ProvIoT: Detecting Stealthy Attacks in IoT through Federated Edge-Cloud Security

Kunal Mukherjee[1], Joshua Wiedemeier[1], Qi Wang[2], Junpei Kamimura[3], John Junghwan Rhee[4], James Wei[1], Zhichun Li[3], Xiao Yu[3], Lu-An Tang[5], Jiaping Gui[6], and Kangkook Jee[1(✉)]

[1] The University of Texas at Dallas, Richardson, TX, USA
{kunal.mukherjee,josh.wiedemeier,james.wei,kangkook.jee}@utdallas.edu
[2] University of Illinois at Urbana-Champaign, Champaign, IL, USA
qiwang11@illinois.edu
[3] Stellar Cyber, San Jose, CA, USA
jkamimura@stellarcyber.ai, zhichun@gmail.com, yuxiaoinf@gmail.com
[4] University of Central Oklahoma, Edmond, OK, USA
jrhee2@uco.edu
[5] NEC Labs America Inc., Princeton, NJ, USA
ltang@nec-labs.com
[6] Shanghai Jiao Tong University, Shanghai, China
jgui@sjtu.edu.cn

Abstract. Internet of Things (IoT) devices have increased drastically in complexity and prevalence within the last decade. Alongside the proliferation of IoT devices and applications, attacks targeting them have gained popularity. Recent large-scale attacks such as Mirai and VPNFilter highlight the lack of comprehensive defenses for IoT devices. Existing security solutions are inadequate against skilled adversaries with sophisticated and stealthy attacks against IoT devices. Powerful provenance-based intrusion detection systems have been successfully deployed in resource-rich servers and desktops to identify advanced stealthy attacks. However, IoT devices lack the memory, storage, and computing resources to directly apply these provenance analysis techniques on the device.

This paper presents ProvIoT, a novel federated edge-cloud security framework that enables on-device `syscall`-level behavioral anomaly detection in IoT devices. ProvIoT applies federated learning techniques to overcome data and privacy limitations while minimizing network overhead. Infrequent on-device training of the local model requires less than 10% CPU overhead; syncing with the global models requires sending and receiving ~2MB over the network. During normal offline operation, ProvIoT periodically incurs less than 10% CPU overhead and less than 65MB memory usage for data summarization and anomaly detection. Our evaluation shows that ProvIoT detects fileless malware and stealthy APT attacks with an average F1 score of 0.97 in heterogeneous real-world IoT applications. ProvIoT is a step towards extending provenance analysis to resource-constrained IoT devices, beginning with well-resourced IoT devices such as the RaspberryPi, Jetson Nano, and Google TPU.

C. Pöpper and L. Batina (Eds.): ACNS 2024, LNCS 14585, pp. 241–268, 2024.
https://doi.org/10.1007/978-3-031-54776-8_10

Keywords: Provenance graph analysis · anomaly detection · dynamic malware analysis · federated learning · deep learning · privacy

1 Introduction

The Internet of Things (IoT) revolution established a radical new computing paradigm that traditional security protocols have failed to comprehensively cover. With the recent development of small and powerful devices [8,39,65], increased network connectivity [70] has allowed IoT devices, including wearables, drones, and autonomous vehicles, to be deployed at an unprecedented scale [31]. These IoT devices are not only independently security critical [1,78], but they are also entry points into a network to perform data theft, surveillance, and denial-of-service [21,22,83] attacks. As IoT technology's attack surface increases, so will the prevalence of attacks targeting IoT devices.

Traditional stealthy attack techniques are quickly being adapted to threaten IoT devices and cyber-physical systems [15,42]. Various security solutions for IoT devices have been proposed to defend against these attacks, but they are limited in their capability to defend against skilled adversaries [60,71]. Beyond the legacy approach to defense, several provenance-based security approaches [14,41,43,82] have been proposed to protect conventional IT infrastructure (*e.g.,* server and desktop computers) against sophisticated malicious actors. Instead of a naïve dependence on static signatures, provenance-based solutions analyze the runtime behavior of known programs to detect anomalies.

Provenance-based defenses provide a promising approach for IoT devices as well. These defenses first capture the benign behavior of the program by aggregating auditing data to show causal relations between system events. These causal events are represented as a provenance graph, which is then vectorized so that machine-learning (ML) techniques can model the typical (*i.e.,* benign) behavior of the program. Provenance graphs are rising in popularity alongside advances in graph-based learning approaches [29]. However, the overhead incurred by graph techniques that digest an entire provenance graph is unacceptably high for most IoT devices.

Since most IoT devices run on limited resources, system components such as the CPU, memory, and storage are engineered to serve a single dedicated task, leaving scarce resources for security. Network bandwidth is likewise constrained in mobile IoT situations. These limitations severely hinder the data processing efforts required to support graph-based ML security solutions for IoT systems. Since the quality of the data deteriorates, the detection accuracy of ML models using the data also deteriorates. Additionally, the often scattered nature of IoT deployment makes the task of consistent and stable data collection challenging.

To address these issues, we propose ProvIoT, a novel federated edge-cloud collaborative security architecture for IoT that extends the detection capability of IoT security against sophisticated and stealthy attacks. ProvIoT aims to provide a system-wide behavioral graph analysis framework for the IoT domain. To overcome the IoT specific data collection and privacy constraints, ProvIoT uses

a federated edge-cloud collaborative framework with two major components: *(1)* A *Local Brain* for system event collection, summarization, model training and anomaly detection in an edge device, and *(2)* a *Cloud Brain* for performing federated averaging [55] on these local models to produce a global model and orchestrating the distribution of the global model.

Our work extends a path-based graph summarization approach for system provenance analysis [41,43,82] that reduces computational overhead by extracting subgraphs (*i.e.,* causal paths) from the whole provenance graph. We design a novel federated anomaly detection framework using Local Brains and a Cloud Brain in the context of IoT. Our prototype runs Local Brains on multiple IoT platforms — ARM-based IoT, edge GPU devices [39,65], and x86-based Linux hosts. For a long term evaluation, we deployed Local Brains to 33 devices and collected low-level system events over twelve months.

The Cloud Brain established global behavioral models for the twenty commonly installed programs listed in Table 3 and the five major IoT use cases listed in Table 2. We evaluated ProvIoT against real-world IoT malware designed to impersonate long-running and trusted software which included both natively fileless malware and malware [20] that uses a fileless wrapper [36]. We also used realistic testbeds to reproduce prominent attacker tactics, techniques, and procedures (TTPs) that comprise the essential components of advanced persistent threat (APT) campaigns following the MITRE ATT&CK framework. Our evaluation results in Sect. 7 show that ProvIoT efficiently constructs behavioral models that can accurately detect stealthy attacks, including fileless IoT malware and APT-style attack campaigns.

In summary, our work brings in the following contributions:

- To the best of our knowledge, ProvIoT is the first proposed provenance based security detection approach in the context of IoT that counters stealthy attacks using federated learning and on-device detection.
- ProvIoT provides a new design choice for federated edge-cloud collaborative security learning by streamlining computationally expensive graph-based behavioral security in the IoT context.
- We extensively evaluated the efficiency and effectiveness of ProvIoT with realistic deployments. Adversarial cases are carefully designed using realistic attack cases and fileless malware samples.
- We will publish the complete IoT provenance dataset and tools required for our data analysis pipeline [61] as open artifacts.

2 Background

In this section, we first introduce fileless attacks, their operations, and their application to the IoT domain. We then provide insights for using in-host system provenance graphs to build behavioral models in IoT devices.

2.1 Fileless Attacks on IoT Devices

In this paper, *fileless attack* refers to a group of attack techniques with no footprint in the file system. Alternative terms used in the field include "zero-footprint", or "living off the land" [26].

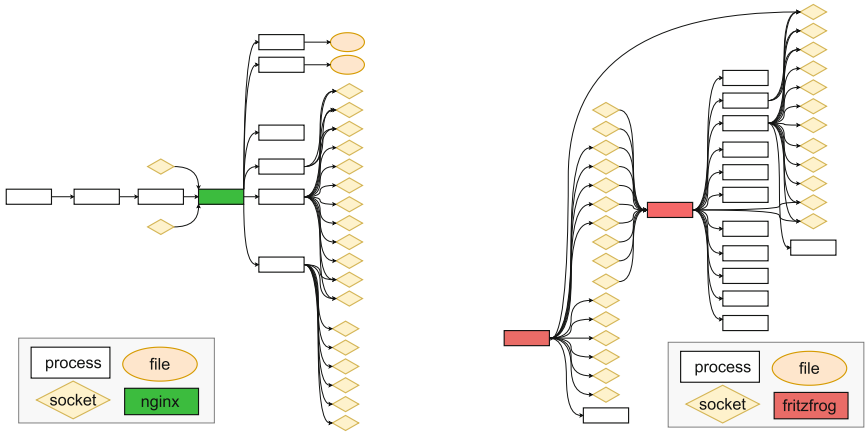
Fileless attacks are characterized by the impersonation of trusted off-the-shelf applications and pre-installed system utilities. Since many of these trusted applications are commonly used by users and system administrators, it is harder for defenses to block access to them to prevent such attacks completely. Such impersonation techniques have seen rising popularity in recent cyberattacks [26,51]. Instead of storing the malware payload directly onto the disk before executing it, this malware uses the strategy of "living off the land" by injecting it into benign running processes (*e.g.,* trusted applications) and avoiding detection by executing only in process memory. During runtime, the malware may also rename itself to a seemingly benign process name using a `prctl(PR_SET_NAME)` call. These impersonation approaches have diverged and evolved in multiple ways in IoT systems [33,34]. Some possible impersonation approaches are highlighted below.

