Endgame's Event Query Language Repository (EQL) (Endgame 2020). The majority of these analytics all rely on some form of easily evaded string matching.

A recent survey of host-based intrusion detection (HIDS) was provided by Liu et. al. (Liu et al., 2018), which provides insight to the approaches that have been applied to HIDS. Many of the recent approaches utilize methods such as n-grams, bag-of-words, natural language processing, and hidden Markov models. However, enumerating sequences can be computationally expensive, which can cause performance challenges. Other challenges identified for HIDS in the survey included managing the massive amount of endpoint data, having sufficient hardware resources, and managing the quantity of false positive (FP) alarms that are raised. Several rule-based approaches to detect anomalies were referenced in the survey, however the majority of these approaches are over a decade old (Lee and Stolfo, 1998, Lee et al., 1996, Jiang et al., 2007). Yukin et al. (Yuxin et al., 2011) applied common classifiers such as k-nearest neighbors (knn), support vector machines, and decision trees to system call graphs from benign and malicious executables. An interesting approach is provided by Matsudi et.al. (Matsuda et al., 2019), where the authors investigated dynamic-link library (DLL) loading as a means to differentiate between benign and attack host-based data. The authors noted that DLL loading varied from computer to computer and it also varies between different versions of lsass credential dumping tools such as Mimikatz. As an example with Mimikatz, `apphelp.dll` is not loaded on Windows 7. They concluded that there is value is understanding common DLL lists for different malicious tools..

In another HIDS survey by Bridges et al (Bridges et al., 2019), authors noted that the features used in applicable HIDS studies generally fell into two categories: sequence or frequency features. While numerous studies were cited for sequence-based features, there was significantly less studies that implemented frequency based features, as we did in this work. It was noted that one benefit of frequency-based features is that they are inherently more lightweight to work with. The Bridges et al (Bridges et al., 2019) survey also provided a review of several studies that relied on using system log information to detect threats. For example in Verma & Bridges (Verma and Bridges, 2018) authors propose log processing combined with a kNN classifier to detect ransomware attacks. The majority of the cited works were applied to older datasets, and did not address TTP detection.

Deep learning approaches have also been applied to host-based intrusion detection. Kim et.al. (Kim et al., 2016) applied an ensemble of long short-term memory (LSTM) models to the ADFA-LD and LANL datasets using natural language processing techniques on collections of system calls. Similarly, convolutional neural nets and recurrent neural nets have also been applied (Chawla et al., 2018). A recent work (Hendler et al., 2018) by Hendler et.al. also experimented with applying deep neural networks to the growing PowerShell attack domain by using character embeddings to identify malicious PowerShell scripts.

While numerous network intrusion detection datasets were released over the last 20 years such as KDD99 (Pfahringer, 2000), UNSW-NB15 (Moustafa and Slay, 2015), and AWID (Mikhail et al., 2019), there is the lack of labeled datasets for host intrusion detection (Turcotte et al., 2017). The Liu et al (Liu et al., Nov. 2018) study noted that traditional HIDS approaches struggle to demonstrate robustness against advanced threats. A review of published works provides insight to the limited openly available research in the host-based detection domain. Los Alamos National Laboratory (LANL) released a limited dataset that primarily correlates host data to network data, however, the majority of the labels are for systems behaviors rather than modern attacks. Siadati et. al. (Siadati et al., 2016) used the LANL dataset to detect malicious logins using the network data. Similar to our proposed approach, rules were generated to detect these logons. The final system was able to achieve a low false positive rate of 0.005%, however, it was only able to detect about half of the malicious logins. The ADFA-WD and ADFA-LD datasets (Borisaniya and Patel, 2015) for both Windows and Linux respectively have been also used for some host-based detection purposes (Sudqi Khater et al., 2019, Mouttaqi et al., 2017), but these datasets lack labeled attack TTP data. As noted previously, many of the HIDS datasets are not representative of current diversified attack methods (Liu et al., 2018).

3. Approach

The overall framework of this experiment consists of data collection, offensive technique execution, and model/analytic validation. An overview of the validation process is shown in Fig. 3. Both benign and attack data was collected in the lab using the procmonML endpoint sensor. All data was ingested in a local Splunk instance and then provided to a random forest tree ensemble where the top performing rules from individual decision trees were translated into a query to detect a specific attack technique for use in a SIEM platform.

One of the primary objectives of the research effort was to provide human-interpretable model feedback. A challenge with many machine learning models is that they often act like a "black box" and provide limited insight about how their output is generated (Ribeiro et al., 2016, August). A random forest model was chosen for its ability to be explainable, since it is comprised of individual decision trees. The individual decision trees are comprised of nodes and leaves where combinations of detection rules can be extracted from each of the individual paths. Since the decision tree splitting algorithm identifies the features which result in the best splits, the algorithm performs its own feature selection, and therefore can be applied to identify the most important features for each type of attack technique. Rather than using a single decision tree to extract rules, a random forest offers the benefit of generating multiple trees based on random subsets of the entire feature set. This provides a larger, diverse set of rules from which a single rule selection can be made.

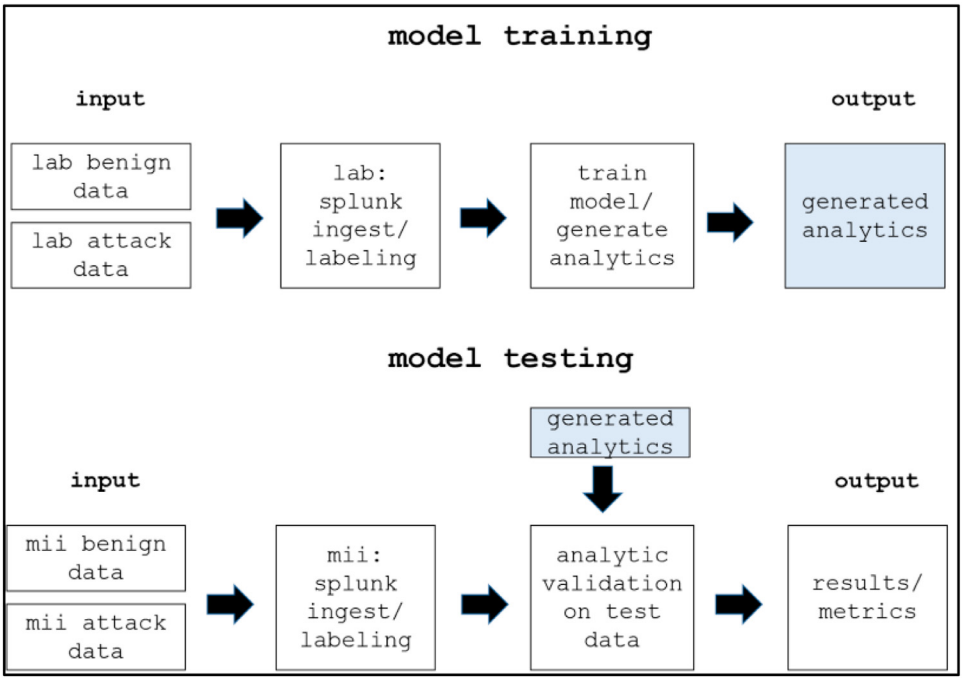

Fig. 3. Overview of the analytic validation process.

Table 1
Ensemble training model parameters.

Parameter	Value	Parameter	Value
Estimator	decision tree	min_samples_split	2
n_estimators	20	min_samples_leaf	1
min_precision	0.50	min_weight_fraction_leaf	0.0
max_samples	0.90	max_leaf_nodes	None
max_features	0.50	min_impurity_decrease	0.0
class_weight	balanced	min_impurity_split	None
criterion	gini	bootstrap	True
max_depth	10	oob_score	False

In the test environment, the procmonML sensors were deployed to MITRE's production network (MII) and data was collected from six endpoints over a 1.5 week period. In addition to the normal benign traffic that was collected, we executed variations of the five chosen attack techniques and apply the generated analytics from the lab environment. The generated analytics were evaluated on their ability to detect variations of attack techniques and the corresponding FPR associated with the final analytic.

3.1. Analytic generation

Analytics were generated by decomposing a random forest ensemble comprised of individual decision tree learners into equivalent text-based analytics using a scikit-learn (Pedregosa et al., 2020) random forest model. Table 1 provides a breakdown of the model parameters, and the parameter defintitions are listed in Appendix D. The model was implemented with the default scikit-learn hyperparameters (Pedregosa et al., 2020). Splits are determined using the standard Gini value metric. In order to provide diversity between the individual learners, a percentage of the total number of training samples and features were included in each estimator. In order to account for the large class differences between the benign and attack samples, individual samples were weighted in a balanced manner, inversely proportional to class frequencies in the input data. In order to simplify the decision boundary for this initial experiment, all analytics were generated using a binary classification objective. Separate models were trained for each of the (5) attack techniques. More formally, this technique takes an input vector $x = (x_1, x_2, \ldots, x_f)$ with f number of features for a single input sample. Given two classes, y_{benign} and y_{attack}, with n_{benign} and n_{attack} samples in each individual class, where $\{y_{benign}\} >> \{y_{attack}\}$, the weight w of a individual sample $x_{i-benign}$, $w_{x_{i-benign}} = 1/n_{benign}$ and $w_{x_{i-attack}} = 1/n_{attack}$. This balanced weighting schema mitigates the negative effects of an imbalanced class distribution.

The resulting trees within the ensemble model can then be traversed into order to determine the paths that meet the minimum precision criteria. Each path is evaluated on only a subset of the training data. Fig. 4 provides an example, showing a traversal path (blue) to an attack labeled sample from which an analytic can be derived. Each node represents a splitting criteria with the corresponding Gini value. Traversal continues to the left if the condition is true, given a sample feature, $x[i]$. For a given feature $x[i]$, the actual feature name is retrieved from the *feature_name* array, which stores the feature names. In general, higher scoring rules are generated for paths that completely separate samples from different classes. For a given traversal path where a node condition is false, the operator (<, >, <=, >=) of the false condition must be flipped.

