Graph Learning for Interactive Threat Detection in Heterogeneous Smart Home Rule Data

Guangjing Wang
Michigan State University
USA
wanggu22@msu.edu

Nikolay Ivanov
Michigan State University
USA
ivanovn1@msu.edu

Bocheng Chen
Michigan State University
USA
chenboc1@msu.edu

Qi Wang
University of Illinois
Urbana-Champaign
USA
qiwang11@illinois.edu

ThanhVu Nguyen
George Mason University
USA
tvn@gmu.edu

Qiben Yan
Michigan State University
USA
qyan@msu.edu

ABSTRACT

The interactions among automation configuration rule data have led to undesired and insecure issues in smart homes, which are known as interactive threats. Most existing solutions use program analysis to identify interactive threats among automation rules, which is not suitable for closed-source platforms. Meanwhile, security policy-based solutions suffer from low detection accuracy because the pre-defined security policies in a single platform can hardly cover diverse interactive threat types across heterogeneous platforms. In this paper, we propose Glint, the first graph learning-based system for interactive threat detection in smart homes. We design a multi-scale graph representation learning model, called IT-GNN, for both homogeneous and heterogeneous interaction graph pattern learning. To facilitate graph learning, we build large interaction graph training datasets by multi-domain data fusion from five different platforms. Moreover, Glint detects drifting samples with contrastive learning and improves the generalization ability with transfer learning across heterogeneous platforms. Our evaluation shows that Glint achieves 95.5% accuracy in detecting interactive threats across the five platforms. Besides, we examine a set of user-designed blueprints in the Home Assistant platform and reveal four new types of real-world interactive threats, called "action block", "action ablation", "trigger intake", and "condition duplicate", which are cross-platform interactive threats captured by Glint.

CCS CONCEPTS

• **Security and privacy** → *Vulnerability scanners*; • **Computing methodologies** → **Information extraction**; **Learning latent representations**.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
SIGMOD'23, June 2023, Seattle, Washington, USA
© 2023 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

KEYWORDS

Graph learning; transfer learning; threat detection; smart home

ACM Reference Format:
Guangjing Wang, Nikolay Ivanov, Bocheng Chen, Qi Wang, ThanhVu Nguyen, and Qiben Yan. 2023. Graph Learning for Interactive Threat Detection in Heterogeneous Smart Home Rule Data. In *Proceedings of ACM Conference (SIGMOD'23).* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Modern homes are becoming smarter with the deployment of smart devices, such as smart locks, smart plugs, and smart ovens. Considering many factors, such as compatible device interactions, central voice control, and ease of operation, a user can deploy multiple systems to enable cross-platform automation features. A recent survey [9] shows that 82.4% smart home deployments have multiple automation rules to control one device, and 62.4% users deployed more than one platform at home. With the support of common application programming interfaces (APIs), apps from multiple platforms can interact with one another. For example, IFTTT [20] can connect to other platforms such as SmartThings [34], Amazon Alexa [4], and Home Assistant [5].

However, there could be undesirable or insecure interactions among a diverse set of devices, dubbed as interactive threats. Specifically, many factors such as user mistakes [18, 40] and various attacks [1, 51] can cause interactive threats during the operation of smart home systems, especially when a user configures multiple systems at the same time. For instance, in a smart home, suppose a Home Assistant app has set up the rule *"If smoke is detected, open the window"*, and the following automation rule *"Close the window when the outside temperature is high"* would force the window closed. As a result, the interaction between two automation rules will expose a potential problem "the window cannot open when smoke is detected". Therefore, it is imperative to manage the interaction logic among automation rule data to prevent property damage.

Existing approaches use code analysis to uncover the interactions among different apps. However, they suffer from two major deficiencies. **First**, most real-world platforms, such as IFTTT and Alexa, are closed-source, which renders these methods based on source code analysis or instrumentation ineffective [2, 11, 39]. **Second**, there are large and complex interactions across heterogeneous platforms. Many security policy-based approaches [7, 14, 43] assume that all automation rules run on a single platform [9]. They only focus on limited intra-app and inter-app analysis and perform evaluations on one platform, such as the open-source SmartThings. These methods need to pre-define threat patterns and security policies, which can hardly cover various threat types associated with complex rule interactions across platforms.