Process Injection. `ptrace()` is a system API used to support code injection to another process for development purposes. However, attackers have abused `ptrace()` to inject malicious code into the memory of legitimate processes [13].

In-Memory Execution. The `memfd_create()` system API family creates an anonymous file in memory-mounted file systems. Using `memfd_create()`, an attacker can directly load malware from the memory space without writing a payload to the filesystem. This attack enhances the traditional attack strategy of storing malware in transient storage (*e.g.,* `/tmp`, `/var/run`, `/dev/shm`). With `memfd_create()`, the malware further reduces its footprint, preventing users from locating it with standard filesystem access even during runtime. Multiple loader frameworks [36] exist that are able to encode regular file-based malware into different fileless variants.

Case Study: FritzFrog. In January 2020, a security group discovered and reported FritzFrog [42], a sophisticated peer-to-peer (P2P) malware botnet. FritzFrog is a crypto mining worm that breaks into and spreads through SSH servers. Written in Golang to natively target different architectures, FritzFrog uses fileless techniques to leave no traces on the filesystems of the infected devices. We specifically consider FritzFrog in the context of IoT devices.

FritzFrog performs file operations in memory to impersonate a regular benign system process's identity. After the initial break-in, FritzFrog masquerades as the `nginx` web server or the `ifconfig` process. The infected IoT device connects to a command and control (C&C) server via encrypted sessions to seemingly benign beacons. Then, the malware infects other IoT devices to mine cryptocurrencies by exploiting a weakness in SSH services. Figure 1 compares the behavior of the original `nginx` process and that of FritzFrog impersonating `nginx`. Although FritzFrog leaves no filesystem footprint, provenance-based intrusion detection



(a) Benign `nginx` (b) Impersonated `nginx`

Fig. 1. FritzFrog malware impersonating `nginx` web-server.

systems can detect and defend against it as the behavior of benign `nginx` and FritzFrog are distinct.

2.2 System Provenance and Graph Learning

ProvIoT extends system provenance analysis [53], originally proposed by King et al. [45], to implement on-device edge IoT behavior monitoring system. System provenance operates through the installation of a data collection agent on each host to collect `syscall` level system events. These events are then sent to an in-memory database to build a causality graph by associating data and control dependencies between processes, files, and network resources. The events that system provenance collects are as follows: (1) process events, such as process create and destroy; (2) file events, including file read, write, and execute; and (3) network events, including socket create, destroy, read, and write.

With the increased deployment of provenance-based security solutions in the last decade [35], the output of system provenance, the *provenance graph*, forms the foundation for graph-based learning and detection approaches. In this regard, provenance graphs best represent the runtime characteristics of system entities and have quickly become an essential source of input to model a program's runtime behavior. Along with recent developments in graph-based learning approaches [46,67], research on behavioral modeling and its application for anomaly detection has gained considerable momentum [43,82].

While Graph Neural Network (GNN)-based learning analysis techniques have exploded in popularity, they often struggle to digest provenance graphs, which are large and extremely dense. Typically, system provenance graphs contain many nodes and edges that store the different labels and detailed attributes system events. For instance, our graph dataset produced nodes and edges with

an average of 10 ~ 15 attributes for node and edge types. In our attempt to evaluate a GNN-based framework on our data, we encountered many limitations with the open-sourced framework [76,84] especially in regard to the processing power needed to process provenance graphs.

Works such as [43,82] have addressed the challenge of data processing overhead in general Neural Network (NN) approaches by implementing efficient path selection to build behavior models for detecting anomalous deviations. To adapt this design for use in IoT devices, we collect system-level events on the IoT devices and summarize them using the path selection approach. Recent advances in IoT machine learning frameworks have also made significant strides in executing sophisticated neural architectures in low-resource IoT environments [10,52]. The Local Brain locally trains a model on local data and shares only the model weights with the Cloud Brain. The Cloud Brain aggregates the model weights in a federated way [27] to build behavioral profiles that integrate global perspectives across multiple devices while preserving each device's privacy.

3 Threat Model

Our threat model assumes that the data collection and summarization pipeline on the IoT device is trusted *i.e.,* the integrity of the provenance records are guaranteed by existing secure provenance systems [41,43,62,82]. This assumption is consistent with existing provenance research that requires end-host data collection and reporting [43,53,82]. Securing and verifying the trustworthiness of the end-host data reporting is an important research topic that is orthogonal to our research [19]. Procedural dataset poisoning is outside the scope of our work. We consider the use of distributed consensus protocols [49] or attestation approaches that extend the root of trust with hardware level support [79].

Attacks targeting the IoT platform, communication infrastructure [18], or the analysis process running in the cloud are outside the scope of this paper. We further assume that the reporting agents are honest and restrict our target IoT devices to those with at least 375MB of RAM [60,71] to support provenance summarization. Many modern commodity IoT devices (*e.g.,* smart thermostats [3], smart watches [2], smart fridges [16], smart doorbells [11], and smart home devices [5]) are equipped with 512MB or more of RAM.

ProvIoT attempts to detect malicious behavior in IoT systems by learning the distribution of expected benign behaviors and reporting significant deviations from that expectation. We primarily consider APT scenarios [62] and fileless malware [42,58,80] that impersonates one or more of a set of whitelisted programs to evade traditional IDS [41] mechanisms.

4 System Overview

Figure 2 presents the architecture of ProvIoT, that is composed of two collaborating subsystems: *Local Brains* and a *Cloud Brain*. Each Local Brain gathers host-level monitoring data from the IoT device into an in-memory database. It

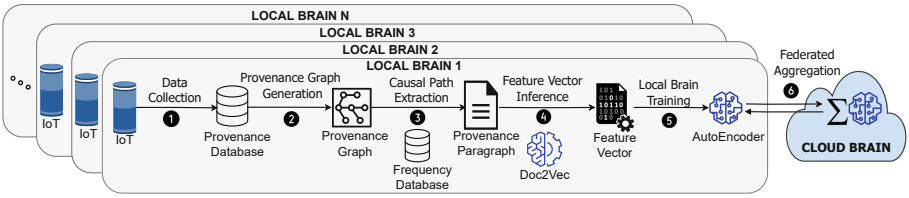


Fig. 2. The federated training pipeline of ProvIoT.

then summarizes the data and converts it to neural embeddings for ML model training. Data summarization only incurs 10% CPU usage and 65MB of RAM overhead. We can set the relevant local events and model training to run infrequently during low-load periods. After the local training, the Local Brain sends the updated neural weights to the Cloud Brain.

The Cloud Brain uses federated averaging [55] to combine the weights received from the Local Brains into a global model, which is sent back to each Local Brain for use in detection. The Local Brain can then perform detection directly on the IoT device using the federated global model. Periodically, the Local Brain will synchronize with the Cloud Brain, pushing up its local weights and fetching the updated global model. The *only* communication that the Local Brains have with the Cloud Brain is the communication of model weights during training. The Local Brains are fully capable of defending the IoT devices even when disconnected from the network.

4.1 Local Brain

We deploy a Local Brain to each IoT device to collect host-level monitoring data including process creations, file operations and network socket interactions. The Local Brain's training has the following major steps: *(1)* data collection, *(2)* provenance graph generation, *(3)* causal path extraction, *(4)* feature vector inference and *(5)* model training.

The first step in doing provenance analysis in IoT is data collection ❶, where we collect system monitoring data and create system event records. Similar to [41,43,53,62,82], we collect monitoring data for the following types of system entities: processes, files, and Unix domain sockets. Each entity type is associated with a set of attributes. For example, the attributes of a process are its creation time, command used to invoke, executable path and other relevant information. We use these entities and the interactions between them (*e.g.,* creation, reading, writing) to represent the system behaviors of the IoT device.

The collected data consists of raw syscall sequences which are translated into meaningful system information (*e.g.,* file descriptors are translated into file paths and PIDs are translated into process names) and stored in the provenance database. After translation, the data collection module processes the information into system events, which embodies the interaction between two system entities. Formally, we define a system event as $e_R(n_s, n_d, t)$ where n_s is the source entity,

n_d is the destination entity, t is the time when e occurs, and R is the relationship (*e.g.,* read, write, create). For example, `Process A opens (with write permission) File B` at time T is $e_w(A, B, T)$.

System events are queried from an in-memory database to generate ❷ the provenance graphs, $G(p)$, for a particular program. The generated provenance graphs are decomposed into subgraphs (*i.e.,* provenance paths). Formally, we define a causal path λ in a provenance graph $G(p)$ as an ordered sequence of system events (or edges) $\{e_1, e_2, \ldots, e_n\}$ in $G(p)$, where $\forall e_i, e_{i+1} \in \lambda$, $e_i.dst == e_{i+1}.src$ and $e_i.time < e_{i+1}.time$. The time constraint enforces that an event can only be dependent on events in the past, which prevents infinite loops.

After causal paths are extracted from provenance graphs, the relevant causal paths are extracted ❸ using a frequency database. Relevant causal paths during training are the common causal paths since we want to train the behavioral model with common provenance paths, but during anomaly detection relevant causal paths are the rare, since we want to detect these rare behaviors.