In order to simplify the generated analytic, the highest scoring single path from all the trees is extracted. The score of the path, is based on calculating the precision of the path (Eq. 1) with the training data. Precision was chosen since it takes into account both the true positives (TP) and FPs of a given path. A TP reflects an actual attack sample, whereas a FP is defined as a benign sample that is incorrectly classified as an attack sample. While single paths were found to be sufficient for the use cases that we tested in this work, multiple high scoring paths can also be combined with an `OR` statement for future applications.

$$P = TP / (TP + FP) \quad (1)$$

3.2. Data collection environments

Fig. 5 depicts the separate training and test environments that were used in this experiment. These separate environments are implemented in order to evaluate the generalization ability of the

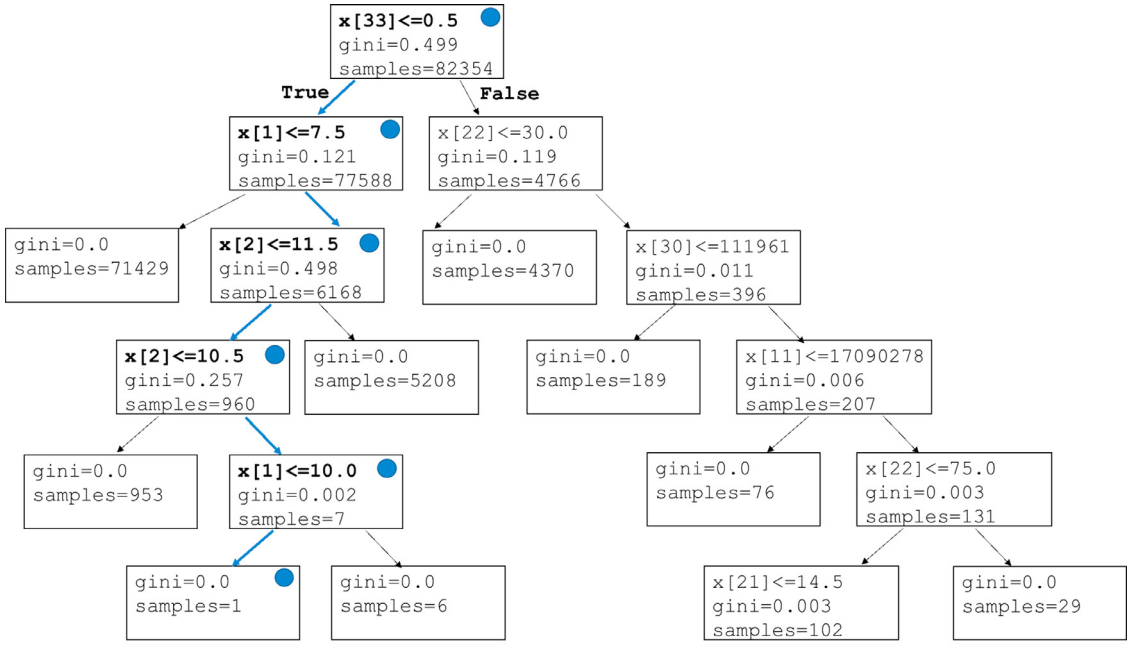

Fig. 4. Converting a decision tree path into an equivalent text-based analytic.

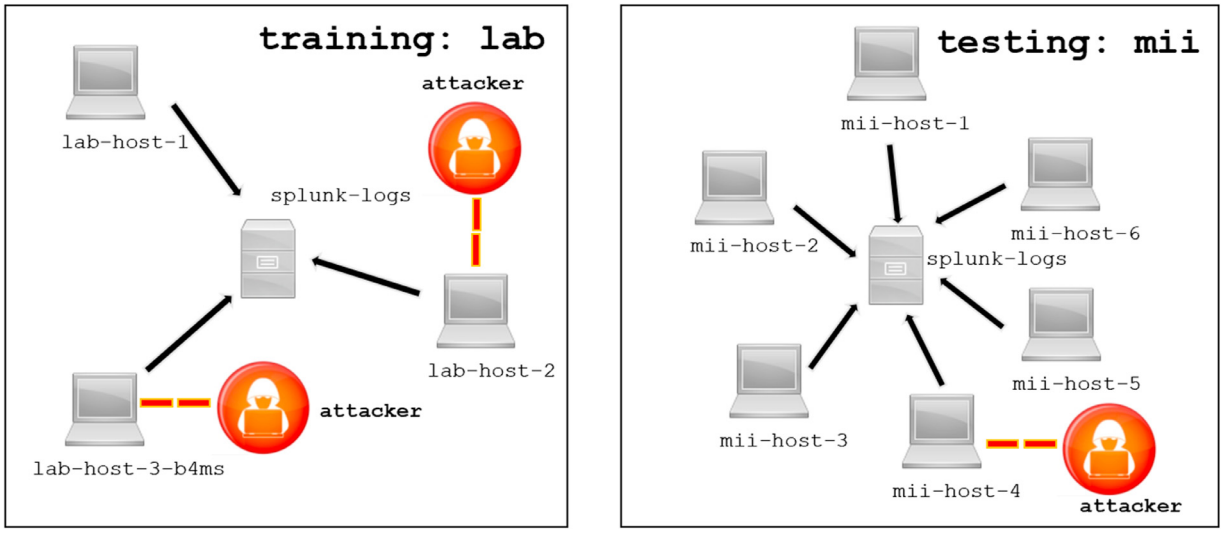

Fig. 5. Separate Lab and MITRE production data collection environments.

lab-generated analytics. The training lab consists of an azure B4MS virtual machine (VM), and two desktop computers running standard Windows 10 images. The virtual host is a newly created azure instance. A complete system reset was performed on the other two hosts in the training environment. After the procmonML sensor is deployed to each endpoint, a user is assigned to the training hosts and this serves as the benign training data collection. The hosts were then subject to an attacker taking control of the machine and then executing multiple attack variations of the five different techniques. Both benign background data and attack data is collected, labeled, and ingested into the training data index of a local version of Splunk. The ensemble model is trained and the corresponding analytics are generated with the lab data. The testing environment includes a subset of the machines on MITRE's production enterprise network (MII). All MII hosts are Windows 10 machines. The same procmonML sensor is deployed to six different machines over a 1.5 week period. Users were instructed to perform their typical work actions over the data collection period. Typical work actions for these users included activities such as document/spreadsheet editing, email communication, web browsing, and teleconferencing. Automated backup tools and a Splunk forwarder was also present on the machines. One of the hosts was also designated as an attack machine where offensive techniques were executed and the corresponding procmonML process logs were collected and labeled.

3.3. procmonML endpoint sensor

As noted previously, an endpoint sensor is responsible for collecting and aggregating multiple Sysmon events into a single process event. The sensor resides on the endpoint throughout the data collection phase and includes the Sysmon executables (Russinovich, 2009) and a Sysmon configuration file that defines which Sysmon events should be generated. The sensor was developed using a PowerShell script to collect events from the Sys-

Table 2
Summary of the collected/filtered sysmon events for the procmonML Sensor.

Event ID	Configuration Description
Event ID 1: Process creation	Log all new processes.
Event ID 3: Network connection	Log all transmission control protocol (TCP)/user datagram protocol (UDP) connections.
Event ID 5: Process terminated	Log all process terminations.
Event ID 7: Image loaded	Log all modules loaded.
Event ID 8: CreateRemoteThread	Log all instances where a process attempts to create a thread in another process.
Event ID 9: RawAccessRead	Log all instances where a process conducts reading operations from the drive using the \\.\ denotation.
Event ID 10: ProcessAccess	Log all instances where a process attempts to access another process, excluding access codes 0×1000 (`PROCESS_QUERY_LIMITED_INFORMATION`), 0×1400 (`PROCESS_QUERY_LIMITED_INFORMATION | PROCESS_QUERY_INFORMATION`).
Event ID 11: FileCreate	Log all instances where a file is created or overwritten.
Event ID 12: RegistryEvent	Log specific instances of registry key/value create/delete operations. See Appendix B.
Event ID 13: RegistryValueSet	Log specific instances of registry value setting. See Appendix B.
Event ID 14: RegistryEvent	Log specific instances of registry key and value renaming. See Appendix B.
Event ID 15: FileCreateStreamHash	Log all instances of a named file stream creation.
Event ID 17: PipeEvent	Log all instances of a named pipe creation.
Event ID 18: PipeEvent	Log all instances of a named pipe connection.
Event ID 22: DNSEvent	Log instances of domain name system (DNS) queries.

mon data feed and process them into the procmonML process format. These events are organized by the process identification number. Each event has a corresponding label to designate the event as either a benign background event or a malicious attack event. Table 2 provides an overview of the types of events that are collected/filtered by the endpoint sensor. Each event corresponds with a unique system behavior.

3.4. Model features

While the complete list of sensor features is provided in Appendix A along with a general description, we provide a brief discussion here on the feature set. As noted previously, each Sysmon event that is generated on the endpoint is grouped into a corresponding process event and each process event has a corresponding set of features. At the most basic level, features include the quantity of each type of event that is generated for each process, in addition to process time, child process counts, and integrity level. Events 3, 10, and 12-14 have additional features that are calculated in near real-time by the procmonML sensor.

For Event-3 (network connection) the average and maximum source/ destination ports is also calculated. Event-10 provides additional information on process behavior including what target process a particular source process interacts with. This specific event contains a lot of important information; however, it also generates many alerts. Many organizations cannot afford to turn this event on without considerable filtering. Since the proposed approach compresses all these events for a given process into a single process event, we are able to encode the information in a more efficient manner. Information extracted from Event-10 includes the average access code, the largest access code, attempts to access the lsass process, and call trace information. While the access code field is a discrete value field formed from combinations of 14 different access rights, we ultimately decided to extract average and maximum values for simplicity purposes. The results section provides additional discussion on how this field contributed to the detection of certain attacks.

Events 12-14 alert on various registry actions. The sensor creates a time-series distribution of the registry key depth, which refers to the depth of the registry key location. For instance given the registry key `HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run`, this will have a registry key depth of six. We initially hypothesized that this approach would encode timeseries information about the registry behavior of a process without the need to translate a registry string into an associated feature set. The formal definitions and pseudocode for the registry key depth features are provided in Appendix A. In general, the time-series registry key depth features include metrics such as distribution size, mean, maximum, and number of points that are above/below the mean.

Most system processes begin and end over a period of several seconds to several minutes. However, there are numerous processes that remain active over the entire period an endpoint is running. Time slicing is needed since anomalous events are more difficult to detect in larger quantities of benign data. In other words, long running processes are broken into multiple separate process events using a time window of 10 minutes. While this parameter is tunable, we found that a 10 minute time window balances the number of generated events with a sufficiently small window in order to ensure that anomalies are identifiable. The process consolidation algorithm is provided in Algorithm 1.

Algorithm 1 Process consolidation.

Input: Sysmon event, s_{eid} for event# *eid*
process event array, P
time start/current/window, $t_{start}, t_{current}, t_{window}$
Output: process event, $P(x)$
extract process id, $s_{eid}.pid$
if ($eid == 1$) //process start event
  $P += s_i$
else
  for index x in P **do**
  if $P(x).pid == s_{eid}.pid$
    update process event, $P(x) \leftarrow s_{eid}$
    if $eventid == 5$ //process end event
      write $P(x) \rightarrow output\_file$
      delete $P(x)$
if $(t_{current} - t_{start}) > t_{window}$
  for index x in P **do**
    write $P(x) \rightarrow output\_file$
  $P(x) \leftarrow \{0, 0, ..0\}$
  $t_{start} \leftarrow t_{current}$

Table 3
Training dataset.

Technique ID	Attack Description	Sample Count	Percent
-	Benign Background Data	91,488	99.8875%
T1003	Lsass.exe memory credential dumping/minidump (PowerSploit 2020)	3	0.0033%
	Lsass.exe memory credential dumping/procdump (Russinovich, 2009)	8	0.0087%
	Lsass.exe memory credential dumping/taskmanager	5	0.0055%
	Lsass.exe memory credential dumping/Mimikatz (Delpy, 2014)	2	0.0022%
T1050	New Service Creation - sc.exe or PowerShell	8	0.0087%
T1053	New Scheduled Task Creation - schtasks.exe, PowerShell, or GUI	8	0.0087%
T1055	Process Injection - CreateRemoteThread	19	0.0207%
	Process Injection - Hooking	10	0.0109%
	Process Injection - Asynchronous Procedure Calls (APC)	5	0.0055%
	Process Injection - SetThreadContext	6	0.0066%
	Process Injection - Reflective DLL (RDLL)	3	0.0033%
	Process Injection – Hollowing	12	0.0131%
T1117	Regsvr32 squiblydoo - start a new process	9	0.0098%
	Regsvr32 DLL load - start a new process	4	0.0044%

Table 4
Test dataset.

