



A Federated Machine-Learning Paradigm for Scalable Cyber Threat Detection

John A. Emanuello and Andrew Golczynski

The ubiquity of cyber threats, coupled with the speed and scale at which they can attack society's critical cyber infrastructure has made the protection of these systems a major technical challenge. While great strides have been made toward some level of automation to aid network defenders, many of the tools are based on brittle signatures, which fail to detect novel cyber threats. This combination of circumstances has driven a small-but-growing body of research and development around the application of artificial intelligence (AI) and machine learning (ML) tools to the detection of malicious behavior in real time.

There are numerous practical and engineering challenges in fully bringing AI to bear on the cyber threat hunting problem. At the heart of these challenges is the incompatibility between the need for a detailed view of network state as captured by sensors, and the bandwidth constraints associated with moving such massive data volumes to a centralized repository for AI model training. The federated learning paradigm, wherein models are trained in a distributed fashion without the need for data aggregation, presents a potential strategy for training ML models for cybersecurity. In this article, we describe a federated ML paradigm that is consistent with the detection of malicious cyber activity in near real time, test on a benchmark dataset, and conduct an analysis of the practical implications for deploying such a model on a real network.



Background

The difficulty of cybersecurity lies in the asymmetric advantages the adversary holds. Indeed, defenders must protect all assets against all their possible vulnerabilities; whereas, an attacker need only exploit a single vulnerability to gain access on a network and set into motion a multistage cyberattack. Armed with a seemingly unlimited number of tools, advanced persistent threats (APTs) are highly sophisticated and motivated, enabling them to relentlessly target our most sensitive networks, including those that support critical infrastructure.

APT-style attacks are rarely composed of a single action; rather, they typically contain numerous events which constitute multiple stages of an attack. The progression of these stages is often modeled by ontologies such as Lockheed Martin's Cyber Kill Chain, which tracks and organizes detectable behaviors from individual log events or collections thereof [1, 2]. The subtlety of attacker behaviors, especially from sophisticated actors, makes detection difficult, with discovery often only happening after the goals of the attack have been accomplished.

It is worth mentioning that this is not an abstract problem. Digital systems underpin nearly all aspects of modern societies, including economic institutions, critical infrastructure, and even democracy itself. From a commercial standpoint, malicious cyber actors are motivated by economic espionage, especially the theft of intellectual property, which by some estimates can cost a company billions of dollars in revenue losses alone [3]. For democracies, cyberattacks could disrupt elections and erode public confidence in democratic institutions. As such, robust measures to secure cyber systems are critical components of safeguarding societal stability, economic resilience, and national security.

Intrusion detection systems

The evidence of attacks on an enterprise network can be captured by a high number of disparate sensors, logging network traffic, cloud telemetry, end point activity, etc. As such, development of automated approaches to intrusion detection have been an area of interest at least since the late 1980's [20]. These intrusion detection systems (IDS) have historically relied heavily on signature-based rules, which describe a known malicious activity that, upon matching an

observed behavior, alert a human defender to investigate. These rules include byte patterns in network traffic packets or attribute patterns in host logs [4, 5].

However, as these rules can only describe known malicious behavior, APTs can easily defeat these signatures. For example, if a signature includes a specific rule involving a byte pattern in network packets, the adversary can simply fragment packets or otherwise change the contents of packets to defeat the rule and still perpetrate the attack. As such, these signatures must be constantly updated in an at least partially manual process that often consumes more time than the adversary needs to implement a countermeasure. Further in the adversary's favor is the wide availability of such rules to exploit in order to misdirect defenders or otherwise obfuscate their attacks; indeed, by overloading defenders with alerts, they can make it extremely difficult for defenders to decide what to focus on. All told, these drawbacks are driving a body of work to apply ML to create more flexible iterations of IDS.

Given the extremely high degree of variance of normal user activity, these models must be trained on data which sufficiently captures the network's baseline, and this can only be done when data is sampled at a high rate across the assortment of sensors. Contrary to centralized ML paradigms, the data volumes and bandwidth constraints of an enterprise network prohibit the ingestion of sufficient amounts of data to a single compute server. These unique constraints suggest that federated learning may be a potential paradigm for training autonomous threat hunting tools. More specifically, such approaches support the training of ML models on multiple compute nodes, each with their own training data (potentially drawn from different distributions) that feed a global model that can be deployed at the edge for inference.

It should be noted that, given the evolving nature of cyber threats and the high degree of variance of normal activities between different networks, the availability of a labeled dataset would not be conducive to the task at hand. Thus, it is far more effective to train these ML models in unsupervised fashion, as anomaly detectors, rather than as explicit malicious behavior classifiers. Algorithms such as clustering, isolation forests, and autoencoders are well suited for anomaly detection and are integrated in widely used software packages, making them easy for researchers and cybersecurity analysts to use.

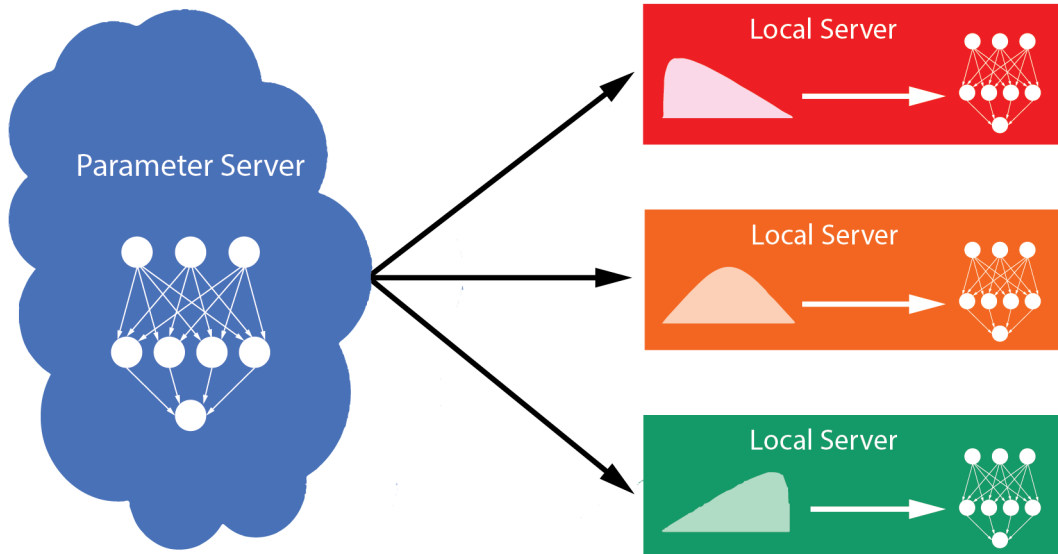


FIGURE 1. Federated learning is a machine learning setting where multiple entities (i.e., local servers) collaborate to train a model, under the management of a central parameter server. Each client’s raw data is stored locally and is not exchanged or transferred; instead, local servers complete some number of training steps on the model, and their feedback is aggregated by the central server that serves an updated model for either further training by the local servers or deployment.

Neural network approaches for intrusion detection

In the approach we outline below, we appeal to an autoencoder (AE), which is a neural network that is designed to be an identity function on the input space, and is composed of two functions: an encoder and a decoder [6]. More specifically, the training process of an autoencoder $f = g \circ h$ results in a nonlinear encoder $h: \mathbb{R}^n \rightarrow \mathbb{R}^m$ and decoder $g: \mathbb{R}^m \rightarrow \mathbb{R}^n$ functions, where $m \ll n$ and the objective function is mean squared error loss with respect to the standard Euclidean distance between input and output: $\|x - f(x)\|_2$. We note that this choice of relative dimension sizes results in what is called an “undercomplete autoencoder” and is, in effect, a nonlinear analogue principal component analysis [6]. By construction, (undercomplete) AEs learn salient information about the training data, with the compression/reconstruction process failing on outliers or other data drawn from distributions vastly different from that of the training set. This property allows an AE to indicate when a given datum was drawn from a novel distribution, making the architecture ideal for anomaly detection [7, 8].