The frequency database stores historical behavior information for a particular program and is used during the ranking process, including how many times the system has seen a particular system event in the past. For example, if an entry in the frequency database is `</bin/bash|CREATE|/bin/cat, [1000]>`, it means in the past `/bin/bash` created `/bin/cat` one thousand times. False positives due to benign program evolution is an important issue for ML-based detectors. Therefore, ProvIoT updates the frequency database at run-time using benign behavior to capture the evolution of program behavior.

The relevant causal paths are converted ❹ to feature vectors using *doc2vec* [50]. The local model is then trained ❺ on the feature vectors, and the model weights are sent ❻ to the Cloud Brain to update the global model and propagate the localized information to the other connected Local Brain instances. After the Local Brain receives the aggregated global model weights, it starts the anomaly identification process. The Local Brain model uses the new model weights to detect anomalous behavior and raises an alert if any anomalous events are found. The pipeline is visualized in Fig. 3 and explained in Sect. 5.

Since the only connection with the Cloud Brain host is for sending and receiving model weights, the network overhead is constant and independent of the amount of data processed on each IoT device. Additionally, since the global models are stored on the device itself, the Local Brain can still operate even if the network connectivity is lost. This gives ProvIoT an advantage over other IoT behavioral anomaly detectors [32,69] as it does not require the transmission of the data to a centralized server for detection to occur. This also preserves the privacy of the device. We describe the detection models in more detail in Sect. 5.

4.2 Cloud Brain

Since the Cloud Brain resides in the cloud, it has sufficient computing power to aggregate ❻ the model updates from multiple Local Brain instances to build the global detection models and to synchronize the aggregated global weights with the Local Brain instances. This architecture scales more efficiently than

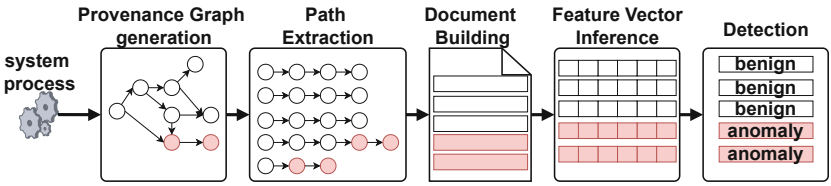


Fig. 3. The detection pipeline of the Local Brain.

centralized off-device detection schemes because federated averaging is infrequent and is less intensive than performing anomaly detection for an entire fleet of IoT devices, so expanding the fleet does not dramatically increase the computational requirements of the Cloud Brain.

Federated Aggregation. Device specific anomaly detection models are aggregates them using the `FederatedAveraging` algorithm described in [55]. Because each device gathers data only from the information it encounters, the data from a single device represents a slice of all the potential benign behaviors. The aggregation that takes place in the Cloud Brain improves the detection accuracy by combining the different pieces of information from all the connected clients.

5 Federated Detection

A core component of ProvIoT is its ability to perform detection autonomously on the IoT device without a centralized server. The local detection module raises alerts when suspicious events occur.

While a centralized server is used to keep the detection module up to date, it is not necessary for detection. The detection pipeline in the Local Brain use the same data collection and preprocessing steps as the training pipeline, but selects rare paths for detection instead of common paths for training.

The detection pipeline, shown in Fig. 3, works in the following manner: first, the Local Brain will generate provenance graphs for each target program and extract rare causal paths for consideration. These causal paths are converted into causal sentences [82], which are combined to form a causal document. Next, we use an NLP model, *doc2vec* [50], to embed the causal document as set of k-dimensional feature vectors. Finally, we use the trained *autoencoder* [40] model to detect the malicious causal paths as done by recent studies [41,62]. The intuition is that when feature vectors are inferred using the *doc2vec* model, benign causal paths will generate feature vectors that would be clustered separately from anomalous feature vectors.

It is possible that there is no anomaly in a process, but a combination of processes can lead to the anomaly, even still ProvIoT would be able to identify these anomalies. Since, during the graph building phase we capture both the forward dependencies (*e.g.,* creating new interactions with different system artifacts or modifying system artifacts) and backward dependencies (*e.g.,* capturing the malware payload deployment event that started the attack as well

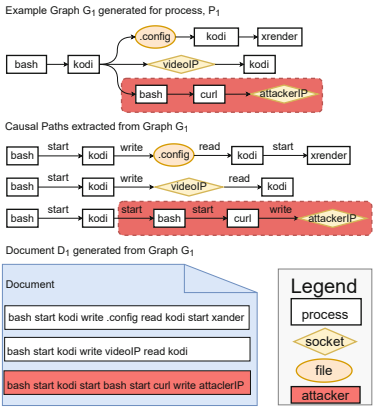


Fig. 4. Example causal paths extracted from a provenance graph, G_1, generated for process, P_1. Using the extracted causal paths the sentences are formed for a document, D_1.

as different program and data dependencies), we obtain a holistic system snapshot. Because malicious activities contain previously unseen behavior, their corresponding causal paragraphs will contain rare sentences, which will be inspected during the detection process.

5.1 Graph Building and Path Selection

For each target program, the Local Brain will generate provenance graphs from system events gathered in the data collection module. Causal paths are extracted from the provenance graphs through a series of random walks. We consider the rarest 15 % of the causal paths using [43]; 15 % was empirically determined in our training phase. Following [43,62,82], the rarity of a causal path is calculated using the frequency database introduced in Sect. 4.1. The regularity of an event is $R(e = (u, v, r)) = \frac{|Freq(u,v,r)|}{|Freq(u,*,r)|}$, and the regularity of a causal path is $R(P = (e_1, e_2, \ldots, e_n)) = \Pi_{e \in P} R(e) \cdot \alpha$, where α is a correction factor to prevent the regularity of long paths from trending towards zero. The rarity of a path is simply the complement of its regularity, $1 - R(P)$.

The information embedded in the provenance graph needs to be extracted to be used as features. One naïve approach may be to use the whole provenance graph for detection. However, using the entire graph will result in a lot of benign noise (events) being mixed into the overall data and the overhead needed to digest the entire graph for ML purposes are unreasonable in an IoT context. Many stealthy malware writers use this property to attempt to blend in with the surrounding benign noise in the graph. Thus, we use a frequency database, as defined in [43] to extract *rare* causal paths from the whole provenance graph. An example of causal paths extracted from a provenance graph in Fig. 4.

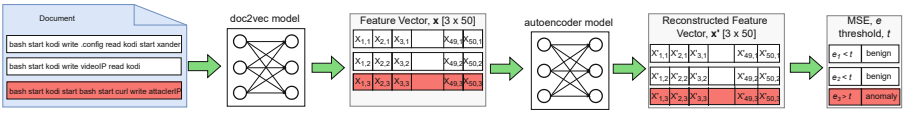


Fig. 5. Example detection workload for graph G_1 in Fig. 4. After the document D_1 is formed, the causal sentences in the document are converted into feature vectors (fv) using doc2vec model. Then the fv are fed into the autoencoder to get the reconstructed fv. Sentences are flagged as anomalous if the mean squared error between the original fv and the reconstructed fv is above a threshold determined during training.

For each selected path, ProvIoT removes the host/entity-specific features, such as host name and identifier, from each node and edge. This process ensures that the extracted representation is general for the subsequent learning tasks.

5.2 Document Embedding Model

The extracted causal paths need to be vectorized before they can be processed by the local detection model. As illustrated in Fig. 4, we first translate the causal paths into causal sentences, a process detailed in [82]. These causal sentences collectively form a document. Following recent methodologies [62,82], we employ the *doc2vec* Natural Language Processing (NLP) model [50] to transform these causal sentences into their corresponding feature vectors, as depicted in Fig. 5. Our *doc2vec* model, trained using data from benign deployments, ensures that causal sentences common in benign contexts yield neural embeddings that are more similar to each other compared to embeddings from rare causal sentences.

5.3 Federated Autoencoder

In ProvIoT, each Local Brain trains autoencoder models on the feature vectors from 5.2 and shares the model weights with the Cloud Brain for aggregation using federated averaging [55]. After fetching the global autoencoder models from the Cloud Brain, the Local Brain is ready to independently detect anomalies.

The Cloud Brain is distinct from the central server in the current state-of-the-art (SOTA) provenance system for IoT [32,69], which collects all the device data over the network and performs anomaly detection serverside. ProvIoT's on-device detection approach affords several advantages: *(1)* sending only the model weights over the network both reduces network overhead and preserves the privacy of activities on the IoT device; *(2)* on-device detection allows the IoT device to remain protected even when disconnected from the network; and *(3)* distributing the detection workload to the edge devices allows ProvIoT to scale horizontally with the size of the IoT device fleet, rather than requiring a vertically scaling central server.

The Local Brain's autoencoder models follow a typical structure for anomaly detection. The autoencoder has an encoder, which maps the benign feature vectors to a latent space representation that captures behavioral patterns, and a

decoder, which reconstructs the original input. To detect anomalies, we measure the Mean Squared Error (MSE) of the reconstructed input and the original input; the input is flagged as anomalous if the MSE is higher than an experimentally determined threshold, which for our implementation was the 99th percentile. The intuition behind this detection scheme is that the autoencoder can effectively reconstruct benign samples similar to the ones it was trained on, but should struggle to reconstruct samples that are substantially different (*i.e.,* anomalies).