Technique ID	Attack Description	Sample Count	Percent
-	Benign Background Data	752,199	99.9790%
T1003	Lsass.exe memory credential dumping/minidump	2	0.0003%
	Lsass.exe memory credential dumping/procdump	18	0.0024%
	Lsass.exe memory credential dumping/taskmanager	3	0.0004%
	Lsass.exe memory credential dumping/Mimikatz	2	0.0003%
	Lsass.exe memory credential dumping/mimidogz*	2	0.0003%
	Lsass.exe memory credential dumping/SharpDump (Modified)* (SharpDump 2018)	2	0.0003%
	Lsass.exe memory credential dumping/dumpert*	2	0.0003%
	Lsass.exe memory credential dumping/ppldump*	3	0.0004%
T1050	New Service Creation - sc.exe or New-Service	3	0.0004%
	New Service Creation - scfake.exe*	2	0.0003%
	New Service Creation - WinSW* (WinSW Windows Service Wrapper Mar. 20, 2020)	2	0.0003%
	New Service Creation - SharPersist* (Fireeye Mar. 20, 2020)	2	0.0003%
T1053	New Scheduled Task Creation - schtasks.exe, New-ScheduledTask	2	0.0003%
	New Scheduled Task Creation - schfake.exe*	3	0.0004%
	New Scheduled Task Creation - GUI*	3	0.0004%
	New Scheduled Task Creation - Macro*	3	0.0004%
	New Scheduled Task Creation - SharPersist*	2	0.0003%
T1055	Process Injection - CreateRemoteThread	24	0.0032%
	Process Injection - Hooking	6	0.0008%
	Process Injection – Asynchronous Procedure Calls (APC)	7	0.0009%
	Process Injection - SetThreadContext	3	0.0004%
	Process Injection - Process Hollowing	12	0.0016%
	Process Injection – Reflective DLL (RDLL)	2	0.0003%
	Process Injection – AutoCorrectProc*	4	0.0005%
	Process Injection – CtrlInject* (Klein and Kotler, 2019)	6	0.0008%
	Process Injection - PROPagate* (Klein and Kotler, 2019)	2	0.0003%
T1117	Regsvr32 squiblydoo attack starting a new process	23	0.0031%
	Regsvr32 malicious DLL starting a new process	4	0.0005%
	Regsvr33 squiblydoo loading a remote sct file*	2	0.0003%
	Regsvr33 malicious DLL starting a new process*	2	0.0003%

* Not in Training Data.

3.5. Attack techniques

Five different attack techniques were used for evaluation in this work. These techniques were chosen based on adversary popularity in recent threat reports and potential for evasion. For each technique we also demonstrate the limitation of masquerading techniques (file renaming) on traditional heuristic analytics. Tables 3 and 4 provide the breakouts of the training and test datasets. The test dataset contains additional variations of attacks that are not present in the training data.

- T1003 (Credential Dumping from lsass memory): Attackers can steal user credentials stored in lsass memory.
- T1050 (New Service Creation): This technique is used by attackers to remain resident on a compromised system by creating a new service.
- T1053 (New Scheduled Task Creation): This is another technique that attackers utilize to remain resident on a system by creating a new scheduled task
- T1055 (Process Injection): Attackers use process injection to insert malicious code into a benign process in order to evade detection. Process hollowing techniques are included here, where a new process is created in a suspended state, malicious code is injected, and the process is resumed.
- T1117 (Regsvr32): Regsvr32 is a “living off the land” evasion technique used to elevate privileges or execute payloads via a squiblydoo attack (using `scrobj.dll`) or loading/unloading a malicious DLL.

4. Results

This section provides a discussion on the validation results of the procmonML analytics (PA) including a comparison to tradi-

```
TA-T1003-1: EventID=10 AND TargetImage="c:\windows\system32\lsass.exe"
(CallTrace="*dbghelp.dll*" OR CallTrace="dbgcore.dll")

TA-T1003-2: EventID=10 TargetImage="C:\\WINDOWS\\system32\\lsass.exe"
(GrantedAccess=0x1410 OR GrantedAccess=0x1010 OR GrantedAccess=0x1438 OR
GrantedAccess=0x143a OR GrantedAccess=0x1418)
CallTrace="C:\\windows\\SYSTEM32\\ntdll.dll+*|C:\\windows\\System32\\KERNELBA
SE.dll+20edd|UNKNOWN(*)"
```

Fig. 6. Traditional analytics for detecting Lsass credential dumping.

```
Event10_LsassAccess >= 1 AND Event10_MaxCallTrace <= 25 AND
Event12_RegistryCreateOrDelete <= 5 AND pIntegrityLevel <= 75
```

Fig. 7. procmonML Analytic PA-T1003-1 to Detect Lsass Credential Dumping.

Table 5
Comparison of typical analytics vs. procmonML analytics in detecting different variations of lsass credential dumping.

Analytic	FPR	Taskmanager	Minidump	Procdump	Mimikatz	Mimidogz*	SharpDump (Modified)*	Dumpert*	ppldump*
PA-T1003-1	**0.053%**	**Detect**	**Detect**	**Detect**	**Detect**	**Detect**	**Detect**	**Detect**	**Detect**
TA-T1003-1	-	Detect	Detect	Detect	-	-	-	Detect	-
TA-T1003-2	-	-	-	-	Detect	Detect	-	-	-

* Not present in training data.

tional analytics (TA) in detecting multiple variations of the five attack techniques. The FPR is only provided for the procmonML analytics since the traditional analytics are incompatible with the process data format that procmonML collected for the training and test data sets.

4.1. T1003 credential dumping from Isass memory

The attacks in this technique category utilize either using a Mimikatz variation, or a variation that uses the `MiniDumpWriteDump` API function. The `MiniDumpWriteDump` function requires a reference to `dbghelp.dll` or `dbgcore.dll`. Common lsass credential dumping tools include Taskmanager, a minidump PowerShell script (PowerSploit 2020), Mimikatz (Delpy, 2014), and the procdump.exe sysinternals tool (Russinovich, 2009). Three newer stealthier variations were also utilized in this work: SharpDump, Outflank-Dumpert (de Plaa, 2020) and ppldump (Hammou, 2018). In the modified SharpDump variation, which uses the `MiniDumpWriteDump` function, `dbgcore.dll` is renamed to `dbgcore1.dll` in the source code prior to compilation for evasion purposes. The ppldump attack is a newer approach to credential dumping that uses driver registration and process injection to dump lsass memory internally from the lsass process itself. In this variation, shellcode to run `MiniDumpWriteDump` is injected into the protected lsass process. Data labeling in the training and test datasets for this technique requires identifying the process event that performs the lsass credential reading.

Two common traditional analytics for detecting lsass credential dumping are provided in Fig. 6. TA-T1003-1 is useful for detecting instances of lsass credential dumping that utilize the `MiniDumpWriteDump` function, which are utilized by the majority of the lsass credential dumping techniques tested in this work. This analytic does moderately well in detecting several of the `MiniDumpWriteDump` variations, except for the modified SharpDump and the ppldump attack variations. Since `dbgcore.dll` was renamed in the SharpDump attack, it did not show up in the call trace field and therefore was undetectable. The ppldump attack failed to generate an Event-10 alert associated with credential dumping. We hypothesize that since the `lsass.exe` process is accessing `lsass.exe` memory, Event-10 is not generated, since this event is only generated when one process accesses a different process. Given that this particular attack also leverages new service creation and process injection, this attack is also reviewed in subsequent sections in order to investigate the additional opportunities for detection. TA-T1003-2 is able to detect both instances of Mimikatz, including the renamed binary (Mimidogz).

The resulting procmonML analytic, PA-T1003-1 is provided in Fig. 7. This analytic identifies processes that access the lsass process using the `Event10_LsassAccess` parameter. Further parameter filtering includes processes where the maximum call trace is less than 25 files (`Event10_MaxCallTrace`), where the number of Event-12 events is less than or equal to 5, and where the integrity level of the process is not `SYSTEM` (<=75). Overall this analytic detected all tested variations of lsass credential dumping with a FPR of 0.0529%. This included four new variations of attacks in the test dataset. A review of the returned false positive results show that 46 percent of the returned results are the `splunkd` process. While there was no Splunk processes in the training data, introducing Splunk into the training environment could possibly improve future results. Other significant returned false positive processes include the `svchost.exe` and the WMI Provider Host (`wmiprvse.exe`) processes. Table 5 provides a comparison of the traditional analytics to the procmonML analytic. Compared to the traditional analytics, the procmonML analytic is shown to be resilient to the instances of SharpDump and ppldump. In addition, we generated a single analytic that works for detecting both the Mimikatz and the `MiniDumpWriteDump` variations of this technique. Future work with this particular technique includes further investigation to determine the susceptibility of having a renamed `lsass.exe` process successfully loaded by the endpoint. Since `lsass.exe` is a protected process loaded by the kernel, a simple rename to `lsass1.exe` would not cause the masqueraded file to be loaded, without additional modification to system binaries or kernel memory.

4.2. T1050 new service creation

In addition to the common `sc.exe` and `New-Service` commands to execute a new service creation, four additional variations of this technique were added to the test data set to demonstrate the generalization ability of the procmonML analytics. The `scfake.exe` variation is a renamed version of the `sc.exe` binary. WinSW (WinSW Windows Service Wrapper, 2020) and SharPersist (Fireeye, 2020) are two open source libraries that use .NET/C# libraries to create new services for persistence purposes.

Table 6
Comparison of Typical Analytics vs. procmonML Analytics in Detecting Different Variations of New Service Creation.

	FPR	sc.exe	New-Service	scfake.exe*	WinSW .NET*	SharPersist-NewService*	ppldump*
PA-T1050-1	**0.0111%**	**Detect**	**Detect**	**Detect**	**Detect**	**Detect**	**Detect**
TA-T1050-1	-	Detect	Detect	-	-	-	-

* Not in Training Data.

```
TA-T1050-1: CommandLine="*sc*create*" OR CommandLine="*New-Service*"
```

Fig. 8. Traditional analytic for detecting new service creation.

```
RegistryDepthCAbove_ts >= 1.5 and RegistryDepthMean_ts < 5.13
```

Fig. 9. Proposed procmonML Analytic PA-T1050-1 for detecting new service creation.

In both cases, binaries are run to execute the new service creation. The ppldump variation described in the previous attack technique is also included here, since it creates a new service to load a driver.

Interestingly, the procmonML sensor data showed that neither the `sc.exe` process nor the PowerShell process issuing the `New-Service` command was performing any kind of registry access behavior that we expected to see with the new service creation event. Further process monitoring analysis showed that the `services.exe` process acts as an intermediary process between front end commands/tools and the actual registry. Regardless of whether a new service command was issued via `sc.exe` or PowerShell, the `services.exe` process was performing the following actions:

1. Creating a registry object in: `\System\CurrentControlSet\Services\NewServiceName`
2. Setting several values at this registry key including `(Type, Start, ErrorControl, ImagePath, ObjectName.`

Since the `services.exe` process starts at system bootup and remains active, time slicing is activated for this process, with a time window of 10 minutes. Training and test data for this technique were generated by executing the attacks and labeling the corresponding specific time slices of the `services.exe` process.