In this paper, for the first time, we address the critical research question: *how to manage smart home automation configuration data to avoid interactive threats across multiple* ***closed-source*** *platforms in* ***real time****, where the interactive threat patterns can hardly be enumerated in a* ***heterogeneous*** *smart home environment?* To address this research question, we aim at developing a data-driven approach to effectively mining the interactive threat patterns. Existing data mining-based approaches for threat detection [13, 22, 47, 54] usually take the event logs as input. Yet, they fail to fully leverage the interaction logic of the smart apps, as the rich semantics of events and the complex interaction logic of smart apps could hardly be represented by time-series data.

Graph neural network (GNN) models [15, 19, 21, 45] have shown a surge of success for many tasks such as graph or node classification, link prediction, and matching. GNNs have been used to learn both explicit node features and implicit structural graph features. Naturally, the interactions among rules from different platforms can be composed as a graph, in which a node is a rule, and an edge represents the interactions among different rules. The node features refer to the semantic-aware embeddings extracted by natural language processing (NLP) techniques [6]. In this paper, we formulate the interactive threat analysis as a task of GNN-based graph representation learning and propose **Glint**, **G**raph **l**earning for **int**eractive threat analysis, to secure the heterogeneous smart home platforms. There are three main challenges in designing Glint:

Challenge 1. *There is a lack of graph data with ground truth labels for learning interaction patterns in the closed-source smart home platforms.* We find that most smart home platforms provide detailed app descriptions, which contain rules about device control. Besides, previous studies [11, 14, 39] on the open-source platforms provide expert knowledge about the vulnerable interactions of smart devices. Thus, we can build and label large interaction graph datasets with the learned expert knowledge. Moreover, by fusing rule texts and event log information, we can build a real-time interaction graph. In this way, we can model interactive threat analysis as a supervised graph representation learning problem, which has two merits. First, it achieves high accuracy by learning a function from both labeled normal data and vulnerable data. Second, it achieves high efficiency in the testing phase with a well-trained model.

Challenge 2. *The interaction patterns can drift due to the drifting sample issue.* Drifting samples originate from the new types of interactive threats or the evolution of the existing threats, which makes it infeasible to enumerate all possible interactive threat patterns. Therefore, we design the ITGNN model with contrastive learning loss to learn graph embedding, such that we can compare samples with different labels and detect drifting samples. Compared with static analysis and dynamic testing, our data-driven approach leverages the GNN model to learn potential interactive threat patterns from the graph datasets, avoiding the reliance on pre-defined patterns. Therefore, Glint can mitigate the security policy coverage issue.

Challenge 3. *The availability of data varies across different platforms, which could drastically affect the overall performance of graph representation learning.* Fortunately, despite the disparity in the information format across different platforms, the rules configured in different platforms are all used to control smart devices and thus contain common semantics. Hence, we propose the cross-domain graph transfer learning method to transfer knowledge from heterogeneous platforms, which further enhances the model performance on the platforms with insufficient data.

We evaluate Glint on multiple datasets collected from 5 high-profile platforms with over 48,000 interaction graphs. We show that Glint can effectively detect interactive threats on single platforms and across multiple platforms. Remarkably, it detects the interactive threats in the SmartThings platform with 100% accuracy, and it detects the threats in heterogeneous interaction graphs with 95.5% accuracy and 95.6% F1 score. Based on drifting sample detection results, Glint also uncovers four new types of interactive threats in the user-designed blueprints in the Home Assistant platform: "action block", "action ablation", "trigger intake", and "condition duplicate". We make our data and code publicly available at https://github.com/seitlab/Glint.

In summary, we make the following contributions:

- We build the first semantic-based interaction graph datasets with ground truth labels by analyzing the trigger-action correlations among rules from closed-source platforms.
- We design the ITGNN model for interaction pattern learning across heterogeneous platforms. We leverage contrastive learning to detect drifting samples, and transfer learning to enhance the generalization ability of GNN models.
- We implement a novel GNN-based system, Glint, and perform a comprehensive evaluation on the real-life testbed. The results show that, for cross-platform interactive threat detection, Glint outperforms the state-of-the-art methods by 12.8% in precision and 12.6% in recall rate.