6 Implementation

Our ProvIoT prototype was written in C++, Java, and Python. The system level data collection agent was written in C++ with the provenance graph generator and path selection module implemented in Java. The document embedding and ML model were implemented in Python. The Local Brain's data pipeline modules communicate using the Unix domain socket.

System Level Data Collection. In a Local Brain, our prototype's data collection module uses the Linux audit framework to collect a subset of system calls relevant to our interested system entities (*i.e.,* files, processes, and network sockets), which include system calls for *(1)* file operations (*e.g.,* `read()`, `write()`, `unlink()`), *(2)* network socket operations (*e.g.,* `connect()`, `accept()`), *(3)* process operations (*e.g.,* `fork()`, `exec()`, `exit()`). We used SQLite as an in-memory database where system level data are stored. The in-memory database is computationally lightweight and executes queries quickly. Therefore, our provenance graph generation can be done without putting too much strain on the IoT device's resources. The primary workload of the IoT device is taken into consideration as well as the limited onboard resources, such that the Local Brain will pause data collection and subsequent processes (*e.g.,* graph generation, path extraction, training and detection) if the resource usage exceeds a present threshold, set at 30% CPU time or 1024MB memory by default.

Data Processing and Summarization. We use the NLP *doc2vec* model in the Gensim Library [9] for document embedding. The Keras library with Tensorflow [77] backend was used to implement the *autoencoder* model. The autocoder model has four fully connected layers with 50, 10, 10, and 50 neurons respectively. The first two layers are used for encoding, and the last two are used for decoding. The Adam optimizer with L1 regularization is used to prevent overfitting.

7 Evaluation

In this section, we evaluate ProvIoT's efficacy in detecting stealthy attacks in IoT devices. To this end, we seek answers for the following three research questions (RQs):

RQ1: Detection Accuracy. How effective is ProvIoT at detecting stealthy attacks (*e.g.,* fileless IoT malware impersonating trusted system programs) and APT campaigns? (Sects. 7.3, 7.4)

RQ2: Benefit of Federated Architecture. What benefits does the collaborative architecture have over a centralized approach? (Sect. 7.5)

RQ3: Resource Efficiency. What CPU and memory overhead does ProvIoT incur? (Sect. 7.6)

7.1 Dataset

In this section, we introduce the provenance datasets that consist of provenance graphs generated by capturing the benign and malicious IoT system's behavior.

Dataset Components. Our datasets consist of three major components: forward graphs, backward graphs and causal paths. The forward graphs consist of all the system events that are caused by the process associated with a Point of Interest (POI) event, *e.g.,* process creation, file and socket reads/writes. The backward graphs consist of the system events that created the POI event. We merge the forward and the backward graphs to get a unified graph that captures all the system events associated with the POI event. We then extract causal paths from this unified graph; the size statistics for the graphs and causal paths are shown in Table 3 in the appendix. To generate a graph dataset for a given program, we use all process creations for the given program name as POI events to build forward and backward graphs.

Benign Dataset. We consulted our university's Institutional Review Board (IRB) to develop an ethical experimental protocol for selecting volunteers for benign data collection. Once the volunteers were chosen, they received information about how their data would be used and securely stored to ensure confidentiality. The benign data collection took place over a period of twelve months, from January 2021 to December 2021, and resulted in the collection of over 30 TB of data. The benign profile for the programs was constructed by gathering system events from a diverse set of 33 devices, including ARM-based IoT devices such as Raspberry Pi, Google TPU, and NVIDIA Jetson Nano boards [8,39,65]. The device platforms consist of 1 Google TPU, 1 NVIDIA Jetson Nano, 3 Raspberry Pi 4, 5 Raspberry Pi 3B+, 5 desktops, 5 laptops, and 13 servers. Importantly, the provenance graphs that capture the behavior of a given system program exhibit a relative consistency across different IoT devices and platforms.

The IoT devices in our benign testbeds performed various IoT tasks and common system operations categorized as *IoT Applications* and *System Programs* respectively in Table 3. Using this system event data, we generated provenance graphs for popular IoT applications [25] and common system programs [33,34] that are frequently targeted for impersonation. We chose 1000 benign process instances for each of the 20 programs and 150 instances for each of the 5 IoT applications to create the benign dataset. The provenance graphs generated from

the benign IoT applications consisted of 237,923.84 causal paths, 1,046.97 vertices, and 1,534.66 edges (IoT Application in Table 3) on average. Similarly, the provenance graph generated from the Linux system processes had an average of 168,652.11 causal paths, 332.49 vertices, and 398.48 edges (System Program in Table 3). For readers interested in further details about the statistics of the benign dataset and how it was generated, please refer to Sect. A.

Malicious Dataset. We created two isolated testbeds to run the malicious workloads. Firstly, we launched publicly known IoT malware using a fileless wrapper [36] to impersonate the identities of the popular IoT applications in Table 2. Second, we conducted a typical APT scenario by carefully coordinated the APT attack vector with the MITRE ATT&CK [59] framework to comprise the end-to-end attack [59] campaign. We launched a stealthy attack campaign that contains five kill-chain [12] stages (Table 5) — *(S1) gain access* by injecting a malicious payload into an active benign process; *(S2) establish a foothold* by communicating back to a C&C server over HTTPS (port 443); *(S3) deepen access* using a privilege escalation exploit [57], *(S4) move laterally* by scanning the local network for vulnerable hosts with open ports; and *(S5) look, learn, and remain* by exfiltrating sensitive user data to the C&C server. Each attack stage was conducted by different attack TTPs using Metasploit [57].

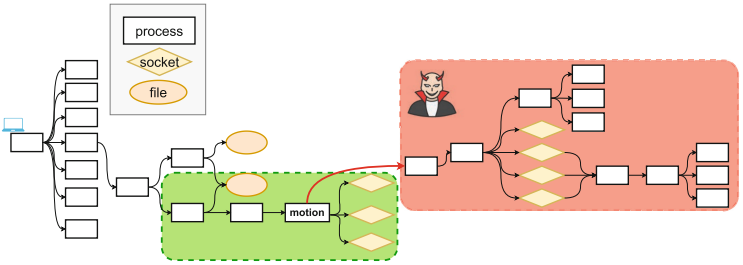


Fig. 6. Attacker injects and creates fileless malware as a child process of `motion` process. The provenance graph captures the attacker's behavior which can be used for detection.

We injected each attack TTP into five common IoT applications listed in Table 2 using a fileless wrapper [36]. Therefore, the IoT application's behavior captured in the provenance graph would contain additional nodes and edges (*i.e.,* malicious subgraphs) corresponding to the malicious behavior due to the injected attack TTPs. Because the malicious payload behaves differently than the benign application behavior, those malicious subgraphs are likely to contains rare and anomalous paths that will be detected by the Local Brain. In Fig. 6, we render the simplified provenance graph where we injected one of the attack TTPs to `motion`. It adds a subgraph whose size is proportional to the number of malicious activities performed.

We performed the program impersonation experiment five times for each of the four fileless IoT malware samples, with a total of twenty impersonation targets (Table 1), resulting in a total of 400 experiments. We conducted the APT

Table 1. ProvIoT is highly effective in distinguishing IoT malware impersonating as benign system process as evident from high F1 scores. Grey cells contain low F1 score to indicate indistinguishable malware behavior for system process, discussed in Sect. 7.3.

Impersonation target	Malware			
	BASHLITE	FritzFrog	ransomware	lizkabab
bash	0.98	0.96	0.96	0.98
cat	0.93	0.99	1.00	0.97
cp	0.92	0.97	0.92	0.95
cron	0.97	0.98	0.98	0.97
dash	0.95	0.96	1.00	0.98
dbus-daemon	0.94	0.95	0.92	0.98
dd	0.96	0.97	0.98	0.99
firefox	0.97	0.96	0.99	1.00
grep	0.96	0.97	0.94	0.95
java	0.96	0.96	0.96	0.98
ls	0.99	0.96	0.94	0.98
nginx	0.97	0.98	0.98	0.96
perl	0.96	0.96	0.95	0.97
ps	0.98	0.97	0.95	0.97
python	0.93	0.97	0.93	0.99
rm	0.92	0.96	0.93	0.98
service	0.93	0.95	0.90	0.99
sh	0.96	0.97	0.91	0.98
smbd	0.96	0.96	0.99	0.99
sshd	0.97	0.96	0.97	0.98
Average	**0.96**	**0.97**	**0.96**	**0.98**

scenario seven times on each of the five APT attack stages for five IoT applications (Table 5), totaling 175 experiments to build the APT dataset. Combining all our experiments, we conducted a total of 575 experiments (175 APT + 400 malware) to create the anomalous dataset. The provenance graphs collected from the malware evaluation have an average of 11,726.98 causal paths, 207.25 vertices, and 211.35 edges. The provenance graphs for the APT Kill chain scenario have an average of 19,716.37 causal paths, 435.49 vertices, and 481.40 edges. Interested readers can refer to Appendix Sect. A for further details.

7.2 Experimental Protocol

To generate the training and validation sets, we extract all the causal paths from the provenance graphs generated during benign deployment, reserving 90% of the data for training and 10% for validation. To generate the test set, we extract the rarest 15% of causal paths from the malicious testbeds, which simulates a real environment that has been attacked [43,82] and includes a mix of benign and anomalous paths. The Local Brain instances train on the benign training data and propagate their model weights to the Cloud Brain. The Cloud Brain then performs federated aggregation on those models to generate a global model, then propagates the global model back to the Local Brain instances. Each Local Brain tunes its detection threshold using its own validation set. In intrusion detection, we emphasize the importance of *unsupervised* learning because the defender should not make strong a priori assumptions about the attacker's behaviors.