A consolidated version of the traditional analytic is provided in Fig. 8. While the traditional analytic TA-1050-1 is able to detect standard executions of `sc.exe` and `New-Service`, an unsophisticated adversary can easily utilize masquerading techniques to evade detection, such as renaming `sc.exe` to `scfake.exe`. This heuristic analytic also fails to detect any kind of custom tool brought by an adversary to add a new service via the Windows API.

The procmonML analytic for detecting new service creation (PA-T1050-1) is provided in Fig. 9. The `RegistryDepthCAbove_ts` parameter identifies processes where the average number of registry access to a key depth greater than the mean is greater than 1.5. The `RegistryDepthMean_ts` parameter matches processes where the average registry key depth is less than 5.13. The analytic is able to detect all six variations of the technique that created a new service. This includes the detection of the ppldump attack described in the previous section, which calls the `CreateServiceA` function to create a new service to load a driver. The proposed analytic achieved an overall FPR of 0.0111%. Further analysis showed that the production test data contains 2,789 instances of `services.exe`, of which there are only eight false positive instances returned by the procmonML analytic, providing substantial evidence that the analytic is not only identifying instances of `services.exe`, but also able to differentiate behavior associated with `services.exe` creating a new service from its other registry access behavior. Significant false positive results include the `lsass.exe` process and the Windows Provider Host (`wmiprvse.exe`) process. The `lsass.exe` process was responsible for 50% of the returned results. Table 6 provides a comparison of the analytics tested for this technique.

4.3. T1053 new scheduled task creation

New scheduled task creations can be executed via `schtasks.exe`, the `New-ScheduledTask` PowerShell command, the scheduled task GUI, and a custom binaries that utilize the Windows API. Four new attacks are included in the test data, each leveraging a different variation of creating a new scheduled task: a renamed `schtasks.exe(schfake.exe)` file, manual GUI creation, a .NET/C# solution (SharPersist), and a VB macro.

Similar to the previous technique, process monitoring data showed no registry activity for any of the front end commands that are typically used for creating a new scheduled task (`schtasks.exe/New-Scheduled Task`). Registry behavior for this technique was traced to the `svchost.exe -k netsvcs -p -s Schedule` process. Similar to the previous technique, time slicing is activated for this process, with a time window of 10 minutes. Training and test data for this attack technique were generated by executing the attacks and labeling the corresponding specific time slices of the `svchost.exe -k netsvcs -p -s Schedule` process when attack instances of this technique were executed. Further analysis of the registry activity showed that for new scheduled task creations, this particular svchost process performs the following actions:

1. Creates a registry object `NewTaskName` at `*\Software\Microsoft\Windows NT\CurrentVersion\Schedule\TaskCache\Tree\NewTaskName`
2. Sets registry key values `(SD, ID, Index)`.
3. Another registry object is created at `*\Software\Microsoft\Windows NT\CurrentVersion\Schedule\TaskCache\Tasks\ID` where `ID` is the unique identifier assigned to the new scheduled task and assigns a path and hash key.
4. A file is also created in the target folder `C:\windows\system32\tasks.`

The traditional analytics for detection of this technique primarily rely on string matching (Fig. 10). TA-1053-1 aims to identify common commands for creating a new scheduled task. A simple binary rename is sufficient to avoid detection. Furthermore, any binary leveraging the Windows API to create a new scheduled task would not produce an alert, such as with the SharPersist variation. This analytic also fails to detect manually created new scheduled task via the TaskScheduler GUI. TA-T1053-2 aims to detect a single variation (VB macros within common MS Office programs) of creating a new scheduled task.

```
TA-T1053-1: CommandLine="*schtasks*create*" OR CommandLine="*New-ScheduledTask*"
TA-T1053-2: (ProcessName="excel.exe" OR ProcessName="winword.exe" OR
ProcessName="powerpnt.exe" OR ProcessName="outlook.exe") AND EventID=7 AND
ImageName= "taskschd.dll"
```

Fig. 10. Traditional analytics for detecting new scheduled task creation.

```
Event11_FileCreate >= 1 AND RegistryDepthLongestAbove_ts > 2.5 AND
RegistryDepthMean_ts > 8.6 AND RegistryDepthStddev_ts <= 0.92
```

Fig. 11. Proposed procmonML Analytic PA-T1053-1 for detecting new scheduled task creation.

Table 7
Comparison of typical analytics vs. procmonML analytics in detecting different variations of new scheduled task creation.

Analytic	FPR	Schtasks.exe	New-ScheduledTask	Schfake.exe*	TaskScheduler GUI*	SharPersist-NewTask*	VB Macro*
PA-T1053-1	**0.1903%**	**Detect**	**Detect**	**Detect**	**Detect**	**Detect**	**Detect**
TA-T1053-1	-	Detect	Detect	-	-	-	-
TA-T1053-2	-	-	-	-	-	-	Detect

* Not in training data.

```
TA-T1055-1: EventID=8 AND (Start_Function= "LoadLibraryA" OR Start_Function=
"LoadLibraryW")
```

Fig. 12. Traditional Analytic using RemoteCreateThread (Event-8) to Alert to a Process Injection Incident.

```
Event10_AverageAccessCodeDest > 1048576 AND
Event10_AverageAccessCodeSource > 4121 AND
Event10_AverageCallTrace <=15 AND Event10_ProcessAccessDest > 2.5
AND Event10_ProcessAccessSource <= 3 AND pIntegrityLevel <= 75
```

Fig. 13. Proposed procmonML Analytic PA-T1055-1 to Detect Process Injection.

The proposed procmonML analytic, as provided in Fig. 11, uses Event-11 (`Event11_FileCreate`) in addition to three of the registry depth timeseries features to alert on new scheduled task creations. This includes filtering processes where the average number of consecutive registry key depth accesses greater than the mean is greater than 2.5 (`RegistryDepthLongestAbove_ts`), where the registry key depth mean is greater than 8.6 (`RegistryDepthMean_ts`), and where the standard deviation of the average registry key depths (`RegistryDepthStddev_ts`)is less than 0.92. The standard deviation of the key depths encodes information about whether the registry key depth values are clustered close to the mean or spread further apart. The proposed analytic PA-T1053-1 was able to detect all variations, with an overall FPR of 0.1903%. Further analysis showed that the test data contains 2,215 instances of the `svchost.exe -k netsvcs -p -s Schedule` process, of which there were 23 false positive instances returned by the proposed analytic, providing substantial evidence that the analytic is not only identifying instances of `svchost.exe -k netsvcs -p -s Schedule`, but also able to differentiate the behavior associated with `svchost.exe -k netsvcs -p -s Schedule` creating a new service from its other registry access behavior. The `SearchProtocolHost.exe` process was found to be responsible for over 95% of the returned false positive results. Table 7 provides a comparison of the procmonML analytic to the traditional analytics.

4.4. T1055 process injection

There are numerous methods to execute process injection. Common techniques include calling the `CreateRemoteThread` function, hooking, asynchronous procedure calls (APCs), thread hijacking, and reflective DLL injection. Ten different variations of process injection attacks were evaluated including four newer attacks that were not present in the training data. AutoCorrectProc uses the `EM_CALLAUTOCORRECTPROC` message for process injection. CTRLInject abuses the `CtrlRoutine` function used by command line applications. PROPagate uses the `SetWindowSubclass` API. As noted previously, we also included the ppldump attack, which uses an APC-variant of process injection to inject shellcode into lsass.

The only open-source analytic that we could find to detect process injection is provided in Fig. 12. It is able to detect instances of `CreateRemoteThread` API function call (Event-8), however there are numerous other API functions that can be utilized by attackers to execute process injection.

The proposed analytic PA-T1055-1, shown in Fig. 13, for detecting process injection primarily leverages the behavior of Event-10 to detect process injection. The parameters include the average access codes, the average call trace size, and the number of times where the injecting process is either a source process (`Event10_ProcessAccessSource`) or a destination process (`Event10_ProcessAccessDest`). The average call trace size (`Event10_AverageCallTrace`) reflects the average number of files shown in the call trace field for Event-10. The `pIntegrityLevel<=75` parameter restricts returned results to processes that are not run as `SYSTEM`. A review of the generated Event-10 alerts shows several processes accessing the injecting process including `cmd.exe`, `msmpeng.exe`, `csrss.exe`, and `conhost.exe`. There is a significant number of events where the process performing the malicious injection is actually the destination process —in one case being accessed over 110 times. The number of instances where that the injecting process is the source process is consistently less than ten instances. This ratio would perhaps be a good future feature to incorporate. The

Table 8
Comparison of Typical Analytics vs. procmonML Analytics for Detecting Process Injection (1/2).

Analytic	FPR	CreateRemoteThread	Hooking	Hollowing	APC	SetThreadContext	RDLL
PA-T1055-1	**0.291%**	**Detect**	**Detect**	**Detect**	**Detect**	**Detect**	**Detect**
TA-T1055-1	-	Detect	-	-	-	-	-

*Not in Training Data.

Table 9
Comparison of typical analytics vs. procmonML analytics for detecting process injection (2/2).

Analytic	FPR	AutoCorrectProc*	CTRLInject*	PROPagate*	ppldump*
PA-T1055-1	**0.291%**	**Detect**	**Detect**	**Detect**	**Detect**
TA-T1055-1	-	-	-	-	-

* Not in Training Data.

```
TA-T1117-1: EventCode=1 regsvr32.exe | search
ParentImage="*regsvr32.exe" AND Image!="*regsvr32.exe*"
TA-T1117-2: EventCode=1 regsvr32.exe scrobj.dll | search
Image="*regsvr32.exe"
```

Fig. 14. Traditional analytics to detect regsvr32 creating a child process.

```
Event10_AverageAccessCodeSource > 2097152 AND Event10_AverageSystem32CallTrace
> 16.5 AND Event10_ProcessAccessDest <=13.5 AND Event18_PipeConnect <= 0.5 AND
pEventCount > 76.5 AND pTotalTime <=60
```

Fig. 15. Proposed procmonML analytic PA-T1117-1 to Detect regsvr32 creating a child process.

Table 10
Comparison of analytics for detecting T1117 regsvr32.

Model	FPR	Regsvr32 - Load DLL to load new process	Regsvr32 -Squiblydoo	Regsvr33 – Squiblydoo Remote File*	Regsvr33 – Load DLL to load new process*
PA-T1117-1	**0.0077%**	**Detect**	**Detect**	**Detect**	**Detect**
TA-T1117-1	-	Detect	Detect	-	-
TA-T1117-2	-	-	Detect	-	-

* Not in Training Data.

average access code thresholds for the destination and source processes represent values of 0 × 100000 and 0 × 1019 respectively. A review of the events generated for process injection attacks showed that for Event-10 events generated where the injection process was listed as the destination process, that access codes appeared to cluster at either the 0 × 1FFFFFF value or around the 0 × 1400 value. We hypothesize that the `AverageAccessCodeDest>1048576(0 × 100000)` parameter captures process instances where the majority of the Event-10 destination access codes reflect the 0 × 1FFFFFF value. We also noted that the procmonML analytic is also able provide a successful alert for the process injection behavior of the ppldump attack. A review of the false positive results that were returned showed the `conhost.exe` process responsible for nearly 50% of the returned results. Tables 8 and 9 provide a summary of the proposed analytic to the traditional analytic in detecting process injection attacks.