The choice of a neural network as a proposed architecture for detecting malicious cyber activity

has been studied in the literature, and shows promise as a more flexible alternative approach from traditional signature-based detection schemes [9, 10]. While cyber telemetry logs largely contain categorical data points, there are techniques such as *log2vec* that can transform these logs into numerical data points that are ingestible by neural network architectures [9, 11, 12].

Our prior work demonstrates that the numeric representations of cyber logs can be learned in tandem with detection tasks [12]. Here, our NLP-inspired approach to anomaly detection requires us to embed our preprocessed records into a semantically relevant vector space, which is the input to an AE-based anomaly detector at inference. We tested this technique on the Operationally Transparent Cyber (OpTC) dataset created by Defense Advanced Research Projects Agency (DARPA) [17, 18], which consists of network and host logging collected from a network of hundreds of Windows hosts over a one-week period. The activity represents both normal (benign) user activity, as well as logging from an APT-inspired red team attack over the course of three days, during which numerous machines were under different phases of a multi-staged attack. Our technique was able to detect high volumes of anomalous

activities during the red team attack, and very low volumes of anomalies when there were no attacks.

A federated-learning paradigm for intrusion detection

Federated learning (FL) is a relatively new concept in the field of AI/ML that involves training a model across multiple devices, which are often called nodes or learners. Each node trains the model separately on its own data and shares model information with a central server [13]. This central server then aggregates the contributions of the participating nodes to produce a model, which is shared back to the nodes for further training or inference. There are several flavors of training archetypes (e.g., synchronous and asynchronous) and aggregation techniques which are covered in the literature. For our purposes, we assume that the model in question is some kind of neural network and that a single training step of the model is as follows:

1. The central server transmits the most current model to each of the local servers.
2. Each local server feeds a portion of its local data through the model and computes the gradients of the mutually agreed upon objective function and transmits these gradients back to the central server.
3. The central server aggregates the gradients from the local servers and performs a model update.

Federated ML presents a viable solution by distributing the learning process to the network's edge, where the data is generated. This paradigm balances the need for models to be trained on realistic, live data with the practical constraints presented in cybersecurity [13, 14]. Rather than transmitting raw data to a centralized server, federated learners utilize local computation and collaboration among networked devices to train AI models. This decentralized approach significantly reduces the need for transferring large amounts of data across the network, alleviating bandwidth constraints and minimizing latency issues.

Another consideration is the false positive problem. Indeed, if AI/ML is to be a force multiplier in cybersecurity, any solution must be careful not to produce more false positives than a human analyst can adjudicate. In fact, a major problem with current

systems is that the number of alerts defenders must sift through are unmanageable. If AI/ML solutions are to attain wide adoption, they must add analytic value rather than generate unhelpful alerts. In a FL paradigm, training on a wider variety of data facilitates better approximation of the true distribution of the "data in the wild." However, in a FL paradigm, the nonuniform occurrence of certain data points across the unified training set allows federated models to still learn salient information about these points, and to pass those insights on to a deployed model.

To see the importance of such a consideration in cybersecurity, note that normal activities vary widely across users; for example, certain users have a higher propensity to run programs like Microsoft Excel than others. This means that, for an ML model trained on host-based logs, if a user who rarely launches Excel simply does so, the model would mark the behavior as anomalous. However, simply launching Excel is hardly indicative of a cyberattack. Hence, in a FL paradigm, it is possible to aggregate the vast number of normal activities within a model, in a manner that is reminiscent of balancing the "bias-variance trade-off" [15].

Given these strengths, we propose that an FL paradigm could be employed in protection of an enterprise computer network to discover potentially malicious activity in a way that is both robust to evolving cyber threats and capable of alleviating alert fatigue that exists due to current rules-based IDS.

Early after its inception, a key selling point for federated models was their purported ability to keep local data private [13, 14]. As cyber systems often contain sensitive information such as personally identifiable information (PII), proprietary information/intellectual property, or any information that could cause harm if released in the public domain, one could imagine that FL techniques could be employed across organizations to produce community-driven AI/ML models for cybersecurity, without risking the disclosure of the sensitive information associated with those systems.

However, as noted in the literature, such privacy claims have been grossly overblown [15, 16]. Indeed, ML model updates can leak several varieties of information to even casually motivated attackers. More specifically, attackers can infer properties of the training data or even reconstruct specific training examples. While the amount of work required to extract

this sort of information from ML models can be quite expensive, the economics of cybersecurity make it so that the information to be extracted is worth the resources. Absent advances in techniques such as homomorphic encryption, we believe that the technology is not sufficiently able to mitigate the risks associated with cross-organizational federated learners.

Our approach

We implement a FL paradigm across five different servers, each of which stand in for a single host on an enterprise network. Each server contains data from exactly one host in our dataset, and no data is shared across servers. Using Tensorflow's distributed MultiWorkerMirrored strategy, we instantiate a common model across five g4dn.8xlarge AWS EC2 instances, with one of the workers also playing the role of the global server for aggregating gradients and serving model updates.

The precise model architecture is an implementation that is similar to that of our previous work [12]. The input is a log record corresponding to single network or host activity on a computer (e.g., opening a network flow, editing a registry key, etc). Each record consists of 27 fields (e.g., user name, IP address, path, registry key, etc.), each of which can take on any of a variety of values, which we shall call "words." Each of the 27 words y_i of the record is fed into an embedding layer to obtain a vector representation of each in \mathbb{R}^8 . The 27 eight-dimensional word embeddings are then concatenated into a 216-dimensional vector V , which corresponds to the embedding of the record. The record embedding is fed into an AE with a single hidden layer mapping the record into \mathbb{R}^8 and back to \mathbb{R}^{216} . This reconstruction of the record embedding, \hat{V} is the first output of the model. We then break \hat{V} back into its 27 component vectors, each of which is fed into a common dense layer to effectively compute word-by-word probability distributions across the entire vocabulary space (\hat{y}_i). The objective for the neural network is a two-term loss function, which balances the ability of the network to reconstruct the record embeddings with its ability to predict the component words that constructed the record from the reconstructed embedding. More succinctly, the

record-level loss is $Loss = \sum_{i=1}^{27} CEL(\hat{y}_i, y_i) + \alpha \|V - \hat{V}\|$, where CEL is Tensorflow's sparse categorical cross entropy loss from logits and α is a regularization term to keep the reconstruction loss of the internal AE

roughly on par with the CEL term. We experimented with different values and settled on $\alpha = 100$ for this task, which puts the two loss terms on roughly equal magnitude. Note that this is different from our previous work, where we used $\alpha = 5$ for data drawn from single hosts.

Results from our previous work demonstrate that the model is both effective at learning semantically relevant representations for the words in the records as well as providing a useful anomaly detection scheme on cybersecurity logs. Our goal here is to demonstrate how such a model could be employed in federated fashion to learn on an even larger scale than previously demonstrated, and to demonstrate that such models remain effective at the downstream task of detecting malicious cyber activity in the OpTC dataset.

OpTC dataset

As in our previous work, we appeal to the OpTC dataset. As mentioned above, these data consist of network and host logging collected from a network of hundreds of Windows hosts over a one-week period, representing normal (i.e., benign) user activity [17, 18]. Over a three-day period within this week, a set of red team actors also worked to perform various penetration tests against the network, seeking to infiltrate the network, ensure persistence, and carry out increasingly complex attacks over time. As the data contains logs from hosts that were and were not attacked by the red team, and each host that was attacked was only attacked during a proper subperiod of the three-day attack, this data constitutes a strong benchmark for intrusion detection techniques.