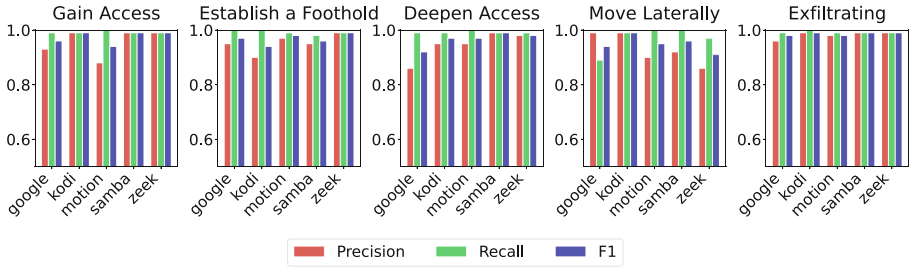


Fig. 7. High detection accuracy of ProvIoT against APT attacks using federated learning, some rare exceptions which are discussed in Sect. 7.4.

7.3 IoT Malware Detection

To represent a wide variety of malware, we selected two popular IoT malwares from [37], a natively fileless IoT malware [42], and a typical ransomware that would target an IoT system. We injected these well-known IoT malwares into trusted system processes using a fileless wrapper [36] to impersonate them. The detection results, summarized in Table 1, demonstrate that ProvIoT achieves high F1 scores for the majority of combinations, ranging from 0.96 to 0.98. This indicates that even when IoT malware is fileless and impersonates benign programs, its behavior remains distinct from the original system behavior.

However, some (impersonation target, malware) pairs, highlighted in Table 1, proved challenging for ProvIoT to reliably detect: `BASHLITE` was able to effectively masquerade as `cp` and `rm` because it primarily performs file copy and delete operations on the local device while preparing to participate in the botnet; `ransomware` effectively impersonated `cp` with large amounts of file copy operations, `dbus-daemon` with significant inter-process communication for cryptographic exchanges, `service` with manipulation of antivirus services, and `sh` with command execution.

7.4 APT Detection

The consistently high detection accuracy [41,82] of ProvIoT, as measured by precision, recall, and F1 score, is showcased in Fig. 7. Outside some rare exceptions, which will be discussed in more depth, the precision ranges from 0.93 to 0.99, the recall ranges from 0.97 to 1, and F1 scores range from 0.95 to 0.99. The results show that ProvIoT can reliably detect APT attacks while limiting the number of false alarms.

ProvIoT generates more false positives than false negatives, evidenced by its higher average recall (99%) than average precision (95%). This trend is also seen in other anomaly detection systems [41,82]. The high F1 score shows that the threshold is chosen in such a way that the actual anomalous behaviors (true positives) are detected rather than reducing FPs. Therefore, ProvIoT does not compromise on its detection ability to address false positive rates.

False Negative Cases. Even with path-based behavioral modeling, certain attack cases (*e.g., move laterally (S4)* attack for `google`) are hard to detect because the attacker's behavior is extremely similar to the application's benign behavior. The precision is 0.99 and the recall is 0.89, which is much lower than the second-lowest recall rate of 0.97. The *move laterally (S4)* stage scans for vulnerable ports to exploit, which is behaviorally similar to `google` scanning ports for available IP cameras.

False Positive Cases. ProvIoT has delivered steady and robust detection performance across our various APT workloads (Table 2). Against some APT stages, ProvIoT had a relatively high false positive rate such as *Deepen Access (S3)* for `google` has precision of 0.86 and recall of 0.99, *Establishing a Foothold (S2)* for `kodi` has precision of 0.90 and recall of 1, *Gain Access (S1)* for `motion` has precision of 0.88 and recall of 1; *Move Laterally(S4)* for `samba` has precision of 0.86 and recall of 0.99 and `zeek` has precision of 0.86 and recall of 0.97.

These instances of high false positive rates are due to system interactions with high behavioral variance. We investigated these cases and outlined the explanations based on the ground truth:`kodi` often reads hidden configuration files, downloads files containing streaming links from the internet and writes them to temporary locations; `google` creates and stops many short-lived threads; `motion` changes directory and file permission configuration for camera video storage; `samba` and `zeek` both scan and listen to different IPs and ports, which generates noisy provenance graphs (high variance). We see a high rate of false positives surrounding the creation and modification of temporary files and directories; since these behaviors are rare and not well-represented in the benign dataset, so they are marked as anomalous even when the actions are not malicious. The majority of the malicious paths were marked correctly as anomalous even though the precision score was below 0.90, the recall score was above 0.96. These results show that ProvIoT is very effective in detecting stealthy malware.

7.5 Federated Learning Benefits

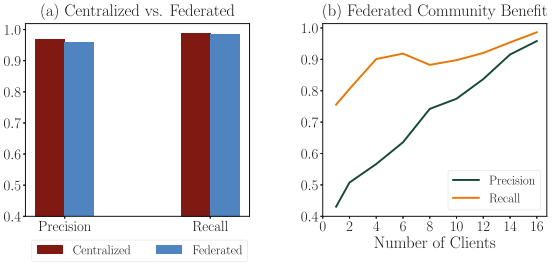


Fig. 8. (a) Federated performance is similar to centralized performance on the same data. (b) Increasing the number of clients increases performance by increasing the amount of data in the system.

We evaluate ProvIoT's federated approach against a traditional centralized architecture using `kodi` as a representative application. Figure 8(a) shows that ProvIoT trades just 1% precision for the scalability, privacy, and reliability benefits of the federated architecture. The centralized model was trained on the full dataset and achieved 0.97 precision and 0.99 recall. For ProvIoT, we used the 16 clients from our benign deployment that had `kodi` installed for training, then evaluated those models in our malicious testbeds. In this experiment, ProvIoT achieved 0.96 precision and 0.99 recall, performing almost identically to the centralized approach.

To demonstrate how ProvIoT is able to overcome the data view limitation of provenance-based anomaly detection on IoT devices, we visualize the average performance of the Local Brains as more clients are incorporated into the system in Fig. 8(b). By adding new Local Brains that see different data, the Cloud Brain is able to aggregate the incoming models to export a global model that better understands the full benign distribution. These model improvements manifest in improved recall and precision as new clients are introduced.

ProvIoT's federated approach provides critical benefits for IoT in data localization and privacy. The primary security benefit is localized detection, which reduces network overhead, allows detection in the absence of a network connection, and distributes the global detection workload across the federated devices. Further, because we only share model weights, specific system events are not shared with the network, which preserves the privacy of the data.

7.6 ProvIoT Overhead

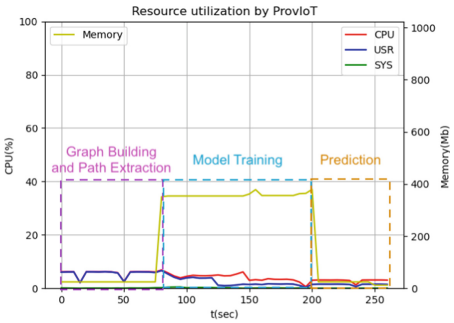


Fig. 9. On RaspberryPi 4B, Local Brain's processing and prediction uses <10% CPU and 65MB memory. Model training takes about 375MB memory and <10% CPU.

We experimentally demonstrate the overhead imposed by ProvIoT using an event database containing 7,085 process creation events, 56,587 file interactions, and 3,608 network interactions. This is typical for 24 h of execution. We experimented using different ARM IoT devices such as RaspberryPi 4B board [8] with

four CPU cores and 8 GB memory for CPU only device; Jetson Nano [65] with four CPU cores, 4 GB memory and NVIDIA gpu; and Google Edge TPU [39] with single core, 512 MB memory, and edge TPU ML accelerator. To train a reliable model for `kodi`, ProvIoT requires two weeks worth of data, which results in 5.46 GB of data and four weight synchronizations.

To accurately characterize the overhead imposed on the edge IoT devices, we need to consider two different modes of execution: training and detection. Training occurs infrequently (approximately once per week) and requires less than 10% of the CPU processing power and less than 375MB of memory for less than four minutes as shown in Fig. 9. Detection occurs frequently (approximately once per day) and requires less than 10% of the CPU processing power and less than 65MB of memory for less than two minutes as shown in Fig. 9. Even during peak resource utilization (*i.e.,* during training), ProvIoT does not monopolize the IoT resources. Many home IoT devices, such as smart fridges, thermostats, and doorbells [2,3,5,11,16] contain sufficient memory to support on-device training.

8 Limitations

Hardware Constraints. The most prominent limitation of ProvIoT is the memory utilization of 375 MB during local training phase as measured in Sect. 7.6. Therefore, ProvIoT cannot be used for IoT which do not have at least 512 MB of RAM, such as ESP32 boards. ProvIoT is suitable for mid to large scale IoT devices. Real world vendors need to ensure that their products posses the required resources before deploying our system.

To reduce memory overhead, it is possible to train the local models on smaller batches of data, but train more frequently. This approach "flattens the curve" of resource usage, requiring more total computation, but reducing the peak memory usage. Increasing the training frequency may also increase the models' vulnerability to incremental dataset shifting attacks. The Local Brain training frequency can be modulated independently of the Cloud Brain synchronization frequency.