The rest of the paper is organized as follows. In Section 2, we define the interactive threat analysis problem. In Section 3, we present Glint and the designed graph model. In Section 4, we provide implementation details and evaluate the performance of Glint. We discuss related work in Section 5, and conclude in Section 6.

2 PROBLEM DEFINITION

We first introduce interaction graphs with a running example, and then we formally define interactive threats. The interactive threat patterns are mined based on graph representation learning.

2.1 Interaction Graph

An interaction graph is an abstract representation of interactions among rules across different platforms. Figure 1 shows an interaction graph that contains rules from three platforms: SmartThings, IFTTT, and Alexa. The detailed rule content of each node in Figure 1 is shown in Table 1. Automation rules contain "trigger-action" logic information, which means the execution of the first rule will trigger the execution of the second rule. For example, "*Alexa, play movies*"

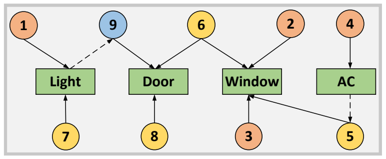


Figure 1: An interaction graph, where nodes (in circle shape) are rules from three platforms. These rules are connected via interacting devices and environment channels.

Table 1: Specific rule contents of nodes in the interaction graph, where the index is the node ID in Figure 1.

Index	Platform	Automation Rule Description
1	SmartThings	Turn off lights if playing movies.
2	SmartThings	If the outdoor temperature is between 65 °F and 80 °F, open windows after sun rise.
3	SmartThings	If outdoor temperature is below 60 °F, then close windows.
4	SmartThings	Turn on the air conditioner when temperature is above 85 °F.
5	IFTTT	If air conditioner is on, then close windows.
6	IFTTT	If the smoke alarm is beeping, then open the window and unlock the door.
7	IFTTT	If motion is detected, turn on lights.
8	IFTTT	If motion is detected, open the door.
9	Amazon Alexa	Lock the door if all lights are turned off.

has "trigger-action" correlation with Rule 1 in Table 1. Different rules can have correlations, which are connected via different devices and physical channels (e.g., temperature, smoke). Moreover, there could be an "action-trigger" correlation between two rules, which means the first rule's action will trigger the second rule's execution. For example, in Table 1, Rule 1 and Rule 9 interact via light. The action of Rule 1 ("turn off lights") will trigger the execution of Rule 9. As a result, the door is locked when watching movies if these two rules are executed. The interaction graph formalizes the interactions among devices, users, and the environment based on the "trigger-action" rules. Both the "trigger-action" and "action-trigger" correlations contain the causality logic among automation configuration rule data.

Given a set of n rules, the number of interaction graphs is uniquely determined. For example, if a user employs ten smart home apps, each associated with only one rule, there will be only one interaction graph in real-time. This is because the "action-trigger" correlations among different rules are determined. If there are no correlations between two rules, two nodes of rules in the interaction graph does not have an edge connection. Therefore, the number of rules affects the complexity or size of an interaction graph, rather than influencing the number of potential graphs in real-time.

2.2 Interactive Threat

Interactive threats refer to vulnerable interactions among different devices, users, and the environment, which contain threats or anomalies [2, 11, 14] and result in undesirable behaviors or severe security and privacy risks. Note that interactions that cause unexpected execution of devices are also regarded as interactive threats. When deploying automation rules, the users may misconfigure their devices, introducing threats in smart homes [18, 40]. During

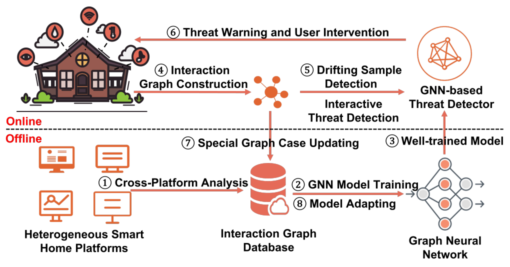

⑥ Threat Warning and User Intervention
④ Interaction Graph Construction
⑤ Drifting Sample Detection
GNN-based Threat Detector
Interactive Threat Detection
Online
Offline
⑦ Special Graph Case Updating
③ Well-trained Model
① Cross-Platform Analysis
② GNN Model Training
⑧ Model Adapting
Heterogeneous Smart Home Platforms
Interaction Graph Database
Graph Neural Network

Figure 2: Glint builds an interaction graph for a house, and uses a well-trained GNN model to automatically discover interactive threats and generate threat warnings.

the automation system execution, an attacker can attack a device, such as a voice assistant to control smart devices by executing inaudible voice commands [1, 51]. Alternatively, an attacker can trick a user into installing malicious apps to trigger unexpected interactive actions.