Specifically, we used the ECAR collection (so named as its format is an extension of the Mitre Cyber Analytics Repository, or CAR, data format), which consisted of combined network and host metadata logging drawn from Sysmon, Procmon, and other sensors [19]. While each record contained only a handful of fields, 58 unique fields were present across the entirety of the raw data, including the network five tuple (source and destination IPs/ports and protocol), image and module paths, and usernames. To prepare the data for use in our experiments, we applied a very-light handed preprocessing approach to the logs from five hosts, namely sysclient 0201-0205 to create a vocabulary consisting of terms that were tokenized. We chose to follow the exact same preprocessing pipeline, except where noted below.

These steps resulted in a vocabulary size of 10,667 words across the training and test corpora:

1. **Dropping unnecessary features:** Of the 58 keys in the ECAR file, only 27 were kept for processing. The remaining fields were determined to be insufficiently relevant to our chosen task/approach and were dropped.
2. **Feature prefixing:** Some terms may have specific meanings depending on which feature they are associated with. For example, the number 443 has a specific meaning in the “dest port” field, but that meaning would not be preserved in other fields. To ensure that these meanings are respected, values of select features were prefixed with their feature name (e.g., “443” would be mapped to the term “PORT_443” when it is found in the “dest port” field).
3. **Connection time bucketing:** Network activity records contain start and end times for the connection; to discretize these values, connection durations were calculated and bucketed into SMALL, MEDIUM, and LARGE buckets based on manual inspection of the distribution of connection durations.
4. **Path/file name extraction:** File paths and registry keys can contain machine-specific sets of directories, even when considering a common item (e.g., “xyz.dll” might be found in different directory locations on separate machines, even though the underlying file is the same). To correct for this, such paths are reduced to only the file name. Similarly, when parsing command line input, only the name of the executed program is maintained (i.e., the full path and any arguments are dropped).
5. **/24 CIDR subnet extraction:** IPv4 addresses were, in many cases, too sparse across the dataset to be well-represented, so we opted to remove the last octet from each address and only use the /24 subnet. While this did reduce the resolution of the representation of our addresses, we gained additional robustness by aggregating rare terms into a smaller number of representations.
6. **Ephemeral port aggregation:** Ports used for outgoing network traffic are typically arbitrarily chosen from the high end (i.e., greater than 49151) of the range of valid ports. These choices carry no real meaning, and needlessly expand the vocabulary space, and

so such ports were replaced with a generic “EPHEMERAL_PORT” token.

7. **Removal of rare terms:** After performing the above tokenization, terms that appeared fewer than 10 times (determined by examination of the distribution of term frequencies) over our training corpus were discarded and replaced with a generic “OBSCURE_TERM” token.
8. **Conversion to vocabulary indices:** All remaining vocabulary tokens were indexed, and each token was replaced with its index. Empty fields (e.g., file path fields are empty for network traffic events) were replaced with a “NULL_TERM” token.

With the vocabulary determined, we needed to select a training corpus for the neural network model described above. Our goals were primarily to measure the efficacy of the model in detecting red team activity, and to assess the model’s ability to generalize to normal activities rarely-or-never seen by the model. To accomplish this, we chose to train on sysclient 0203, sysclient 0204, and sysclient 0205. The choice of these three hosts is primarily motivated by the fact they had the largest volumes across the five hosts, at roughly 24 million records each (the other two hosts had roughly 10 million fewer records available). Given the data volumes, we restricted the training set to only include the three-day period preceding the red team events. Test data constituted the three-day period corresponding to the red team exercise, during which sysclient 0201, sysclient 0203, and sysclient 0205 were attacked. Sysclient 0202 and sysclient 0204 contain no red team activity, and we include model inference results from these hosts for comparison. In sum, our test set included roughly 45 million records across the five hosts.

We also note that, due to restrictions in Tensorflow’s MultiWorkerMirrored distributed learning strategy, we had to ensure that every node had precisely the same batch size and same number of batches. We chose to limit the number of records from each node in the training set to be precisely 24 million records.

Neural network training paradigm

We do wish to note one key difference in the training paradigm between this and our previous works. Previously, we constructed a custom Tensorflow training loop within Python, wherein we selected

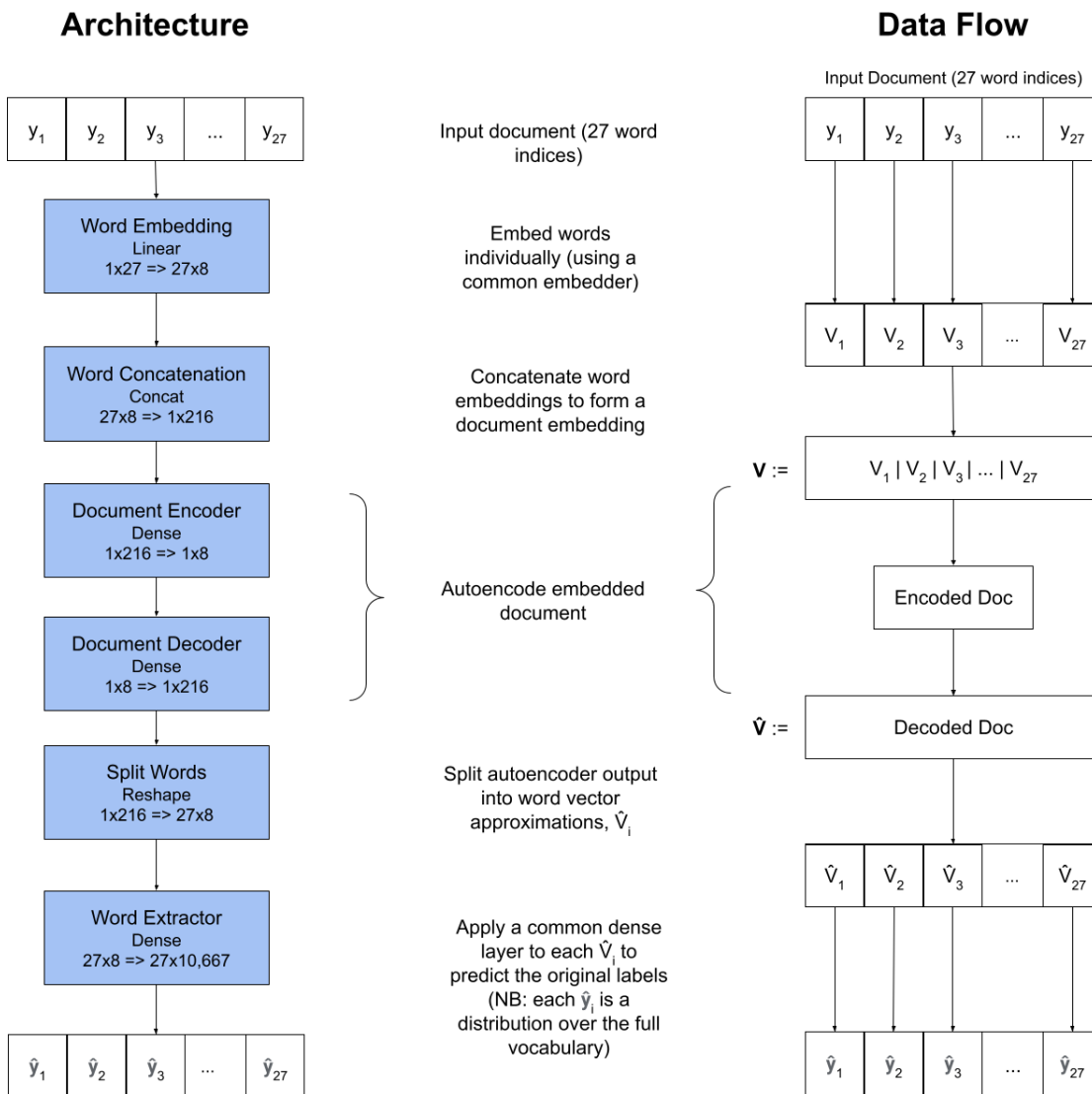


FIGURE 2. Each compute node conducts a training step on a replica of this model, which is described in the text. The total number of trainable parameters is 185,189, of which 85,416 are for vocabulary embeddings. The remaining 100,000 parameters are spread across the encoder, decoder, and word extractor layers, the last of which constituted 96,093. Thus, the model puts a strong emphasis on producing effective vocabulary that are informed by the semantic relationships within the data.