Privacy of Federated Learning. Recent advancements have shown that attackers can use model weights to infer statistics about the training dataset. These statistics can then be used to craft targeted payloads and APT stages that blend in with the typical behavior of the system to evade detection. To protect the confidentiality and ensure the integrity of the model weights, communication between the Local Brains and the Cloud Brain should be encrypted and signed using public/private key pairs, which can be distributed by the vendor during manufacturing. To further improve the privacy preservation, the communication of model weights and computation of federated averaging can leverage recent advancements in fully homomorphic encryption for IoT devices [17,54], which shifts the heavy computational workload to the resource-rich Cloud Brain; this method would preserve the privacy of the IoT devices even against an attacker with full read access to the Cloud Brain and the capacity to recover private data from model weights alone.

Poisoning on Federated Learning. In ProvIoT, the design tradeoff between false positives due to benign software evolution and vulnerability to malicious incremental model shifting attacks is parameterized by the frequency of training and synchronization between the Local Brains and the Cloud Brain. In the real world, this tradeoff is of critical importance and will require careful consideration by experts on the security requirements of the specific application of ProvIoT.

9 Related Work

IoT Security. With the growth of IoT, a significant number of vulnerabilities have been identified in IoT devices [58,74,80], protocols [86], applications, and platforms [38]. In response to IoT attacks, diverse detection and prevention approaches have been proposed, such as network-based solutions [75], platform-based solutions [32,69,72,73] and application-based solutions [44,82]. Our work defends against stealthy attacks including fileless malware and APTs.

Cosson et al. [32] and Rieger et al. [69] has proposed a centralized node-level monitoring system for IoT using network traffic. However, it requires the local devices to send their local data to a centralized server where the detection occurs. ProvIoT has a major advantage over [32] because the users' data does not leave the local device and detection occurs on the local device without a network connection. [60,63,71] have showed how to do federated anomaly detection on IoT, but solely focused on network data. While the network data is important, stealthy attacks can easily circumvent those defenses with specially crafted network packets. To the best of our knowledge, we are the first to propose a federated, privacy preserving, collaborative learning framework using host-level provenance data for IoT.

IoT Defenses. General intrusion detection [37,81] approaches have been extensively studied. For example, [28] and [68] designed defenses to detect routing attacks. However, their work focuses on the 6LoWPAN protocol. Our work focuses on creating a generalized federated framework for IoT.

Recently, several anomaly-based solutions have been proposed to detect different IoT attacks. SDN-based approaches [66], signature-based approaches [48] and machine learning based approaches [24,30,56,63,64] have been proposed to detect IoT botnet attacks such as Mirai. However, these approaches only focus on analyzing network traffic, limiting their capability in detecting attacks with minimal network footprints. The most directly related previous work is [32,69], which forwards telemetry data for the entire IoT fleet to a central server for anomaly detection; ProvIoT improves upon the privacy and scalability of [32,69] by enabling on-device detection with federated learning and provenance analysis.

10 Discussion and Future Work

Real-Time Prevention. Although we focus on a detection system in this paper, ProvIoT can be easily extended to provide real-time prevention [23] (*e.g.,* blocking or killing anomalous processes). ProvIoT can also be augmented with other kinds of defenses (*e.g.,* dynamic quarantine or deep inspection) when it raises alerts. ProvIoT supports online forensic analysis including backtracking analysis and data query by leveraging its extensive system event collection.

Applicability to IoT Devices with Other OSes. While our current implementation and evaluation mainly focuses on Linux-based IoT devices, our approach is general and applicable to the devices with other operating systems, such as Windows, TinyOS, or Riot. For example, the Windows OS also has a system to log system events [4] that ProvIoT can use to generate provenance graphs for training and anomaly detection.

Trends in IoT Capacity and ML Overhead. As a step towards bringing powerful provenance-based threat detection to IoT devices, ProvIoT synergizes both with advancements in increasing hardware power in IoT devices and low-resource ML [10]. Indeed, recent work [52] has demonstrated training a 43-layer CNN in less than 256KB of RAM. We envision new works that combine ProvIoT's federated architecture with innovations in memory-constrained provenance analysis to extend on-device protection to the general IoT space.

11 Conclusion

In this paper, we present ProvIoT, a novel end-to-end edge-cloud collaborative security framework for IoT security. ProvIoT adapts modern provenance graph-based anomaly detection to IoT devices. ProvIoT is the first anomaly detection framework to perform on-device provenance analysis with federated learning in IoT devices. ProvIoT preserves the privacy of local system events and achieves high detection accuracy while incurring low overhead and enabling localized detection. We extensively evaluated ProvIoT with a realistic provenance dataset against real-world IoT malware and APT attack campaigns. ProvIoT detects fileless malware and APT attacks with an average F1-score of 0.97 and 0.99, respectively. During periodic detection cycles, ProvIoT incurs less than 10% CPU overhead, 65MB memory overhead, and does not require network connectivity. The detection with infrequent training cycles incurs similar CPU overhead, less than 375MB memory overhead, and up to 2MB network bandwidth consumption for model updates.

Acknowledgments. We thank the anonymous reviewers for their helpful feedback.

A Appendix

A.1 IoT Workload.

The Table 2 shows the typical usage for the IoT applications. Typical usage for media center (*e.g.,* `kodi` [47]) is to browse different streams to find playable and downloadable content. `kodi` was used to download different medias from the wed along with browsing different steams. A voice assistant such as `Google Assistant` [6] was used for answering common questions such as "what is the weather like?". An IP camera (*e.g.,* `motion` [7]) was used to stream our lab setting from our home. We used a network attached storage unit to access files from remote locations as well as to modify the files. Finally, we used a network security monitoring tool (*e.g.,* `zeek` [85]) to sniff and inspect at the network traffic that was generated in our lab environment.

Table 2. The IoT applications chosen for evaluation as well as their usage examples.

Usages	Application	Scenario
Voice Assistant	`google`	Inquired general knowledge and everyday household questions to Google Assistant.
Media Center	`kodi`	Updated media streams and played media during different parts of the day.
IP Camera	`motion`	Started streaming multiple live camera streaming server and watched them.
Network Attached Storage	`samba`	Performed network storage action such as list all the files, delete a file, or add a file.
Network Security Monitor	`zeek`	Investigated the network traffic coming from IoT using Zeek

A.2 Dataset Statistics.

This section contains the data set details shown in Tables 3 and 4. In Table 3 the benign dataset is represented where we experimented with five commonly used IoT programs [33] and twenty prevalent Linux system programs [53]. Table 4 shows the malicious data set which consists of two parts: four IoT malware which impersonated the twenty Linux system programs and APT kill chain scenarios conducted using the five IoT programs.

Table 3. Number of vertices and edges used to create a benign profile for IoT applications and system programs

	Avg. # of causal paths	Avg. # of total vertices / edges	Avg. # of forward vertices / edges	Avg. # of backward vertices / edges
IoT Application				
google	571,052.33	159.0 / 314.0	95.67 / 216.0	63.33 / 98.0
kodi	29,946.89	210.33 / 273.78	149.33 / 176.89	61.0 / 96.89
motion	9,113.0	179.0 / 504.0	5.0 / 4.0	174.0 / 500.0
samba	85,347.0	2,537.0 / 2,857.0	76.4 / 120.8	2,460.6 / 736.2
zeek	494,160.0	2,149.5 / 3,724.5	1,032.5 / 1,124.5	1,117.0 / 2,600.0
average	**237,923.84**	**1,046.97 / 1,534.66**	**271.78 / 328.44**	**775.19 / 1,206.22**
System Program				
bash	166,355.43	454.25 / 510.76	10.57 / 9.31	443.68 / 501.45
cat	184,346.43	310.51 / 210.9	9.0 / 6.99	301.51 / 203.91
cp	175,636.86	193.42 / 212.7	179.09 / 184.69	14.33 / 28.01
cron	214,827.71	327.16 / 241.85	10.27 / 9.96	316.89 / 231.89
dash	153,808.57	371.87 / 381.97	211.61 / 206.44	160.26 / 175.53
dbus-daemon	156,713.0	20.16 / 20.04	9.02 / 6.42	11.14 / 13.62
dd	213,601.29	995.5 / 1,003.6	551.68 / 501.81	443.82 / 501.79
firefox	176,843.86	194.22 / 504.56	15.84 / 18.78	178.38 / 485.78
grep	212,413.86	191.51 / 502.32	13.51 / 16.43	178.0 / 485.89
java	169,180.71	133.94 / 222.4	17.44 / 19.63	116.5 / 202.77
ls	179,185.86	213.62 / 356.47	10.25 / 9.3	203.37 / 347.17
nginx	258,367.17	514.27 / 514.13	500.76 / 501.26	13.51 / 12.87
perl	809.0	25.01 / 23.22	11.95 / 12.05	13.06 / 11.17
ps	181,846.43	834.01 / 998.14	369.21 / 501.77	464.8 / 496.37
python	161,755.57	365.71 / 348.31	11.51 / 8.14	354.2 / 340.17
rm	174,590.43	452.89 / 440.38	15.06 / 18.5	437.83 / 421.88
service	231.43	18.32 / 21.24	15.32 / 18.55	3.0 / 2.69
sh	208,367.43	445.01 / 851.27	4.16 / 357.78	440.85 / 493.49
smbd	201,559.57	355.37 / 371.15	9.69 / 3.39	345.68 / 367.76
sshd	182,601.57	233.04 / 234.15	9.35 / 6.6	223.69 / 227.55
average	**168,652.11**	**332.49 / 398.48**	**99.26 / 120.89**	**233.23 / 277.59**