We formally define the task of managing automation data to avoid interactive threats as a graph representation learning problem. Given an interaction graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}, \mathcal{X}\}$, the nodes $\mathcal{V}$ are rules from different platforms, the edges $\mathcal{E}$ are correlations among different rules such as "trigger-action" or "action-trigger". Each node $v \in \mathcal{V}$ has a feature vector $x_v \in \mathcal{X}$, which is a word-embedding or sentence-embedding that encodes semantic information. For example, the word "sunset" is encoded as a 300-dimension embedding $[0.60014, \ldots, 0.35422]$ with an NLP library spaCy [35]. We derive the averaged word embedding of each word as the rule-level embedding, which is a node feature. While node features in homogeneous graphs come from the same feature space to better characterize rules from homogeneous platforms, node features in heterogeneous graphs come from distinct feature spaces. Given an interaction graph dataset $\{G\}$, the goal is to design a model $f_w : x^{\{G\}} \to \mathbb{R}^d$ to learn a d-dimensional representation of the graph G. The model f_w can be different learning tasks, such as drifting sample detection and binary classification. The learned graph representation preserves the structural and semantic information [19], which is associated with the interactive threat patterns.

3 SYSTEM DESIGN

In this section, we introduce the offline (back-end) design and online (front-end) usage of Glint. We focus on the four core parts of Glint: (i) We provide details of the interaction graph dataset construction process. (ii) We design the ITGNN model for graph representation learning. (iii) We present the drifting interaction pattern detection algorithm. (iv) We propose to enhance the generalization ability of ITGNN by transferring knowledge across heterogeneous platforms.

3.1 System Overview

Glint mainly has two stages for offline model training and interactive threat detection in real-time. Figure 2 shows the workflow of smart home automation data analysis. ①: In the **offline stage**, Glint analyzes possible intra-platform and inter-platform interactions to build interaction graphs, which compose a large interaction graph dataset stored in a database. ②: Next, we train the designed GNN model on the interaction graph dataset based on contrastive learning and classification model. ③: Using transfer learning, the well-trained model achieves higher generalization ability. Finally, we obtain the GNN-based interactive threat detector.

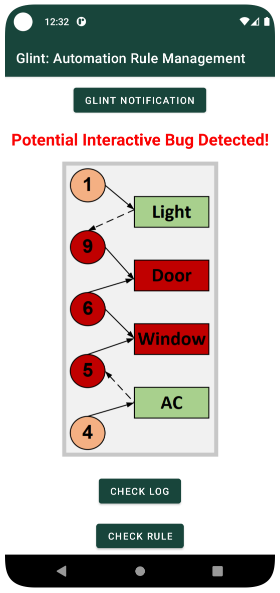
In the **online stage**, with the mined trigger-action logic from rules and device status from event logs, ④: interaction graph can be dynamically constructed in real-time. ⑤: The constructed graph is first tested by the contrastive learning-based drifting sample detector. Then, it is examined by an interactive threat detector (classification model). ⑥: If there is an interactive threat, Glint will generate a warning and request user intervention. ⑦: If a drifting sample is detected or there exist false-alarming graph cases that users do not regard as threats, we can update these special graph cases and ⑧: fine-tune the model to adapt to the users' needs.

A public GNN model is trained on graph datasets from heterogeneous platforms in the offline stage, which can be provided by a security solution provider via cloud services. Every user has the same offline component. Glint can be deployed on the smart home hub, and users only need to care about the online stage. Glint can help check the interactive threats in the initial setup phase of a smart home system. Moreover, with the existing deployment setup and event log information, the GNN model will be fine-tuned based on user preferences during the runtime of smart home systems. A Glint app can be installed on a phone to generate warnings and request user intervention.