batches based on the inverse weight of the occurrence of terms appearing in the corpus. The intention behind that choice was to ensure that the embeddings of rarely occurring terms were given sufficient opportunity to converge, while not overfitting the overall model to frequently occurring terms. This custom training loop did come at the expense of speed; however, when we attempted to implement the same approach using Tensorflow's distributed MultiWorkerMirrored strategy, the latency associated

with model update sharing made the federated training process prohibitively slow. To circumvent this issue, we reworked the training process, and trained the FL models with `model.fit()` using batch sizes of 512 across each of the compute nodes. While we did notice a significant slowdown in training time (as compared training with `model.fit()` on a single worker), the overall speed was still reasonable.

On a related note, our selection of EC2 hardware was the result of some early experiments, wherein

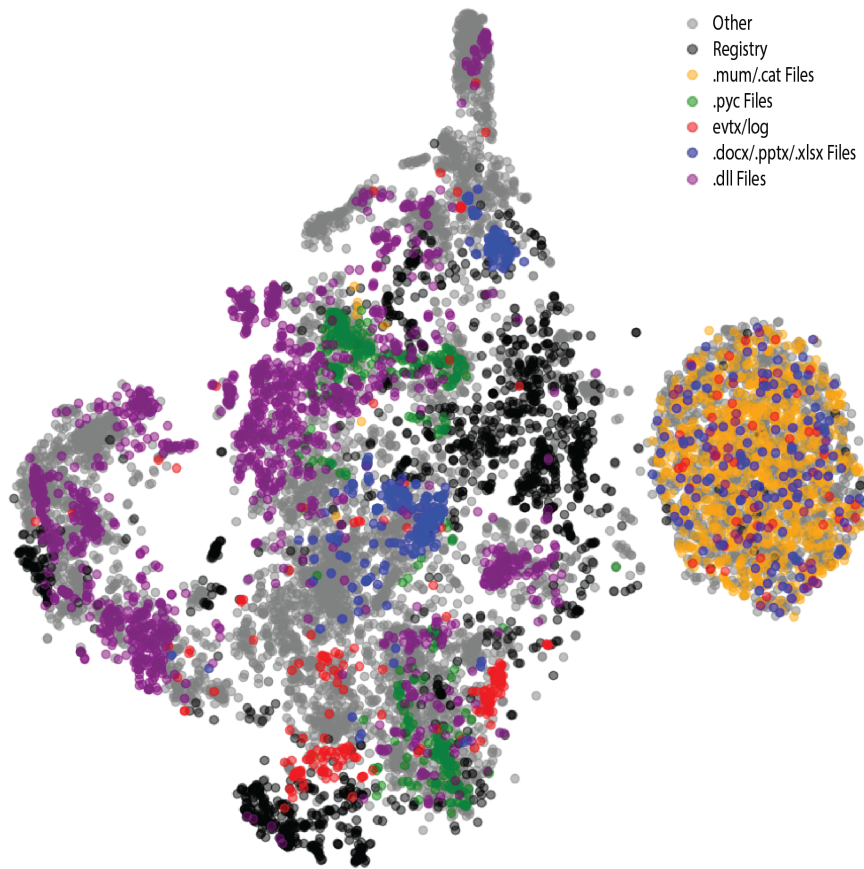


FIGURE 3. These word embeddings show tight clusters around many different types of file types that appear commonly across the dataset (e.g., Microsoft Office files in blue). Note these are a t-distributed stochastic neighbor embedding (TSNE) of the original eight-dimensional vectors down to two dimensions.

we tested this methodology on two g3.xlarge servers and found the training process to be too slow. We expect that this was due to the rather low network performance of these instances (i.e., up to 10 gigabits per second) and opted instead for the g4dn.8xlarge, which provided a guaranteed 50 gigabits per second. The training process here appeared to be much faster, but was still slower than that observed in the nondistributed case.

Results

Our testing methodology reflects our previous work, so that the FL model may be compared to the individual models trained on each of the hosts we examined. This methodology is consistent with using these types of utilities to alert administrators at times when network and host logs are exhibiting anomalous behavior, and to provide prioritized lists of anomalies for investigation/action in an operational

environment. In cybersecurity use cases, overall accuracy is not a useful measure for our performance, as we have a large class imbalance (i.e., less than one percent of our test data is labeled as anomalous). Our goal is not necessarily to correctly classify all of our malicious traffic as anomalous, as some of that traffic may be only incidentally labeled as malicious due to its association with a red team process. Instead, we choose to provide two main results that address the above use cases: a qualitative view of changes in the network's cross-entropy loss over time, and measurements of precision using various error levels as classification thresholds. We feel that these results best promote our goal of allowing administrative users to know when an attack is taking place, while also providing a manageable number of true positives, with a high degree of certainty (i.e., high precision).

In terms of assessing the detection efficacy of the model, we compute an anomaly score s for each record, which as in our previous work is

$s = \sum_{i=1}^{27} CEL(\hat{y}_i, y_i)$, as a measure of how well the model reconstructs the input data. In the ideal, the model should be able to reconstruct benign data with higher fidelity than data associated to red team events.

Overall, the results of these measures indicate that the model is capable of distinguishing benign activities from red team events. Additionally, [figure 3](#) shows the learned embeddings for the vocabulary do result in semantically consistent embeddings. That is, key categories of vocabulary words (e.g., registry keys/values, mum/cat files) and Microsoft office files constitute rough clusters in the embedding space. The specific categories we selected were either constituting a particular file type from the “file path” field (e.g., “.pyc” files or “.dll” files), or types of vocabulary words from different fields that almost always appear together (e.g., registry keys and registry values). While this is merely a heuristic, we can clearly see that along these categories, clusters do emerge among like points. For example, registry keys and values are generally embedded near other registry keys, and values and are generally further away from embeddings of words in other categories. While not perfect, the training paradigm clearly generates meaningful embeddings based on the contexts in which they appear, similar to the nonfederated model featured in our previous work [12].

Results for sysclient 0203-0205

While the model was trained on data from these hosts, the model had no access to the red team activities performed on sysclient 0203 and sysclient 0205 during training. Since 0204 is an unattacked host with its data in the training corpus, we consider this to be our “control” host. Overall, the federated model is able to distinguish between benign activities from red team activities on 0203 and 0205 and generalizes well to the test set on 0204.

As seen in [figure 4](#) (page 34), the anomaly score distributions for benign and red team activities on both attacked hosts are such that there is a clear separation between a large portion of the malicious activity, and the majority of the remaining activity. The scores for sysclient 0204 are similar to the benign scores of the other two hosts.