Table 4. Number of vertices and edges used to create IoT Malware and APT attack profile

	Avg. # of causal paths	Avg. # of total vertices / edges	Avg. # of forward vertices / edges	Avg. # of backward vertices / edges
IoT Malware				
BASHLITE	110.5	21.0 / 21.0	4.0 / 3.0	17.0 / 18.0
FritzFrog	46,253.8	751.0 / 747.4	248.6 / 246.8	502.4 / 500.6
lizkebab	293.2	29.0 / 33.0	6.0 / 4.0	23.0 / 29.0
randomware	250.4	28.0 / 44.0	8.0 / 12.0	20.0 / 32.0
average	**11,726.98**	**207.25 / 211.35**	**66.65 / 66.45**	**140.6 / 144.9**
APT Kill Chain Scenario				
Gain Access (S1)	2,789.6	510.6 / 554.8	495.2 / 537.6	15.4 / 17.2
Establish a Foothold (S2)	46,763.75	470.25 / 550.12	398.38 / 429.5	71.88 / 120.62
Deepen Access (S3)	1,192.4	171.0 / 202.6	164.0 / 195.0	7.0 / 7.6
Move Laterally (S4)	27,314.33	97.5 / 116.0	70.17 / 84.83	27.33 / 31.17
Look, Learn and Remain (S5)	20,521.75	928.12 / 983.5	897.38 / 929.62	30.75 / 53.88
average	**19,716.37**	**435.49 / 481.40**	**405.03 / 435.31**	**30.47 / 46.09**

Table 5. APT TTPs for cyber-killchain stages

Cyber-killchain Stages	Techniques (ATTCK TTP)	Scenarios
Gain Access (S1)	Exploitation for Client Execution (T1203)	Attackers modify a benign looking executable, but once the user opens the application it can be used by the attacker for arbitrary code execution
	File and Directory Permissions Modification (T1222)	Attacker modifies objects in the system so that it can be copied by lower privilege users that the attacker has hijacked
Establish a Foothold (S2)	Data from Local System (T1005)	Attacker moves around the file system, finding files that contain valuable information
	Exfiltration Over C2 Channel (T1041)	Attacker downloads valuable files into a local directory
Deepen Access (S3)	Create and Modify system process (T1543)	Attacker creates a system process that can run in the background and do reconnaissance or mine information
	Service Stop (T1489)	Attacker stops firewall or external IDS services so that they cannot detect the APT
Move Laterally (S4)	Process injection (T1055)	Attacker injects a vulnerable process such as a trojan into a benign application so that IDS cannot differentiate
Look, Learn, and Remain (S5)	System Information Discovery (T1082)	Attacker discovers system hardware information so that they can craft better exploits or exploit hardware vulnerabilities
	Network Service Scanning (T1046)	Attackers scan network services to find services they can use as backup or use as a secondary mode of connections
	Network Sniffing (T1040)	Attackers sniff the network to find insecure SSL connections or any other connections to extract valuable information

A.3 APT Scenarios

The advanced Persistent Threat (APT) scenario was established in our malicious testbed by loading APT kill-chain components using fileless wrapper (Table 5). The APT attack vectors were coordinated to comprise the end-to-end attack campaign referring to MITRE ATT&CK framework.

References

1. Insteon hub 2242–222 - lack of web and API authentication (2013). https://www.exploit-db.com/exploits/27284. Accessed 26 May 2023
2. Apple watch ram size comparison chart: how much ram does apple watch have? (2015). https://www.knowyourmobile.com/wearable-technology/apple-watch-ram-size/. Accessed 26 May 2023
3. Google nest - support (2015). https://support.google.com/googlenest/answer/9230098. Accessed 26 May 2023
4. Auditing security events (2017). https://goo.gl/FkaDCa
5. Google home mini teardown, comparison to echo dot, and giving technology a voice (2017). https://tinyurl.com/ykbay2fu. Accessed 26 May 2023
6. Google Assistant, your own personal Google (2018). https://assistant.google.com/
7. Motion (2018). https://motion-project.github.io/

8. Raspberry Pi - Teach, Learn, and Make with Raspberry Pi (2018). https://www.raspberrypi.org
9. Gensim: Topic modelling for humans (2019). https://radimrehurek.com/gensim/index.html
10. Tinyml foundation (2019). https://www.tinyml.org/. Accessed 25 May 2023
11. Inside amazon's ring alarm system (2020). https://tinyurl.com/yck5jm4m. Accessed 26 May 2023
12. Cyber kill chain® — lockheed martin (2021). https://www.lockheedmartin.com/en-us/capabilities/cyber/cyber-kill-chain.html. Accessed 24 Jul 2021
13. Process injection: Ptrace system calls, sub-technique t1055.008 - enterprise — mitre att&ck® (2021). https://attack.mitre.org/techniques/T1055/008/. Accessed 23 Jul 2021
14. Cloud-based data platform for cybersecurity, it operations and devops — splunk (2022). https://www.splunk.com/. Accessed 23 Jul 2021
15. Iot is a gold mine for hackers using fileless malware for cyberattacks - techrepublic (2022). https://tek.io/30dBnIU. Accessed 23 Jul 2021
16. Smart refrigerator with family hub (2022). https://tinyurl.com/4kz6z6z5. Accessed 26 May 2023
17. Acar, A., Aksu, H., Uluagac, A.S., Conti, M.: A survey on homomorphic encryption schemes: Theory and implementation. ACM Comput. Surv. **51**(4), 1–35 (2018). https://doi.org/10.1145/3214303
18. Acar, A., et al.: Peek-a-boo: I see your smart home activities, even encrypted! In: Proceedings of the 13th ACM Conference on Security and Privacy in Wireless and Mobile Networks, pp. 207–218. WiSec 2020, Association for Computing Machinery, New York, NY, USA (2020). https://doi.org/10.1145/3395351.3399421
19. Ahmad, A., Lee, S., Peinado, M.: HARDLOG: practical tamper-proof system auditing using a novel audit device. In: 2022 IEEE Symposium on Security and Privacy (SP), pp. 1791–1807 (2022)
20. Alrawi, O., et al.: The circle of life: a large-scale study of the IoT malware lifecycle. In: USENIX Security Symposium, pp. 3505–3522 (2021)
21. Antonakakis, M., et al.: Understanding the Mirai botnet. In: 26th USENIX Security Symposium (USENIX Security 17), pp. 1093–1110. USENIX Association, Vancouver, BC (2017). https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/antonakakis
22. Armis Security: Blueborne: Bluetooth exposes android, linux, windows and iOS devices to airborne attacks (2017). https://www.armis.com/research/blueborne/
23. Babun, L., Celik, Z.B., McDaniel, P., Uluagac, S.: Real-time analysis of privacy-(un)aware IoT applications. Proc. Priv. Enhancing Technol. **2021**, 145–166 (2021). https://doi.org/10.2478/popets-2021-0009
24. Bahşi, H., Nõmm, S., La Torre, F.B.: Dimensionality reduction for machine learning based IoT botnet detection. In: 2018 15th International Conference on Control, Automation, Robotics and Vision (ICARCV), pp. 1857–1862. IEEE (2018)
25. Bansal, A., Kandikuppa, A., Chen, C.Y., Hasan, M., Bates, A., Mohan, S.: Towards efficient auditing for real-time systems. In: Atluri, V., Di Pietro, R., Jensen, C.D., Meng, W. (eds.) Computer Security – ESORICS 2022. ESORICS 2022. Lecture Notes in Computer Science, vol. 13556, pp. 614–634. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-17143-7_30
26. Barr-Smith, F., Ugarte-Pedrero, X., Graziano, M., Spolaor, R., Martinovic, I.: Survivalism: systematic analysis of windows malware living-off-the-land. In: IEEE symposium on security and privacy (SP). In: IEEE Symposium on Security and Privacy (SP), pp. 1557–1574 (2021). https://doi.org/10.1109/sp40001.2021.00047