4.5. T1117 Regsvr32 creating a new process

Two common commands to execute this attack are `regsvr32 /s attackdll.dll` and `regsvr32 /s /u /i:attackfile.sct scrobj.dll`. The first command registers and runs a malicious DLL file to create a new process. The second instance is known as a squiblydoo attack which runs javascript or VBscript embedded in a `sct` file to create a new process. Two new attack variations are provided in the test set for this technique. This includes one variation where the renamed `regsvr33.exe` file attempts to download and access a remote `sct` file. The second `regsvr33.exe` variation loads a DLL file, which creates a new PowerShell process.

The two traditional analytics provided in Fig. 14 are aimed at detecting instances of `regsvr32.exe` creating a new process. However, an attacker can easily rename `regsvr32.exe` and/or `scrobj.dll` to avoid producing an alert. Any renaming of these native files results in a missed detection as shown in Table 10.

The procmonML analytic for detecting regsvr32 attacks (PA-T1117-1) is provided in Fig. 15. Relevant Event-10 parameters include the average access code (`Event10_AverageAccessCodeSource`) which corresponds to a value greater than 0 × 200000, the average number of files from the windows\system32 folder in the call trace (`Event10_AverageSystem32CallTrace`), and the average number of times where regsvr32 is listed as a target process (`Event10_ProcessAccessDest`). Additional parameters include the total process time (`pTotalTime`), the total event count (`pEventCount`), where the number of Event-18 alerts is less than 0.5 (`Event18_PipeConnect`. A single Event-10 alert where regsvr32 is the source process is generated for each new child process with an associated 0 × 1FFFFFF access code. An analysis of benign regsvr32 instances versus attack instances showed one of the primary differences being the quantity of events generated. Benign instances of regsvr32 appeared to produce approximately 40-60 total process events with the given Sysmon

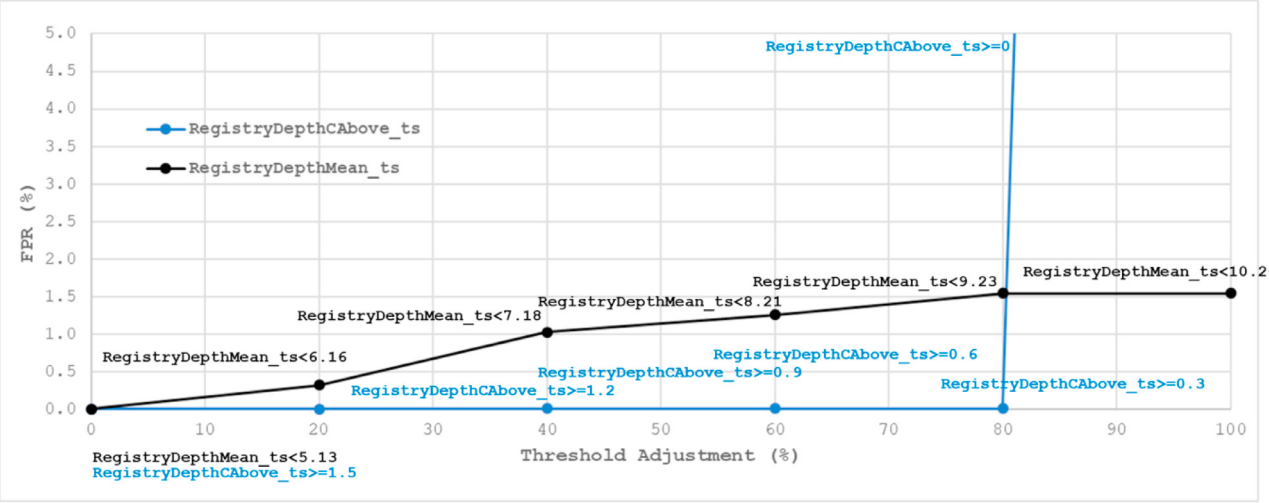

Fig. 16. Sensitivity Plot for PA-T1050-1 to detect new service creation.

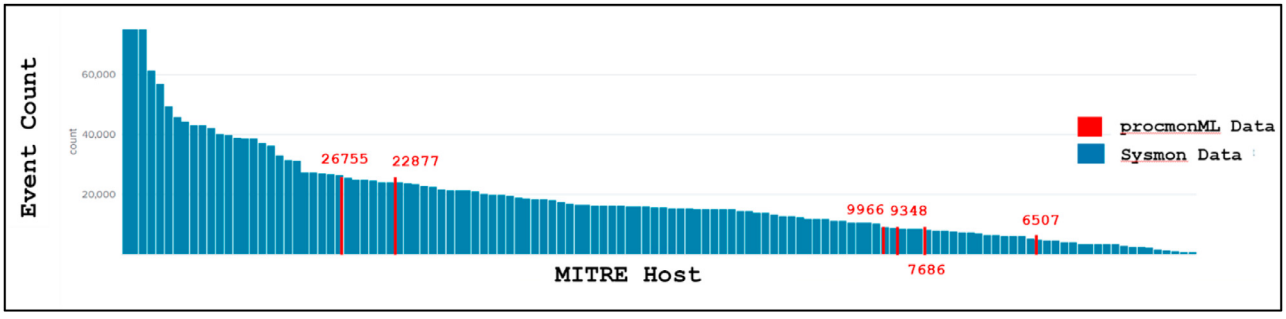

Fig. 17. Comparison of daily events generated between traditional sysmon collection and procmonML Approach.

configuration file, whereas attack instances produced significantly more events with totals in the range 80-110. The increase is primarily attributed to additional DLL loading (Event-7). Unlike the previously tested techniques, the returned FP results did not show a single process that was responsible for a large majority of the FP results. Instead, there were multiple processes each comprising 2-3% of the returned FP results. The results in Table 10 demonstrate that procmonML analytic is able to detect both the standard and the renamed regsvr33 attacks, with a FPR of 0.0077%.

4.6. Analytic sensitivity

An interesting area of future work is to perform a sensitivity analysis for each of the newly generated procmonML analytics. This will provide further insight into how resilient a particular analytic is to evasion attempts by an adversary. A sensitivity plot, as shown in Fig. 16, provides a visual tool to determine which analytic parameters are more susceptible to evasion. Fig. 16 was generated for the PA-T1050-1 analytic, which is comprised of two parameters (`RegistryDepthCAbove_ts` and `RegistryDepthMean_ts`), to detect new service creation events. The x-axis is the analytic threshold adjustment and the y-axis is the corresponding FPR. In other words, it shows how changing the analytic parameter thresholds affect the FPR.

One approach to the analytic tuning process of adjusting the parameter thresholds can be analogous to a two-player game between an adversary and a defender. An adversary's goal is to construct an attack that requires cyber defenders to loosen the analytic threshold constraints to detect the attack, at the cost of increasing the FPR. The defender's goal is to have the ability to loosen the threshold values to provide additional attack detection, while ensuring that the FPR remains at a manageable level.

In the plot below, the two analytic parameters are independently modified and the corresponding FPR is recorded. A zero-percent threshold adjustment corresponds with the original analytic threshold values. As the `RegistryDepthMean_ts` parameter threshold value is loosened from a starting value of 5.13 to a final adjustment value of 10.26, the overall analytic FPR approaches a value of approximately 1.5%. Meanwhile, the plot shows that as the `RegistryDepthCAbove_ts` parameter threshold reaches a value of 0, this results in a significant FPR increase. In this case, an attacker's objective would be to attempt to manipulate the corresponding process behavior resulting in `RegistryDepthCAbove_ts=0`.

4.7. Summary

This research demonstrates the advantage of applying host-based behavioral procmonML analytics rather than traditional signature/heuristic analytics, which are easily evaded and generally only applicable to a single use case. For the lsass credential technique, traditional analytics were aimed at detecting either the `MiniDumpWriteDump` technique or the Mimikatz variation. While more robust than other traditional analytics that were tested in this work, TA-T1003-1 failed to detect two of the more evasive `MiniDumpWriteDump` variations (ppldump and SharpDump). The procmonML analytic (PA-T1003-1) was able to detect all vari-

ations of credential dumping including two new variations that were not present in the training dataset. For both T1050 (new service creation) and T1053 (new scheduled task), we proposed analytics to detect new service and new scheduled task creation by encoding the system process activity of `services.exe` and `svchost.exe`. In both cases, traditional analytics try to match executable or PowerShell command strings. No traditional analytics were identified that could detect custom binaries that used system DLLs to accomplish the same end result. No matter the exact variation executed in these cases, the system-level processes exhibited the same registry access behavior, which was able to be captured into a single analytic for each technique. For T1055 (process injection), Event-10 features were primarily used in detecting process injection techniques. Only process injection techniques that utilized the `CreateRemoteThread` API call were detectable with traditional Sysmon analytics. The procmonML analytics were able to detect a variety of new instances of process injection attacks in the test dataset, using Event-10 behavior as a detection mechanism. For the last technique—T1117 (Regsvr32)—traditional analytics aim to detect instances where the `regsvr32.exe` binary is creating a new child process. A renamed `regsvr33.exe` binary is able to evade simple detection. The procmonML analytic for this technique relies on several Event-10 features and the overall number of events generated in order to differentiate malicious instances from benign instances. An interesting test case is found with the ppldump.exe attack. This attack provides three opportunities for detection based on the techniques that were investigated in this experiment. It is interesting to note that given the three opportunities for detection, none of the traditional analytics were able to provide any kind of detection. Meanwhile, the procmonML analytics were able to detect all three behaviors.

One of the challenges for organizations is balancing the quantity of events generated. This affects the quantity of storage needed to handle all the data in addition to the quantity of false positive events that are generated for a given analytic. Fig. 17 provides a comparison of the average daily number of procmonML process events versus traditional Sysmon events on MITRE production hosts. The proposed approach generates a comparable number of daily Sysmon alerts versus traditional event collection.

The CPU usage of the procmonML endpoint sensor on the test machines was also tracked (Average: 5-7%) and is provided in Fig. C.1 in the Appendix. Sensor debugging statistics showed that the number of Sysmon events that the sensor processed significantly affected the performance of the sensor. The calculation of the timeseries features for the registry key depth array also contributed to the computational complexity of the sensor since it requires enumerating each of the individual array elements. We hypothesize that transitioning the sensor from a PowerShell script to a lower level programming language may provide future performance improvements.

5. Conclusion

The novel procmonML approach presented in this paper provides an improved method to detect malicious host-based activity that is more resilient to adversarial evasion techniques. The ability to capture multiple system behaviors in the form of an interpretable host-based analytic offers cyber defenders an advantage in detecting more variations of attacks. The generated analytics are comprised of system behavioral parameters versus string-matching parameters. We demonstrated the ability to generate analytics in a closed test lab and apply them to detecting attacks in an enterprise production environment.