Emulating deployment of this model in a Security Operations Center (SOC), we opted for a temporally

based anomaly detection system. To that end, we gather our test data into 60-minute buckets, calculate the percentage of records in each bucket over the 99.9th percentile of a random sample of benign data from the given host during the training period and display the resulting time series in [figure 6](#) (page 36). Generally speaking, anomaly scores are within the expected levels outside the attacks on sysclient 0203 and sysclient 0205 and remain at consistently elevated levels during the times each were attacked. There were some exceptions, however. The spikes in our metrics at $t=1500$ and $t=2880$ are correlated with large spikes in the volume of WMI (Windows Management Instrumentation) activity on each host. These spikes are consistent with our previous work and we, also as before, have elected to treat these spikes as nonmalicious anomalies. There are some red team events that appear at $t=1000$, which did not appear in our previous work. This is most likely due to an update in the labeling procedure we employed. That the anomaly scores do not spike around this time is not surprising given the fact that there are only around 100 red team events in total.

For sysclient 0204, anomaly scores appear to be within expectations and in this SOC scenario, no alerts would be generated. Again, this is exactly what one would want to happen, since the red team did not attack this host.

Results for sysclient 0201 and 0202

This is a challenging scenario for the model. Recall that the training data for the model included no records from sysclient 0201 nor sysclient 0202, and were chosen because 0201 contained activities from the red team constituting multiple stages of an attack and 0202 did not.

As we see in [figure 7](#) (page 37), the results for sysclient 0201 are decent, especially given that the training of the FL model had not seen data from this host during training. Here the separation between the anomaly score distributions by class are not as clear, indicating that either there were high volumes of benign activities which were different enough from the training data that they were difficult to reconstruct, or high volumes of the red team activities were too similar to training data. This also means that precision and recall for a naive classifier using the anomaly score as a threshold had significantly degraded performance. However, using the scores

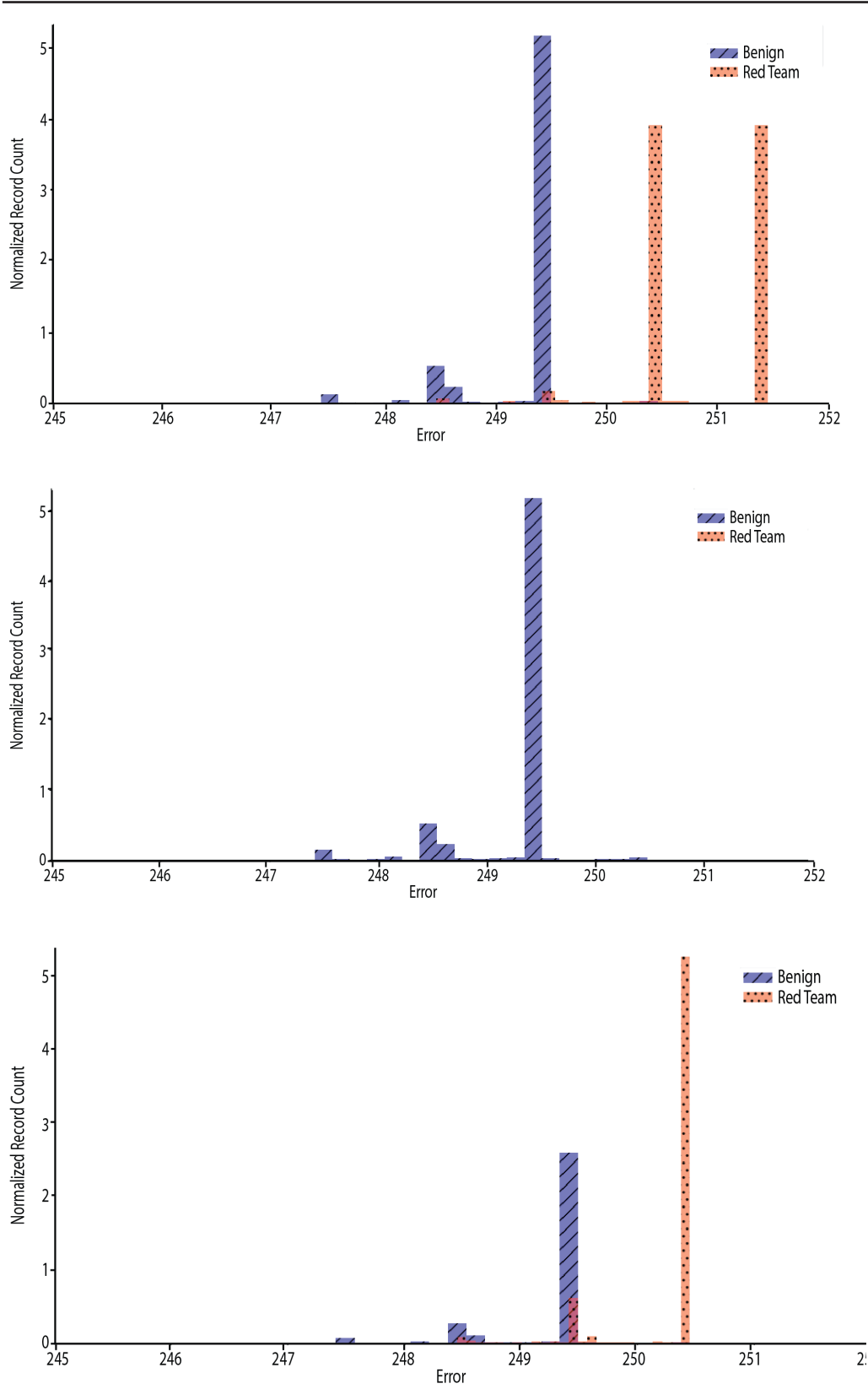


FIGURE 4. These histograms of anomaly score by class (benign and red team) on sysclient 0203 (top) sysclient 0204 (middle) and sysclient 0205 (bottom) show a clear separation between a large portion of the malicious activity, and the majority of the remaining activity. The scores for sysclient 0204 are similar to the benign scores of the other two hosts.