Suppose a user deploys a set of rules as shown in Table 1. For simplicity, suppose Rules 2, 3, 7, and 8 are not executed during a specific time. As mentioned before, a real-time interaction graph is constructed by analyzing deployed rule descriptions and runtime event logs. Glint detects an interactive threat and notifies the user as shown in Figure 3a. In the interaction graph, the potential causes (i.e., nodes and related devices) of the threat are displayed in red in Figure 3a, so as to mitigate the information overload to users. The potential causes can be found with the aid of existing GNN explanation tools such as PGExplainer [26], SubgraphX [50], FexIoT [41] which identify important subgraphs or nodes accounting for the GNN prediction. Based on the daily appliance usage habits, the provided event logs in Figure 3b and potentially vulnerable rules in Figure 3c. If the rule interaction is undesired, the user can stop or update the rule configurations by reconfiguring the corresponding smart home platform apps.

3.2 Building Interaction Graph Dataset

We first build interaction graph datasets using rule descriptions in the offline stage. Then, we construct real-time interaction graphs with deployed rules and event logs in the online stage. We consider exploring IoT interactions in closed-source and cross-platform environments. The design of the NLP pipeline facilitates information access in the closed-source platforms.


12:32

Glint: Automation Rule Management

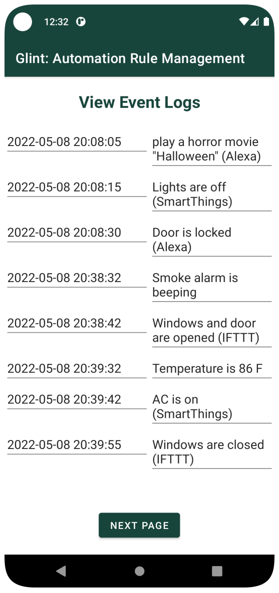
GLINT NOTIFICATION

Potential Interactive Bug Detected!

CHECK LOG

CHECK RULE

(a) Notification.


12:32

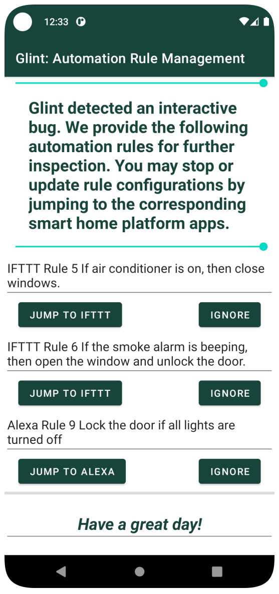
Glint: Automation Rule Management

View Event Logs

2022-05-08 20:08:05	play a horror movie "Halloween" (Alexa)
2022-05-08 20:08:15	Lights are off (SmartThings)
2022-05-08 20:08:30	Door is locked (Alexa)
2022-05-08 20:38:32	Smoke alarm is beeping
2022-05-08 20:38:42	Windows and door are opened (IFTTT)
2022-05-08 20:39:32	Temperature is 86 F
2022-05-08 20:39:42	AC is on (SmartThings)
2022-05-08 20:39:55	Windows are closed (IFTTT)

NEXT PAGE

(b) Event logs.


12:33

Glint: Automation Rule Management

Glint detected an interactive bug. We provide the following automation rules for further inspection. You may stop or update rule configurations by jumping to the corresponding smart home platform apps.

IFTTT Rule 5 If air conditioner is on, then close windows.

JUMP TO IFTTT IGNORE

IFTTT Rule 6 If the smoke alarm is beeping, then open the window and unlock the door.

JUMP TO IFTTT IGNORE

Alexa Rule 9 Lock the door if all lights are turned off

JUMP TO ALEXA IGNORE

Have a great day!

(c) Operation.

Figure 3: The usability demonstration of Glint.

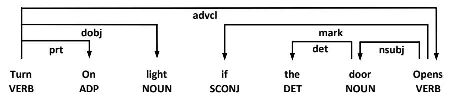


Figure 4: An example of dependency parsing.

3.2.1 *Rule Correlation Discovery.* The main challenge in building interaction graph datasets is to discover correlations among different rules. The large volume of noisy data with disparate formats hampers the discovery of correlations between rules across multiple platforms. Inspired by prior works [37, 38, 43], we use NLP techniques to extract semantic features from publicly available app descriptions. Considering a large number of possible combinations of "action-trigger" pairs, we model correlation discovery as a binary classification problem to reduce manual labeling efforts.