The proposed approach was applied to five different techniques and tested on 30 variations of advanced attacks. While in this particular work, analytics were generated to detect all variations of attacks for a certain technique, further reduction in false positive performance could possibly be achieved by detecting specific variations of attacks for a specific technique. Additional future work includes integrating API hooking to detect more types of techniques, and performing additional sensitivity analysis of the analytics to determine the robustness of the analytics to account for adversaries that have knowledge of these analytics and are trying to evade them. This paper covers the fundamental principles of evasion-resilient behavioral analytic development and we hope to see more future work in this domain.

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.

CRediT authorship contribution statement

Joseph W. Mikhail: Conceptualization, Methodology, Software, Validation, Formal analysis, Writing - original draft. **Jamie C. Williams:** Software, Validation, Writing - review & editing. **George R. Roelke:** Writing - review & editing, Resources.

Acknowledgements

The authors would like to acknowledge their research team members for their invaluable contributions to this effort: Cervando Banuelos, Devon Ellis, Shawn Edwards, Alyssa Rose, Brandon Werner. We would also like to thank the anonymous reviewers who provided insightful comments which led to an improvement of this work.

Funding

This work was supported by The MITRE Corporation's internal research program: MITRE Innovation Program (MIP), Project# 10MSRC20-R1.

Appendix A. procmonML Features

Given the registry depth array of individual elements d_i, where $D_{reg} = \{d_1, d_2, d_3, \ldots n\}$ and n is the number of items in the array, the following registry key depth features are defined as:

$$RegistryDepthCount_{ts} = n \tag{A.1}$$

$$RegistryDepthMean_{ts} = \frac{\sum_{i=1}^{n} D_{reg}(i)}{n} \tag{A.2}$$

$$RegistryDepthMax_{ts} = max(D_{reg}) \tag{A.3}$$

$$RegistryDepthStddev_{ts} = \sqrt{\frac{\sum_{i=1}^{n} (D_{reg}(i) - RegistryDepthMean_{ts})^2}{n}} \tag{A.4}$$

Table A.1
procmonML feature list.

Feature Name	Description
pChildCount	The number of child processes spawned
pEventCount	The total number of Sysmon events generated
pIntegrityLevel	The process integrity level encoded as a categorical value: -1 = Untrusted or Low, 10=Medium, 50=High, 100=System
pTotalTime	The total number of seconds that a process was running.
Event1_ProcessStart	The quantity of Event-1 events
Event3_NetworkConnection	The quantity of Event-3 events
Event3_DPort_Avg	The average destination port
Event3_DPort_Max	The maximum destination port
Event3_SPort_Avg	The average source port
Event3_SPort_Max	The maximum source port
Event5_ProcessTerminated	The quantity of Event-5 events
Event7_ImageLoaded	The quantity of Event-7 events
Event8_RemoteThreadSource	The quantity of Event-8 events generated where the given process is listed as the source process
Event8_RemoteThreadDest	The quantity of Event-8 events generated where the given process is listed as the destination process
Event9_RawAccessRead	The quantity of Event-9 events
Event10_AverageAccessCodeDest	The average access code where the given process is a destination process for Event-10
Event10_AverageAccessCodeSource	The average access code where the given process is a source process for Event-10
Event10_AverageCallTrace	The average number of files in the Event-10 process call trace field
Event10_AverageSystem32CallTrace	The average number of files in the call trace that exist in the windows\system32 folder
Event10_LsassAccess	The number of instances that a process accesses the lsass process
Event10_MaxAccessCodeDest	The maximum value of the access codes where the given process is a destination process for Event-10
Event10_MaxAccessCodeSource	The maximum value of the access codes where the given process is a source process for Event-10
Event10_MaxCallTrace	The maximum number of files in the Event-10 process call trace field
Event10_ProcessAccessDest	The number of instances where the given process is a destination process for an Event-10 alert
Event10_ProcessAccessSource	The number of instances where the given process is a source process for an Event-10 alert
Event11_FileCreate	The quantity of Event-11 events
Event12_RegistryCreateOrDelete	The quantity of Event-12 events
Event12_RegistryCreateCount	The quantity of Event-12 events that create a registry key/value
Event12_RegistryDeleteCount	The quantity of Event-12 events that delete a registry key/value
Event13_RegistryValueSet	The quantity of Event-13 events
Event14_RegistryRename	The quantity of Event-14 events
Event15_FileStreamCreate	The quantity of Event-15 events
Event17_PipeCreate	The quantity of Event-17 events
Event18_PipeConnect	The quantity of Event-18 events
Event22_DNS	The quantity of Event-22 events
RegistryDepthCount_ts	The total number of registry access events where the registry key depth is greater than 0
RegistryDepthMean_ts	The mean registry key depth in the distribution of registry key depths
RegistryDepthMax_ts	The maximum registry key depth in the distribution of registry key depths
RegistryDepthStddev_ts	The standard deviation in the distribution of registry key depths
RegistryDepthCAbove_ts	The average quantity of registry accesses that have a registry key depth larger than the mean depth
RegistryDepthCBelow_ts	The average quantity of registry accesses that have a registry key depth smaller than the mean depth
RegistryDepthFirstMax_ts	The index of the maximum registry key depth
RegistryDepthLongestAbove_ts	The longest consecutive number registry key accesses where the depth is greater than the mean depth
RegistryDepthLongestBelow_ts	The longest consecutive number registry key accesses where the depth is smaller than the mean depth
RegistryDepthPeaks_ts	The number of peaks in the distribution of registry key depths.

Algorithm A.1 RegistryDepthCBelow_ts.

Input: Registry key depth array, D_{reg}, with element d_i
Output: $RegistryDepthCBelow_{ts}$
$m \leftarrow mean(D_{reg})$
return $where(d_i < m)[0].size$

Algorithm A.2 RegistryDepthCAbove_ts.

Input: Registry key depth array, D_{reg}, with element d_i
Output: $RegistryDepthCAbove_{ts}$
$m \leftarrow mean(D_{reg})$
return $where(d_i > m)[0].size$

Algorithm A.3 RegistryDepthFirstMax_ts.

Input: Registry key depth array, D_{reg}, with element d_i
Output: $RegistryDepthFirstMax_{ts}$
if $RegistryDepthCount_{ts} > 0$
  return $argmax(D_{reg})$
else
  return NaN

Algorithm A.4 RegistryDepthLongestAbove_ts.

Input: Registry key depth array, D_{reg}, with element d_i
Output: $RegistryDepthLongestAbove_{ts}$
if $RegistryDepthCount_{ts} > 0$
  return $max(\_get\_length\_sequences\_where(x > RegistryDepthMean_{ts}))$
else
  return 0

Algorithm A.5 RegistryDepthLongestBelow_ts.

Input: Registry key depth array, D_{reg}, with element d_i
Output: $RegistryDepthLongestBelow_{ts}$
if $RegistryDepthCount_{ts} > 0$
  return $max(\_get\_length\_sequences\_where(x < RegistryDepthMean_{ts}))$
else
  return 0

Algorithm A.6 RegistryDepthPeaks_ts.

Input: Registry key depth array, D_{reg}, with element d_i
Output: $RegistryDepthPeaks_{ts}$
$peakcount \leftarrow 0$
while $i + 2 < RegistryDepthCount_{ts}$
  if $D_{reg}(i+1) > D_{reg}(i+2)$ *and* $D_{reg}(i+1) > D_{reg}(i)$
    $peakcount += 1$
  $i += 1$
return $peakcount$