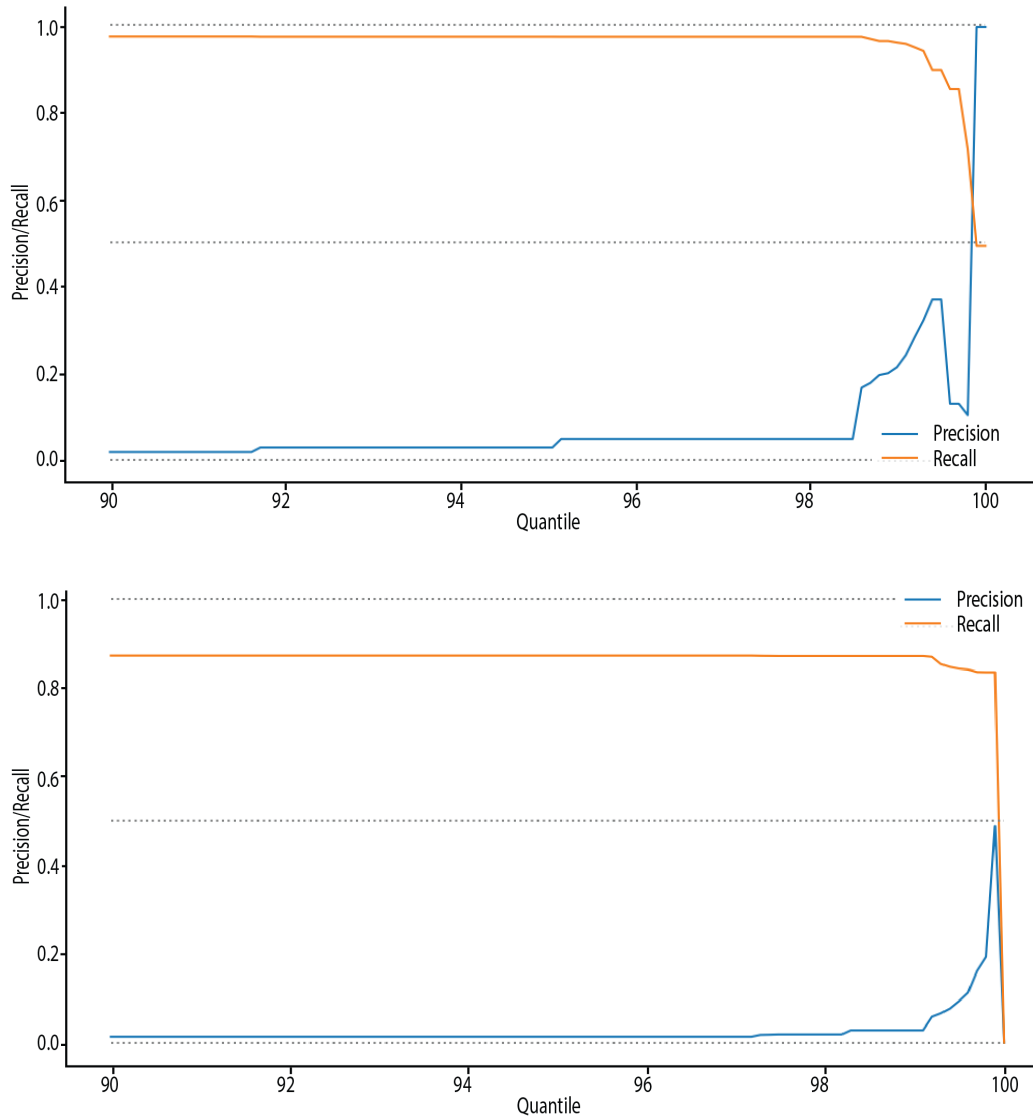


FIGURE 5. For these precision and recall curves for sysclient 0203 (top) and sysclient 0205 (bottom), we used a threshold to create a naïve classifier on the test data. For each quantile, q , between the 95th and 100th percentile of our test loss distribution, we classify every record with a loss above q as being anomalous.

as a temporal anomaly detection scheme, we see that the model does generalize somewhat. Anomaly score spikes early on match the high volumes of red team activities. Spikes around $t=1500$ and $t=2880$ are consistent with sysclients 0203-0205, and lend further credence to the notion that these are benign, but anomalous events.

The results for sysclient 0202 are strong. The anomaly score distribution is very similar to the benign activities of the other hosts. Additionally, these scores remain within expectations in the temporal-based anomaly detection scenario. All told, this very clearly demonstrates that the FL model generalizes well.

Discussion

While we do not perfectly mirror our results from the previous work on sysclient 0201-0204, we do demonstrate that such a FL paradigm has the potential to aid in real-time cyber threat hunting. The results for sysclient 0201 do highlight the need for the training data FL model to be sufficiently close to data in the wild. The degradation in performance as compared to our previous work is most likely due to the model having not seen enough data that is similar to the data that characterizes normal activities on 0201.

Another key component of transitioning such a technique to practice is determining the appropriate

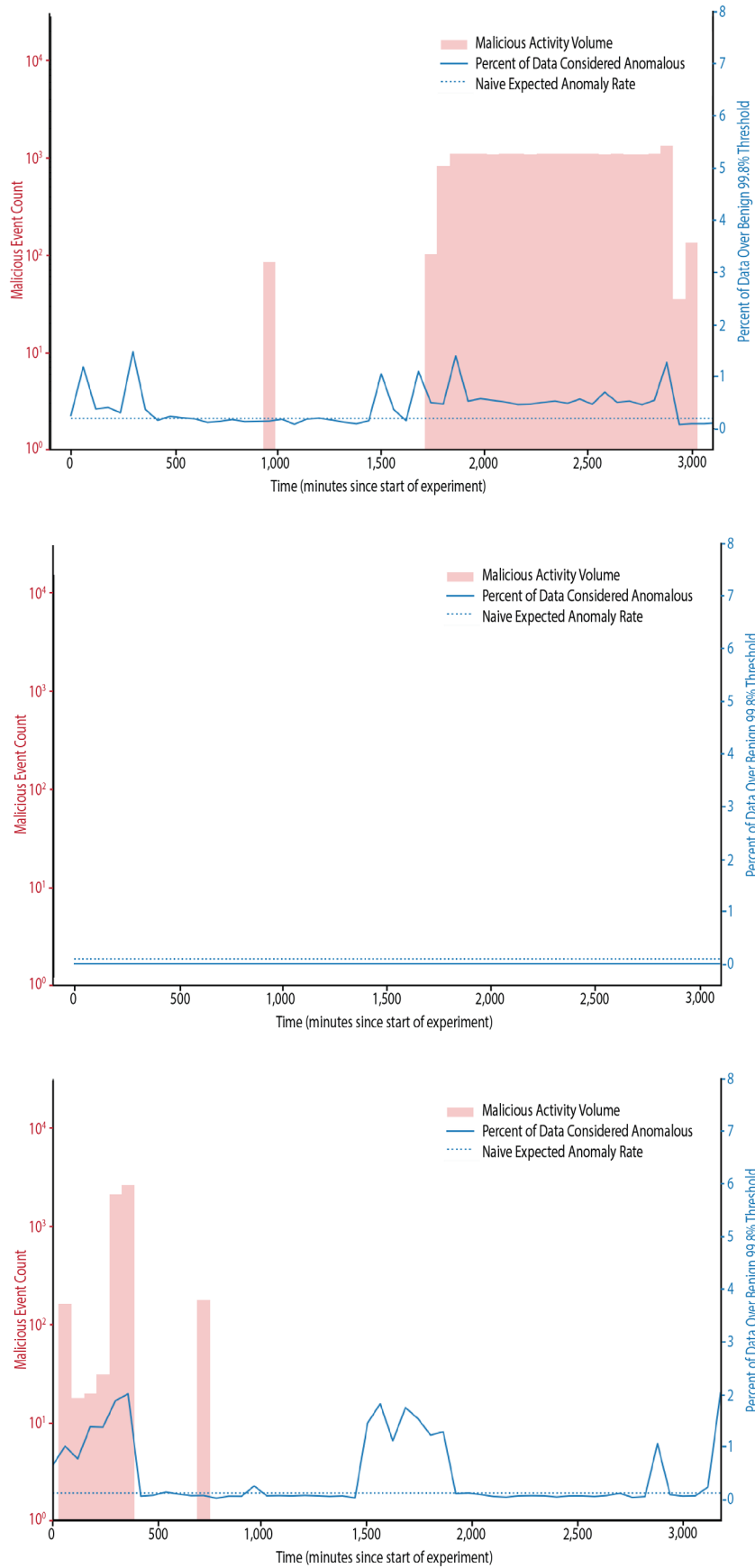


FIGURE 6. In this comparison of the amount of red team activity in a given 60-minute bucket, and the amount of activity over the 99.5th percentile of the error distribution for the training data on the associated host, the dashed line is set at the average expected level of the graph, 0.5 percent. This figure includes two attacked hosts, sysclient 0203 (top) and sysclient 0205 (bottom), as well as the unattacked host, sysclient 0204 (middle).