27. Bonawitz, K., et al.: Towards federated learning at scale: system design. arXiv.org (2019)
28. Bostani, H., Sheikhan, M.: Hybrid of anomaly-based and specification-based IDS for internet of things using unsupervised OPF based on MapReduce approach. Comput. Commun. **98**, 52–71 (2017)
29. Chaudhary, A., Mittal, H., Arora, A.: Anomaly detection using graph neural networks. In: 2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon), pp. 346–350 (2019). https://doi.org/10.1109/COMITCon.2019.8862186
30. Chawathe, S.S.: Monitoring IoT networks for botnet activity. In: 2018 IEEE 17th International Symposium on Network Computing and Applications (NCA), pp. 1–8. IEEE (2018)
31. Chen, J., et al.: Iotfuzzer: Discovering memory corruptions in IoT through app-based fuzzing. In: NDSS (2018)
32. Cosson, A., Sikder, A.K., Babun, L., Celik, Z.B., McDaniel, P., Uluagac, A.S.: Sentinel: a robust intrusion detection system for IoT networks using kernel-level system information. In: Proceedings of the International Conference on Internet-of-Things Design and Implementation, pp. 53–66 (2021)
33. Costin, A., Zaddach, J.: IoT Malware: comprehensive survey, analysis framework and case studies. BlackHat Briefings (2019). https://bit.ly/3DFrCBA
34. Cozzi, E., Graziano, M., Fratantonio, Y., Balzarotti, D.: Understanding Linux malware. In: 2018 IEEE Symposium on Security and Privacy (SP), pp. 161–175. IEEE (2018)
35. CrowdStrkie: Endpoint Detection and Response (EDR), Tech. rep., CrowdStrkie (2020)
36. Cybersecurity, A.: Malware using new Ezuri memory loader — at&t alien labs (2021). https://cybersecurity.att.com/blogs/labs-research/malware-using-new-ezuri-memory-loader. Accessed 23 Jul 2021
37. Ding, F., et al.: DeepPower: non-intrusive and deep learning-based detection of IoT Malware using power side channels. In: Proceedings of the 15th ACM Asia Conference on Computer and Communications Security, pp. 33–46 (2020)
38. Fernandes, E., Jung, J., Prakash, A.: Security analysis of emerging smart home applications. In: IEEE S&P (2016)
39. Google: Edge TPU - run inference at the edge — google cloud (2021). https://cloud.google.com/edge-tpu. Accessed 23 Jul 2021
40. Google: Intro to autoencoders (2021). https://www.tensorflow.org/tutorials/generative/autoencoder
41. Han, X., et al.: {SIGL}: Securing software installations through deep graph learning. In: 30th USENIX Security Symposium (USENIX Security 21), pp. 2345–2362 (2021)
42. Harpaz, O.: FritzFrog: a new generation of peer-to-peer botnets - guardicore (2020). https://bit.ly/3mJzyeq. Accessed 23 Jul 2021
43. Hassan, W.U., et al.: NoDoze: combatting threat alert fatigue with automated provenance triage. In: NDSS (2019)
44. Jia, Y.J., et al.: ContexIoT: towards providing contextual integrity to appified IoT platforms. In: NDSS (2017)
45. King, S.T., Chen, P.M.: Backtracking intrusions. ACM SIGOPS Oper. Syst. Rev. **37**, 223–236 (2003). https://doi.org/10.1145/945445.945467
46. Kipf, T.N., Welling, M.: Variational graph auto-encoders. arXiv preprint arXiv:1611.07308 (2016)

47. Kodi — Open source home theater software (2018). https://kodi.tv/
48. Kumar, A., Lim, T.J.: Early detection of mirai-like IoT bots in large-scale networks through sub-sampled packet traffic analysis. arXiv preprint arXiv:1901.04805 (2019)
49. Lamport, L., Shostak, R., Pease, M.: The byzantine generals problem. ACM Trans. Program. Lang. Syst. **4**(3), 382–401 (1982). https://doi.org/10.1145/357172.357176
50. Le, Q., Mikolov, T.: Distributed representations of sentences and documents. In: International Conference on Machine Learning, pp. 1188–1196 (2014)
51. Li, Z., Chen, Q.A., Yang, R., Chen, Y., Ruan, W.: Threat detection and investigation with system-level provenance graphs: a survey. Comput. Secur. **106**, 102282 (2021)
52. Lin, J., Zhu, L., Chen, W.M., Wang, W.C., Gan, C., Han, S.: On-device training under 256KB memory. In: Advances in Neural Information Processing Systems, vol. 35, pp. 2941–2295 (2022)
53. Liu, Y., et al.: Towards a timely causality analysis for enterprise security. In: NDSS (2018)
54. Matsumoto, M., Oguchi, M.: Speeding up encryption on IoT devices using homomorphic encryption. In: 2021 IEEE International Conference on Smart Computing (SMARTCOMP), pp. 270–275 (2021). https://doi.org/10.1109/SMARTCOMP52413.2021.00059
55. McMahan, B., Moore, E., Ramage, D., Hampson, S., y Arcas, B.A.: Communication-efficient learning of deep networks from decentralized data. In: Artificial Intelligence and Statistics, pp. 1273–1282. PMLR (2017)
56. Meidan, Y., et al.: N-Baiot: network-based detection of IoT botnet attacks using deep autoencoders. arXiv preprint arXiv:1805.03409 (2018)
57. metasploit: metasploit (2021). https://www.metasploit.com/. Accessed 29 Nov 2021
58. Mirai Attacks (2016). https://goo.gl/QVv89r
59. MITRE: Mitre att&ck® (2023). https://attack.mitre.org/. Accessed 23 Jul 2021
60. Mothukuri, V., Khare, P., Parizi, R.M., Pouriyeh, S., Dehghantanha, A., Srivastava, G.: Federated-learning-based anomaly detection for IoT security attacks. IEEE Internet Things J. **9**(4), 2545–2554 (2021)
61. Mukherjee, K.: ProvIoT: detecting stealthy attacks in IoT through federated edge-cloud security (2023). https://github.com/syssec-utd/proviot
62. Mukherjee, K., et al.: Evading provenance-based ML detectors with adversarial system actions. In: USENIX Security Symposium (SEC) (2023)
63. Nguyen, T.D., Marchal, S., Miettinen, M., Dang, M.H., Asokan, N., Sadeghi, A.R.: Diot: a crowdsourced self-learning approach for detecting compromised IoT devices. arXiv preprint arXiv:1804.07474 (2018)
64. Nõmm, S., Bahşi, H.: Unsupervised anomaly based botnet detection in IoT networks. In: 2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA), pp. 1048–1053. IEEE (2018)
65. NVIDIA: Nvidia jetson nano developer kit — nvidia developer (2022). https://developer.nvidia.com/embedded/jetson-nano-developer-kit. Accessed 23 Jul 2021
66. Ozcelik, M., Chalabianloo, N., Gur, G.: Software-defined edge defense against IoT-based DDOs. In: 2017 IEEE International Conference on Computer and Information Technology (CIT), pp. 308–313. IEEE (2017)
67. Pan, S., Hu, R., Long, G., Jiang, J., Yao, L., Zhang, C.: Adversarially regularized graph autoencoder for graph embedding. arXiv preprint arXiv:1802.04407 (2019)

68. Raza, S., Wallgren, L., Voigt, T.: Svelte: real-time intrusion detection in the internet of things. Ad Hoc Netw. **11**(8), 2661–2674 (2013)
69. Rieger, P., Chilese, M., Mohamed, R., Miettinen, M., Fereidooni, H., Sadeghi, A.R.: Argus: context-based detection of stealthy IoT infiltration attacks. arXiv preprint arXiv:2302.07589 (2023)
70. Shafi, M., et al.: 5G: a tutorial overview of standards, trials, challenges, deployment, and practice. IEEE J. Sel. Areas Commun. **35**(6), 1201–1221 (2017). https://doi.org/10.1109/JSAC.2017.2692307
71. Shahid, O., Mothukuri, V., Pouriyeh, S., Parizi, R.M., Shahriar, H.: Detecting network attacks using federated learning for IoT devices. In: 2021 IEEE 29th International Conference on Network Protocols (ICNP), pp. 1–6. IEEE (2021)
72. Sikder, A.K., Aksu, H., Uluagac, A.S.: 6thSense: a context-aware sensor-based attack detector for smart devices. In: 26th USENIX Security Symposium (USENIX Security 17), pp. 397–414. USENIX Association, Vancouver, BC (2017). https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/sikder
73. Sikder, A.K., Aksu, H., Uluagac, A.S.: A context-aware framework for detecting sensor-based threats on smart devices. IEEE Trans. Mob. Comput. **19**(2), 245–261 (2020). https://doi.org/10.1109/TMC.2019.2893253
74. Sikder, A.K., Petracca, G., Aksu, H., Jaeger, T., Uluagac, A.S.: A survey on sensor-based threats and attacks to smart devices and applications. IEEE Commun. Surv. Tutorials **23**, 1125–1159 (2021). https://doi.org/10.1109/COMST.2021.3064507
75. Sivaraman, V., Gharakheili, H.H., Vishwanath, A., Boreli, R., Mehani, O.: Network-level security and privacy control for smart-home IoT devices. In: WiMob, pp. 163–167 (2015)
76. Team, D.: Deep graph library: easy deep learning on graphs (2022). https://www.dgl.ai/. Accessed 21 Sep 2021
77. Team, K.: Keras: the Python deep learning API (2021). https://keras.io/
78. Trend Micro: Brickerbot malware permanently bricks IoT devices (2017). https://tinyurl.com/2wc4vw5b
79. Introducing Arm TrustZone (2018). https://developer.arm.com/technologies/trustzone
80. VPNFilter (2018). https://blog.talosintelligence.com/2018/05/VPNFilter.html
81. Wang, J., et al.: IoT-praetor: undesired behaviors detection for IoT devices. IEEE Internet Things J. **8**(2), 927–940 (2020)
82. Wang, Q., et al.: You are what you do: Hunting stealthy malware via data provenance analysis. In: NDSS (2020)
83. Williams, M.: A new philips hue security patch keeps hackers from taking control of your network (2019). https://tinyurl.com/yejh839k
84. Ying, R., Lou, Z., You, J., Wen, C., Canedo, A., Leskovec, J.: Neural subgraph matching. CoRR abs/2007.03092 (2020), https://arxiv.org/abs/2007.03092
85. Zeek (2021). https://zeek.org/
86. Critical flaw identified in Zigbee smart home devices (2015). https://goo.gl/BFBa1X