Appendix B. Sysmon configuration for registry filtering

```
   <TargetObject name="T1060,RunKey" condition="contains">CurrentVersion\Run</TargetObject>
   <TargetObject name="T1060,RunPolicy" condition="contains">Policies\Explorer\Run
   <TargetObject name="T1050" condition="contains">CurrentControlSet\Services
   <TargetObject name="T1053" condition="contains">TaskCache</TargetObject>
   <TargetObject name="T1088" condition="contains">shell\open\command</TargetObject>
   <TargetObject name="T1484" condition="contains">Group Policy\Scripts</TargetObject>
   <TargetObject name="T1484" condition="contains">Windows\System\Scripts</TargetObject>
   <TargetObject name="T1060" condition="contains">CurrentVersion\Windows\Load</TargetObject>
   <TargetObject name="T1060" condition="contains">CurrentVersion\Windows\Run</TargetObject>
   <TargetObject name="T1060" condition="contains">CurrentVersion\Winlogon\Shell</TargetObject>
   <TargetObject name="T1060" condition="contains">CurrentVersion\Winlogon\System</TargetObject>
   <TargetObject condition="begin with">HKLM\Software\Microsoft\Windows NT\CurrentVersion\Winlogon
\Notify</TargetObject>
   <TargetObject condition="begin with">HKLM\Software\Microsoft\Windows NT\CurrentVersion\Winlogon
\Shell</TargetObject>
   <TargetObject condition="begin with">HKLM\Software\Microsoft\Windows NT\CurrentVersion\Winlogon
\Userinit</TargetObject>
   <TargetObject condition="begin with">HKLM\Software\WOW6432Node\Microsoft\Windows
NT\CurrentVersion\Drivers32</TargetObject>
   <TargetObject condition="begin with">HKLM\SYSTEM\CurrentControlSet\Control\Session
Manager\BootExecute</TargetObject>
   <TargetObject condition="begin with">HKLM\Software\Microsoft\Windows NT\CurrentVersion\AeDebug
</TargetObject>
   <TargetObject condition="contains">UserInitMprLogonScript</TargetObject>
   <TargetObject name="T1112,ChangeStartupFolderPath" condition="end with">User Shell
Folders\Startup</TargetObject>
   <TargetObject name="T1031,T1050" condition="end with">\ServiceDll</TargetObject>
   <TargetObject name="T1031,T1050" condition="end with">\ServiceManifest</TargetObject
   <TargetObject name="T1031,T1050" condition="end with">\ImagePath</TargetObject>
   <TargetObject name="T1031,T1050" condition="end with">\Start</TargetObject>
   <TargetObject name="RDP port change" condition="end with">Control\Terminal Server\WinStations
\RDP-Tcp\PortNumber</TargetObject>
   <TargetObject name="RDP port change" condition="end with">Control\Terminal Server\fSingleSession
PerUser</TargetObject>
   <TargetObject name="ModifyRemoteDesktopState" condition="end with">fDenyTSConnections</TargetO
bject>
   <TargetObject condition="end with">LastLoggedOnUser</TargetObject>
   <TargetObject name="ModifyRemoteDesktopPort" condition="end with">RDP-tcp\PortNumber</Target
Object>
   <TargetObject condition="end with">Services\PortProxy\v4tov4</TargetObject>
   <TargetObject name="T1042" condition="contains">\command\</TargetObject>
   <TargetObject name="T1122" condition="contains">\ddeexec\</TargetObject>
   <TargetObject name="T1122" condition="contains">{86C86720-42A0-1069-A2E8-08002B30309D}</
TargetObject>
   <TargetObject name="T1042" condition="contains">exefile</TargetObject>
   <TargetObject condition="end with">\InprocServer32\(Default)</TargetObject>
   <TargetObject name="T1158" condition="end with">\Hidden</TargetObject>
   <TargetObject name="T1158" condition="end with">\ShowSuperHidden</TargetObject>
   <TargetObject name="T1158" condition="end with">\HideFileExt</TargetObject>
   <TargetObject condition="contains">Classes\*\</TargetObject>
   <TargetObject condition="contains">Classes\AllFilesystemObjects\</TargetObject>
   <TargetObject condition="contains">Classes\Directory\</TargetObject>
   <TargetObject condition="contains">Classes\Drive\</TargetObject>
   <TargetObject condition="contains">Classes\Folder\</TargetObject>
   <TargetObject condition="contains">ContextMenuHandlers\</TargetObject>
   <TargetObject condition="contains">CurrentVersion\Shell</TargetObject>
   <TargetObject condition="begin with">HKLM\Software\Microsoft\Windows\CurrentVersion\explorer
\ShellExecuteHooks</TargetObject>
   <TargetObject condition="begin with">HKLM\Software\Microsoft\Windows\CurrentVersion\explorer
\ShellServiceObjectDelayLoad</TargetObject>
   <TargetObject condition="begin with">HKLM\Software\Microsoft\Windows\CurrentVersion\explorer
\ShellIconOverlayIdentifiers</TargetObject>
   <TargetObject condition="begin with">HKLM\Software\Microsoft\Windows\CurrentVersion\App
Paths\</TargetObject>
```

```
  <TargetObject condition="begin with">HKLM\SYSTEM\CurrentControlSet\Control\Terminal
Server\WinStations\RDP-Tcp\InitialProgram</TargetObject>
  <TargetObject name="T1484" condition="begin with">HKLM\Software\Microsoft\Windows
NT\CurrentVersion\Winlogon\GPExtensions\</TargetObject
  <TargetObject condition="begin with">HKLM\SYSTEM\CurrentControlSet\Services\WinSock</TargetO
bject>
  <TargetObject condition="end with">\ProxyServer</TargetObject>
  <TargetObject condition="begin with">HKLM\Software\Microsoft\Windows\CurrentVersion\Authentication
\Credential Provider</TargetObject>
  <TargetObject name="T1101" condition="begin with">HKLM\SYSTEM\CurrentControlSet\Control\Lsa
\</TargetObject>
  <TargetObject condition="begin with">HKLM\SYSTEM\CurrentControlSet\Control\SecurityProviders
\SecurityProviders</TargetObject>
  <TargetObject condition="begin with">HKLM\Software\Microsoft\Netsh</TargetObject>
  <TargetObject condition="begin with">HKLM\SYSTEM\CurrentControlSet\Control\NetworkProvider\Order
\</TargetObject
  <TargetObject condition="begin with">HKLM\Software\Microsoft\Windows NT\CurrentVersion\NetworkList
\Profiles</TargetObject>
  <TargetObject name="T1089" condition="end with">\EnableFirewall</TargetObject
  <TargetObject name="T1089" condition="end with">\DoNotAllowExceptions</TargetObject>
  <TargetObject condition="begin with">HKLM\SYSTEM\CurrentControlSet\Services\SharedAccess
\Parameters\FirewallPolicy\StandardProfile\AuthorizedApplications\List</TargetObject>
  <TargetObject condition="begin with">HKLM\SYSTEM\CurrentControlSet\Services\SharedAccess
\Parameters\FirewallPolicy\DomainProfile\AuthorizedApplications\List</TargetObject>
  <TargetObject name="T1103" condition="begin with">HKLM\Software\Microsoft\Windows
NT\CurrentVersion\Windows\Appinit_Dlls\</TargetObject>
  <TargetObject name="T1103" condition="begin with">HKLM\Software\Wow6432Node\Microsoft\Windows
NT\CurrentVersion\Windows\Appinit_Dlls\</TargetObject>
  <TargetObject condition="begin with">HKLM\SYSTEM\CurrentControlSet\Control\Session
Manager\AppCertDlls\</TargetObject>
  <TargetObject name="T1137" condition="contains">Microsoft\Office\Outlook\Addins\</TargetObject>
  <TargetObject name="T1137" condition="contains">Office Test\</TargetObject>
  <TargetObject name="Context,ProtectedModeExitOrMacrosUsed" condition="contains">Security\Trusted
Documents\TrustRecords</TargetObject>
  <TargetObject name="T1176" condition="contains">Internet Explorer\Toolbar\</TargetObject>
  <TargetObject name="T1176" condition="contains">Internet Explorer\Extensions\</TargetObject
  <TargetObject name="T1176" condition="contains">Browser Helper Objects\</TargetObject
  <TargetObject condition="end with">\DisableSecuritySettingsCheck</TargetObject>
  <TargetObject condition="end with">\3\1206</TargetObject>
  <TargetObject condition="end with">\3\2500</TargetObject>
  <TargetObject condition="end with">\3\1809</TargetObject>
  <TargetObject condition="contains">{AB8902B4-09CA-4bb6-B78D-A8F59079A8D5}\</TargetObject>
  <TargetObject name="Alert,Sysinternals Tool Used" condition="end with">\EulaAccepted<
/TargetObject>
  <TargetObject condition="end with">\UrlUpdateInfo</TargetObject>
  <TargetObject condition="end with">\InstallSource</TargetObject>
  <TargetObject name="T1089,Tamper-Defender" condition="end with">\DisableAntiSpyware</Target
Object>
  <TargetObject name="T1089,Tamper-Defender" condition="end with">\DisableAntiVirus</Target
Object>
  <TargetObject name="T1089,Tamper-Defender" condition="end with">\SpynetReporting</TargetObject>
  <TargetObject name="T1089,Tamper-Defender" condition="end with">DisableRealtimeMonitoring</
TargetObject>
  <TargetObject name="T1089,Tamper-Defender" condition="end with">\SubmitSamplesConsent<
/TargetObject>
  <TargetObject name="T1088" condition="end with">HKLM\Software\Microsoft\Windows\CurrentVersion
\Policies\System\EnableLUA</TargetObject
  <TargetObject name="T1088" condition="end with">HKLM\Software\Microsoft\Windows\CurrentVersion
\Policies\System\LocalAccountTokenFilterPolicy</TargetObject>
  <TargetObject name="T1089,Tamper-SecCenter" condition="end with">HKLM\Software\Microsoft\Security
Center\</TargetObject>
  <TargetObject name="T1089,Tamper-SecCenter" condition="end with">SOFTWARE\Microsoft\Windows
\CurrentVersion\Policies\Explorer\HideSCAHealth</TargetObject>
  <TargetObject name="T1138,AppCompatShim" condition="begin with">HKLM\Software\Microsoft\Windows
NT\CurrentVersion\AppCompatFlags\Custom</TargetObject>
```

```
  <TargetObject name="T1138,AppCompatShim" condition="begin with">HKLM\Software\Microsoft\Windows
NT\CurrentVersion\AppCompatFlags\InstalledSDB</TargetObject>
  <TargetObject condition="contains">VirtualStore</TargetObject>
  <TargetObject name="T1183,IFEO" condition="begin with">HKLM\Software\Microsoft\Windows
NT\CurrentVersion\Image File Execution Options\</TargetObject
  <TargetObject condition="begin with">HKLM\Software\Microsoft\Windows\CurrentVersion\WINEVT\<
/TargetObject
  <TargetObject name="Tamper-Safemode" condition="begin with">HKLM\SYSTEM\CurrentControlSet\Control
\Safeboot\</TargetObject
  <TargetObject name="Tamper-Winlogon" condition="begin with">HKLM\SYSTEM\CurrentControlSet\Control
\Winlogon\</TargetObject>
  <TargetObject name="Context,DeviceConntectedOrUpdated" condition="end with">\FriendlyName<
/TargetObject>
  <TargetObject name="Context,MsiInstallerStarted" condition="is">HKLM\Software\Microsoft\Windows
\CurrentVersion\Installer\InProgress\(Default)</TargetObject>
  <TargetObject name="Tamper-Tracing" condition="begin with">HKLM\Software\Microsoft\Tracing
\RASAPI32</TargetObject>
  <TargetObject name="InvDB-Path" condition="end with">\LowerCaseLongPath</TargetObject>
  <TargetObject name="InvDB-Pub" condition="end with">\Publisher</TargetObject>
  <TargetObject name="InvDB-Ver" condition="end with">\BinProductVersion</TargetObject>
  <TargetObject name="InvDB-DriverVer" condition="end with">\DriverVersion</TargetObject>
  <TargetObject name="InvDB-DriverVer" condition="end with">\DriverVerVersion</TargetObject>
  <TargetObject name="InvDB-CompileTimeClaim" condition="end with">\LinkDate</TargetObject>
  <TargetObject name="InvDB" condition="contains">Compatibility Assistant\Store\</TargetObject>
```

Appendix C. Additional Figures

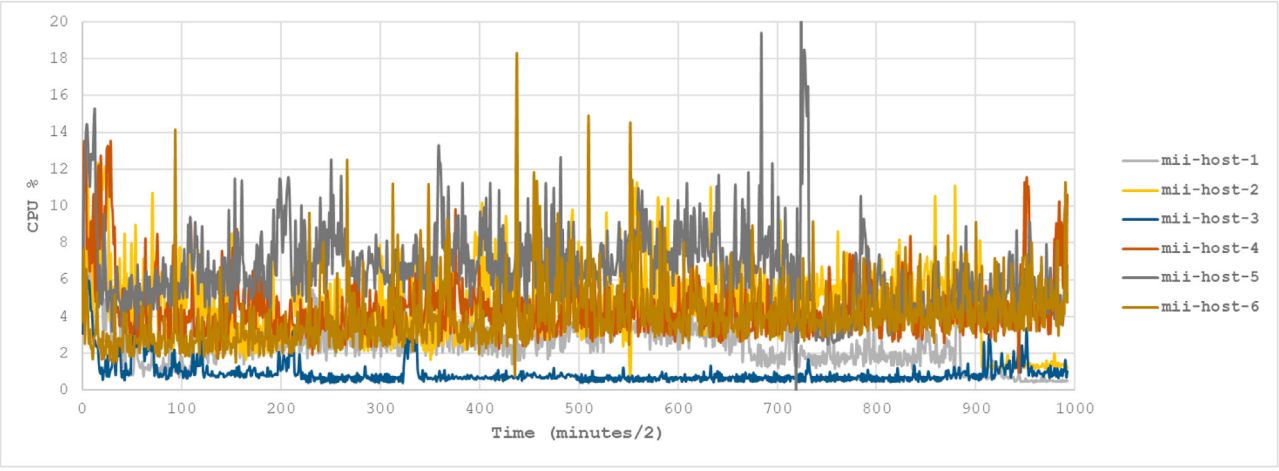

Fig. C.1. Average procmonML Sensor CPU Usage.

Appendix D. Model Parameter Definitions

estimator: The type of classifier.
n_estimators: The number of trees in the forest.
class_weight: The weights associated with the classes.
criterion: The function to measure the node split quality.
bootstrap: The bootstrap method samples a dataset with replacement.
oob_score: Whether to use out-of-bag samples.
min_samples_split: The minimum number of samples used to determine a split.
min_samples_leaf: The minimum number of samples used to determine a leaf node.
min_precision: The minimum precision that a rule must meet to be extracted from an estimator.
min_weight_fraction_leaf: The minimum weighted fraction of the sum total of weights required to be at a leaf node.
min_impurity_decrease: A node will be split if this split induces a decrease of the impurity greater than or equal to this value.
min_impurity_split: Threshold for early stopping in tree growth.
max_samples: The number of samples to draw from the input data.
max_leaf_nodes: Grow trees with max_leaf_nodes in best-first fashion.
max_features: The number of features to include when looking for the best split.
max_depth: The maximum depth of the tree.