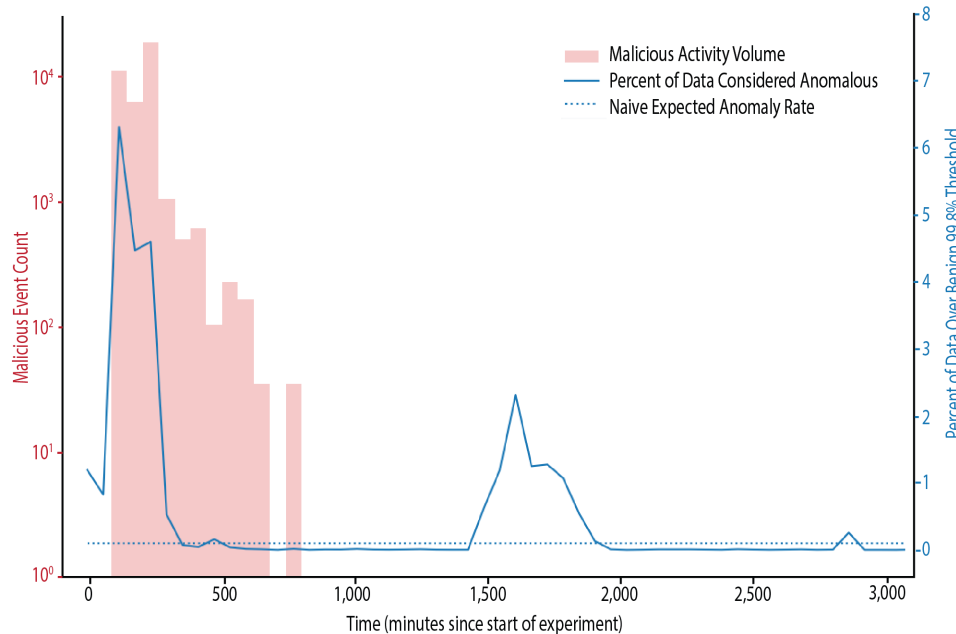
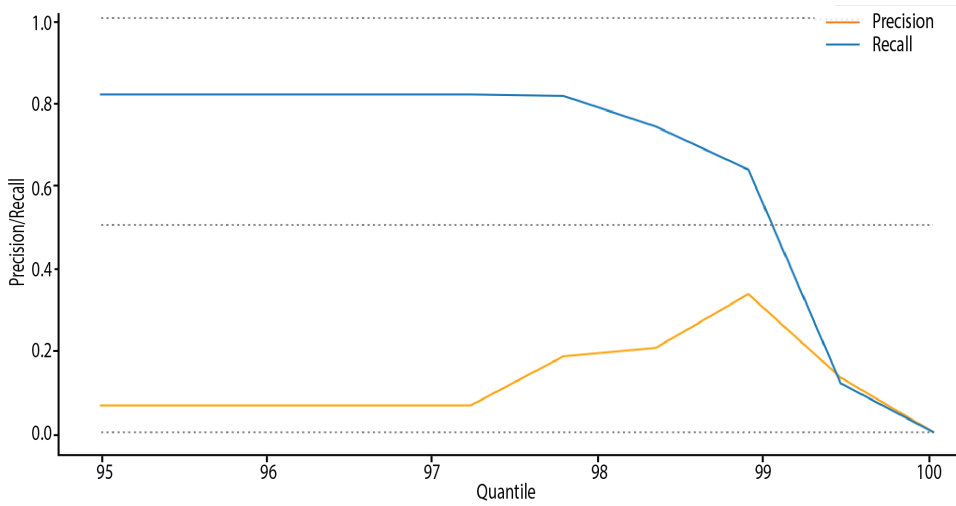
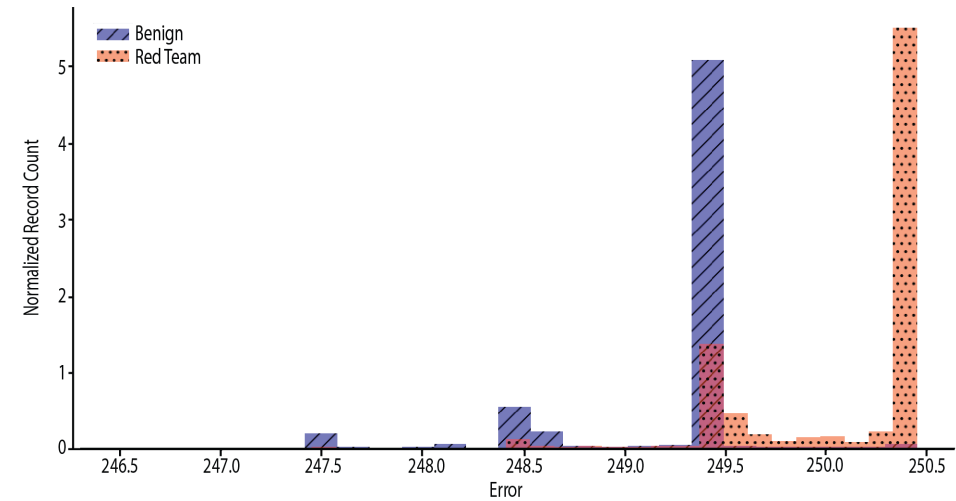


FIGURE 7. In these results for sysclient 0201, the separation between the anomaly score distributions by class are not as clear; however, using the scores as a temporal anomaly detection scheme, we see that the model does generalize somewhat. Anomaly score spikes early on match the high volumes of red team activities.

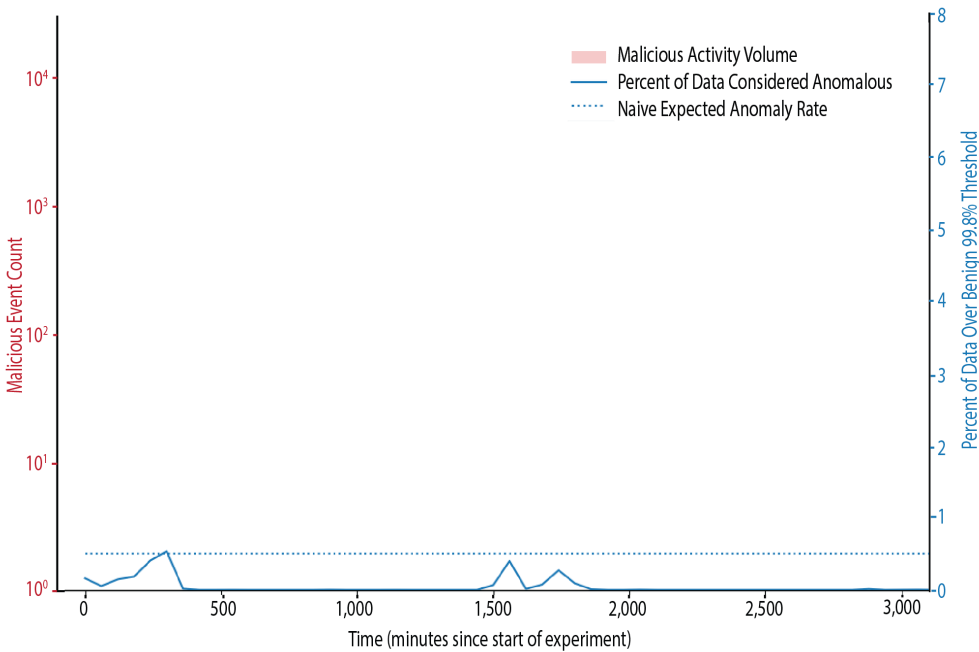
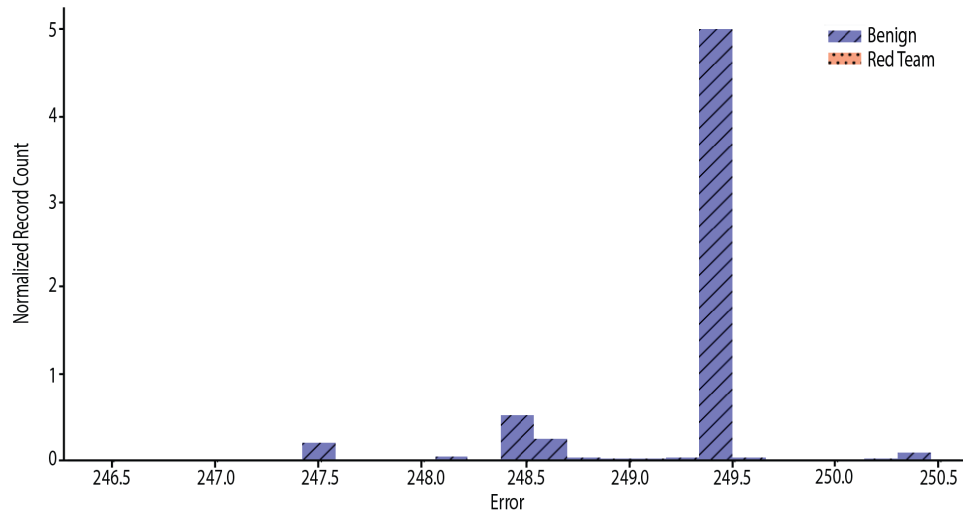