References

Baldin, A., 2019. Best practices for fighting the fileless threat. Netw. Secur. 2019 (9), 13–15. doi:10.1016/S1353-4858(19)30108-4.
Block, F., Dewald, A., 2019. Windows memory forensics: detecting (Un) intentionally hidden injected code by examining page table entries. Digital Invest. 29, S3–S12.
Borisaniya, B., Patel, D.others, 2015. Evaluation of modified vector space representation using adfa-ld and adfa-wd datasets. J. Inf. Secur. 6 (03), 250.
Bridges, Robert A., et al., 2019. A survey of intrusion detection systems leveraging host data. ACM Comput. Surv. (CSUR) 52 (6), 1–35.
Cabau, George, Buhu, Magda, Oprisa, Ciprian Pavel, 2016. Malware classification based on dynamic behavior. In: Proceedings of the 18th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC). IEEE.
Chawla, A., Lee, B., Fallon, S., Jacob, P., 2018. Host based intrusion detection system with combined CNN/RNN model. In: Proceedings of the *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 149–158.
Creech, Gideon, Hu, Jiankun, 2014. A semantic approach to host-based intrusion detection systems using contiguous and discontiguous system call patterns. IEEE Trans. Comput. 63 (4), 807–819 (2014).
CrowdStrike. 2019. Global threat report. Retrieved from https://www.crowdstrike.com/blog/2019-global-threatreport-shows-it-takes-innovation-and-speed-to-winagainst-adversaries/.
C. de Plaa, "Red Team Tactics: Combining Direct System Calls and sRDI to bypass AV/EDR," Jun. 19, 2020. https://outflank.nl/blog/2019/06/19/red-team-tactics-combining-direct-system-calls-and-srdi-to-bypass-av-edr/ (accessed Mar. 15, 2020).
Delpy, B., 2014. Mimikatz.
Endgame, 2020. EQL analytics library. EQL Anal. Library. http://eqllib.readthedocs.io. (accessed Mar. 20, 2020).
Fireeye, Mar. 20, 2020. SharPersist Windows persistence toolkit written in c#. Github. accessed Mar. 20, 2020. https://github.com/fireeye/sharpersist.
Graeber, Matt, Sep. 28, 2017. Subverting trust in windows. Specter Ops. Accessed: Mar. 15, 2020. [Online]. Available: https://specterops.io/assets/resources/SpecterOps_Subverting_Trust_in_Windows.pdf.
Hammou, S., Feb. 06, 2018. MalwareFox AntiMalware (zam64.sys) - Privilege Escalation through Incorrect Access Control. Reverse Engineering 0x4 Fun.
Hendler, D., Kels, S., Rubin, A., 2018. Detecting malicious PowerShell commands using deep neural networks. In: Proceedings of the Asia Conference on Computer and Communications Security, pp. 187–197.
Hubballi, Neminath, Biswas, Santosh, Nandi, Sukumar, 2010. Layered higher order n-grams for hardening payload based anomaly intrusion detection. In: Proceedings of the International Conference on Availability, Reliability and Security. IEEE.
Jethva, Brijesh, et al., 2020. Multilayer ransomware detection using grouped registry key operations, file entropy and file signature monitoring. J. Comput. Secur. Preprint 1–37.
Jiang, Guofei, Chen, Haifeng, Ungureanu, Cristian, Yoshihira, Kenji, 2007. Multiresolution abnormal trace detection using varied-length n-grams and automata. IEEE Trans. Syst. Man Cybern. Part C: Appl. Rev. 37 (1), 86–97 (2007).
G. Kim, H. Yi, J. Lee, Y. Paek, and S. Yoon, "LSTM-based system-call language modeling and robust ensemble method for designing host-based intrusion detection systems," *arXiv preprint* arXiv:1611.01726, 2016.
Klein, E., Kotler, I., 2019. Windows Process Injection in. Black Hat.
Kumar, S.others, 2020. An emerging threat Fileless malware: a survey and research challenges. Cybersecurity 3 (1), 1–12.
Lee, Wenke, Stolfo, Salvatore J., 1998. Data mining approaches for intrusion detection. In: Proceedings of the Usenix Security.
Lee, Wenke, Stolfo, Salvatore J., Chan, Philip K., 1996. Learning patterns from unix process execution traces for intrusion detection. In: Proceedings of the AAAI Workshop on AI Approaches to Fraud Detection and Risk Management, pp. 50–56.
Liu, M., Xue, Z., Xu, X., Zhong, C., Chen, J., Nov. 2018. Host-based intrusion detection system with system calls: review and future trends. ACM Comput. Surv. 51 (5s). doi:10.1145/3214304.
Liu, Wu, et al., 2011. Behavior-based malware analysis and detection. In: Proceedings of the First International Workshop on Complexity and Data Mining. IEEE.
Mansfield-Devine, S., 2017. Fileless attacks: compromising targets without malware. Netw. Secur. 2017 (4), 7–11. https://doi.org/10.1016/S1353-4858(17)30037-5.
Matsuda, W., Fujimoto, M., Mitsunaga, T., 2019. Real-time detection system against malicious tools by monitoring DLL on client computers. In: Proceedings of the IEEE Conference on Application, Information and Netw. Secur. (AINS), pp. 36–41.
Mavroeidis, V., Jøsang, A., 2018. Data-driven threat hunting using sysmon. In: Proceedings of the 2nd International Conference on Cryptography, Security and Privacy, pp. 82–88.
Mikhail, J.W., Fossaceca, J.M., Iammartino, R., 2019. A semi-boosted nested model with sensitivity-based weighted binarization for multi-domain network intrusion detection. ACM Trans. Intell. Syst. Technol. (TIST) 10 (3), 1–27.
Moustafa, N., Slay, J., 2015. UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set). In: Proceedings of the Military Communications and Information Systems Conference (MilCIS), pp. 1–6.
Mouttaqi, T., Rachidi, T., Assem, N., 2017. Re-evaluation of combined Markov-Bayes models for host intrusion detection on the ADFA dataset. In: Proceedings of the Intelligent Systems Conference (IntelliSys), pp. 1044–1052.
Pedregosa et al., 2020 "Scikit-Learn: Machine Learning in Python".
Pfahringer, B., 2000. Winning the KDD99 classification cup: bagged boosting. ACM SIGKDD Explor. Newslett. 1 (2), 65–66.
PowerSploit*PowerSploit - A PowerShell Post-Exploitation Framework*. 2020.
Red Canary, "2020 Threat Detection Report." https://redcanary.com/threat-detection-report/ (accessed Jul. 01, 2020).
Ribeiro, M.T., Singh, S., Guestrin, C., 2016, August. " Why should I trust you?" Explaining the predictions of any classifier. In: Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining, pp. 1135–1144.
Roth, F., Patzke, Thomas, 2020. Generic Signature Format for SIEM Systems. Github. https://github.com/Neo23x0/sigma.
Russinovich, M., 2009. Sysinternals suite. Microsoft TechNet.
Sekharan, S.S., Kandasamy, K., 2017. Profiling SIEM tools and correlation engines for security analytics. In: Proceedings of the International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET), pp. 717–721.
Shackleford, D., 2016. SANS 2016 security analytics survey. SANS Institute, Swansea.
SharpDump. *SharpDump is a C# port of PowerSploit's Out-Minidump.ps1 functionality*. 2018.
Siadati, H., Saket, B., Memon, N., 2016. Detecting malicious logins in enterprise networks using visualization. In: Proceedings of the IEEE Symposium on Visualization for Cyber Security (VizSec), pp. 1–8.
B. E. Strom et al., "Finding cyber threats with ATT&CK-based analytics," The MITRE Corporation, Tech. Rep., 2017.
Sudqi Khater, B., Wahab, A., Bin, A.W., Idris, M.Y.I.B., Abdulla Hussain, M., Ahmed Ibrahim, A., 2019. A lightweight perceptron-based intrusion detection system for fog computing. Appl. Sci. 9 (1), 178.
Tajoddin, Asghar, Abadi, Mahdi, 2019. RAMD: registry-based anomaly malware detection using one-class ensemble classifiers. Appl. Intell. 49 (7), 2641–2658.
The MITRE Corporation, 2020. Cyber analytics repository. Cyber Anal. Reposit.. http://car.mitre.org. (accessed Mar. 20, 2020).
Tobiyama, S., Yamaguchi, Y., Shimada, H., Ikuse, T., Yagi, T., 2016. Malware detection with deep neural network using process behavior. In: Proceedings of the IEEE 40th Annual Computer Software and Applications Conference (COMPSAC), 2, pp. 577–582.
M. J. Turcotte, A. D. Kent, and C. Hash, "Unified host and network data set," ArXiv e-prints, vol. 1708, 2017.
Ussath, M., Jaeger, D., Cheng, F., Meinel, C., 2016. Advanced persistent threats: Behind the scenes. In: Proceedings of the Annual Conference on Information Science and Systems (CISS), pp. 181–186.
Verma, Miki E., Bridges, Robert A., 2018. "Defining a metric space of host logs and operational use cases. In: *Proceedings of the* IEEE International Conference on Big Data (Big Data). IEEE.
WinSW Windows Service Wrapper, Mar. 20, 2020. WinSW Windows Service Wrapper. Github. (accessed Mar. 20, 2020). https://github.com/winsw/winsw.
Yuxin, Ding, Xuebing, Yuan, Di, Zhou, Li, Dong, Zhanchao, An, 2011. Feature representation and selection in malicious code detection methods based on static system calls. Comput. Secur. 30 (6), 514–524 (2011).

Joseph Mikhail is a member of the technical staff at The MITRE Corporation. He completed his Doctor of Engineering at George Washington University in 2019, where his research work involved applying a single classification algorithm to both traditional and 802.11 network intrusion detection. He earned his B.S. and M.S degrees in engineering from Virginia Tech and George Mason University respectively. His current primary research interests include machine learning, cybersecurity, and computer architecture. He is a licensed in the state of Virginia as a Professional Engineer (Computer Engineering).

Jamie Williams is a Lead Cyber Adversarial Engineer at The MITRE Corporation where he works on various efforts involving security operations and research, specializing in adversary emulation and analysis of behavior-based detections. He is also a member of both the MITRE ATT&CK and ATT&CK Evaluations teams. Before joining MITRE, Jamie received his M.S. in Information Systems Engineering from Johns Hopkins University and his B.S. in Information Systems from the University of Maryland, Baltimore County (UMBC).

George Roelke leads MITRE's internal research and development program for cyber security. He leads a team of researchers creating technologies to solve hard problems for the Government and the public. From 2011 to 2016, Dr. Roelke was a program manager in the at the Defense Advanced Research Projects Agency (DARPA). He has a Doctorate in Electrical Engineering and a Master of Science in Computer Engineering from the Air Force Institute of Technology, and a Bachelor of Computer Engineering from the Georgia Institute of Technology.