FIGURE 8. Results for syscli-ent 0202 shows the anomaly score distribution is very similar to the benign activities of the other hosts.


hyper parameters. In our experiments, we set many of the hyperparameters to be consistent with our previous work. We do so in order to enable an apples-to-apples comparison of our works. In practice, one should expect the depth of the network, the size of the code layer, and even the dimensions of the vocabulary embeddings to scale with the complexity of the underlying data. In the case of training an FL model on orders of magnitude more hosts than ours, we advise to perform an ablation study.

Future work

The clearest future work would be to attempt to scale this FL paradigm to even more hosts on OpTC. Such an endeavor would be extremely informative for those wishing to employ scalable AI/ML-enhanced cybersecurity analytics in an SOC. Future work also includes a deeper exploration finding suitable hardware and software that can bring the speed of training a FL model on par with a non-FL one (especially

for the scalability concerns). The tensorflow distributed learning module includes support for a strategy that is optimized for tensor processing units (TPUs). Testing out this concept on such architecture would be an interesting endeavor.

As we have alluded, privacy-preserving methodologies, if perfected, could enable cross-organizational model training and collaboration for enhanced

cybersecurity capabilities without being prohibitively risky. Recent literature does demonstrate that differential privacy can be employed in deep learning models, but is very difficult. Research endeavors in this space are needed, in order to ensure that attackers cannot continue to take advantage of the lack of collaboration among defenders across organizations. 

References

- [1] Hutchins E, Cloppert M, Amin R. "Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains." *Leading Issues in Information Warfare & Security Research 1*. 2011. Available at: <https://www.lockheedmartin.com/content/dam/lockheed-martin/rms/documents/cyber/LM-White-Paper-Intel-Driven-Defense.pdf>.
- [2] Strom BE, Applebaum A, Miller DP, Nickels KC, Pennington AG, Thomas CB. "Mitre att&ck: Design and philosophy." 2018 July, revised 2020 March. MITRE Corporation technical report MP180360R1.
- [3] Mossburg E, Fancher JD, Gelinne J. "The hidden costs of an IP breach, cyber theft and the loss of intellectual property." *Deloitte Review 19*. 2016: 106–121.
- [4] Snort webpage. Available at: <https://www.snort.org/>.
- [5] Sigma GitHub webpage. Available at: <https://github.com/SigmaHQ/sigma>.
- [6] Goodfellow I, Bengio Y, Courville A. *Deep Learning*. MIT Press; 2016. ISBN: 9780262035613.
- [7] Sakurada M, Yairi Y. "Anomaly detection using autoencoders with nonlinear dimensionality reduction." In: *Proceedings of the MLSDA 2014 2nd Workshop on Machine Learning for Sensory Data Analysis (MLSDA'14)*; 2014; Association for Computing Machinery, New York, NY: pp. 4–11. Available at: <https://doi.org/10.1145/2689746.2689747>.
- [8] Hawkins S, He H, Williams G, Baxter R. "Outlier detection using replicator neural networks." In: Kambayashi Y, Winiwarter W, Arikawa M, editors. *Data Warehousing and Knowledge Discovery DaWaK*; 2002. Lecture Notes in Computer Science, vol 2454. Springer, Berlin, Heidelberg. Available at: https://doi.org/10.1007/3-540-46145-0_17.
- [9] Wong V, Emanuello J. "Robustness of ML-enhanced IDS to stealthy adversaries." In: *AI/ML for Cybersecurity: Challenges, Solutions, and Novel Ideas at SIAM Data Mining*; 2021. Available at: <https://doi.org/10.48550/arXiv.2104.10742>.
- [10] Ramström K. "Botnet detection on flow data using the reconstruction error from Autoencoders trained on Word2Vec network embeddings." PhD thesis. Uppsala Universitet, 2019.
- [11] Liu F, Wen Y, Zhang D, Jiang X, Xing X, Meng D. 2019. "Log2vec: A heterogeneous graph embedding based approach for detecting cyber threats within enterprise." In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (CCS '19)*; 2019; Association for Computing Machinery, New York, NY: pp. 1777–1794. Available at: <https://doi.org/10.1145/3319535.3363224>.
- [12] Golczynski A, Emanuello JA. "End-To-end anomaly detection for identifying malicious cyber behavior through NLP-based log embeddings." In: *Proceedings of the First International Workshop on Adaptive Cyber Defense*; 2021. ArXiv: abs/2108.12276.
- [13] Kairouz P, McMahan HB, Avent B, Bellet A, Bennis M, Bhagoji AN, Bonawitz K, et al. "Advances and open problems in federated learning." *Foundations and Trends in Machine Learning*. 2021;14(1–2):1–210.
- [14] Ghimire B, Rawat DB. "Recent advances on federated learning for cybersecurity and cybersecurity for federated learning for Internet of Things." *IEEE Internet of Things Journal*. 2022;9(11):8229–8249. doi: 10.1109/JIOT.2022.3150363.
- [15] Zhou Y, Wu J, Wang H, He J. "Adversarial robustness through bias variance decomposition: A new perspective for federated learning." In: *Proceedings of the 31st ACM International Conference on Information & Knowledge Management (CIKM '22)*; 2022; Association for Computing Machinery, New York, NY: pp. 2753–2762. Available at: <https://doi.org/10.1145/3511808.3557232>.
- [16] Boenisch F, Dziedzic A, Schuster R, Shamsabadi AS, Shumailov I, Papernot N. "When the curious abandon honesty: Federated learning is not private." In: *2023 IEEE Eighth European Symposium on Security and Privacy (EuroS&P)*; 2023; Delft, Netherlands: pp. 175–199. doi: 10.1109/EuroSP57164.2023.00020.

[17] DARPA. Operationally Transparent Cyber (OpTC) Data Release, 2019. Available at: <https://github.com/FiveDirections/OpTC-data>.

[18] Anjum MM, Iqbal S, Hamelin B. 2021. "Analyzing the usefulness of the DARPA OpTC dataset in cyber threat detection research." In: *Proceedings of the 26th ACM Symposium on Access Control Models and Technologies (SACMAT '21)*; 2021; Association for Computing Machinery, New York, NY; pp. 27–32. Available at: <https://doi.org/10.1145/3450569.3463573>.

[19] The MITRE Corporation. "Data Model." MITRE Cyber Analytics Repository. Available at: https://car.mitre.org/data_model/.

[20] Denning DE. "An intrusion-detection model." *IEEE Transactions on Software Engineering*. 1987;13(2):222–232. doi: 10.1109/TSE.1987.232894.