We design the smart home automation rule feature extraction algorithm as shown in Algorithm 1. The input is a set of trigger phrases and action phrases, and the output is correlation feature

Algorithm 1: Home Automation Rule Feature Extraction

```
Input: IoT Automation Rule Trigger Set $T_S$, Action Set $A_S$
Output: Rule Correlation Feature Vector $V$
1 foreach $T \in T_S$ and $A \in A_S$ do
2     $[nouns, verbs]_T = PoS(T)$
3     $[nouns, verbs]_A = PoS(A)$
4     $V1 = DTWDistance([nouns, verbs]_T, [nouns, verbs]_A)$
5     $V2 = binaryRelation([verbs]_T, [verbs]_A)$
6     $V3 = binaryRelation([nouns]_T, [nouns]_A)$
7     $V4 = \overline{E}_T + \overline{E}_A$    ▷ $\overline{E}_T$ and $\overline{E}_A$ are averaged word
        embeddings of a trigger and an action, respectively.
8 end
9 return $[V1, V2, V3, V4]$
```

vectors. We implement part-of-speech (POS) tagging and syntactic element extraction (lines 2-3). POS tagging follows context and word definition to recognize the main task, trigger object, and action object. For example, in Figure 4, the dependency parser recognizes the root verb *Turn* as the main task. The dependency edge connects other syntactic words to the root verb, which indicates a grammatical relation between the trigger object and the action object. We focus on direct objects (dobj), nominal subjects (nsubj), compounds and modifiers, nominal and passive nominal subjects, and clausal complements, based on which we extract the main task, objects, and their properties in one sentence. Moreover, we discard the named entity since a named entity may modify two different objects, which will reduce the uniqueness of features. For example, *Wyze Cam* and *Wyze Thermostat* are two different objects, but *Wyze* will add bias when computing the semantic similarity.

Based on the linguistic elements, we compute the numerical context features. We calculate the verb similarity and object similarity in the trigger-action pairs (line 4 in Algorithm 1). We apply dynamic time warping [30] to compute similarity because the number of verbs or objects in trigger and action sentences varies. Then, we compute the binary semantic features (lines 5-6). We analyze whether the verb or object words in trigger-action pairs have synonym or hypernym relations and whether the object words in trigger-action pairs have meronym (i.e., a constituent part of object) or holonym (i.e., a part of meronym name) relations. These relations can better reflect the generic semantic relationships between triggers and actions. Finally, we compute the trigger-action pair embedding (line 7) by summing the averaged word embeddings in corresponding triggers and actions. These are unique features of "trigger-action" rule pairs, which allow classifiers to learn the internal characteristics of interactions. We compose the above semantic features into feature vectors to train a model that detects correlations among "action-trigger" correlation pairs. In this way, we can build correlation pairs among different rules, which are formulated as a binary classification problem. If the given action can lead to the invocation of the trigger, it is labeled "true". Otherwise, it is labeled "false".

3.2.2 Offline and Online Graph Construction. In the **offline stage**, there are two phases to building interaction graphs. The input dataset is a set of crawled rule sentences, which provide rich "trigger-action" interaction logic. First, based on the rule correlation discovery model, we predict the remaining unlabeled rule pairs. In this way, we obtain "action-trigger" correlations between two rules. In the second phase, based on the correlation results, we randomly select and concatenate the "trigger-action" and "action-trigger" rule interactions to form interaction graphs, which makes generated graphs less prone to bias. We use the DGL [42] library to build and store the graphs, which are accessible with DGL APIs.

In the **online stage**, we consider the scenarios when one house is equipped with single or multiple platforms. With our designed offline rule correlation discovery classifier, we can first build a complete interaction graph from all rules deployed in a user's smart home, as shown in Table 1. However, it cannot reflect the actual interactions among smart devices in real time because such graphs lack the triggers' chronological ordering information. Fortunately, event logs contain three basic elements: time, object, and the current status of the object, which can reflect real-time device status as shown in Figure 3b. The event time helps determine the event sequences. Users can set up the device name and location when they deploy and configure devices. Thus, we obtain semantic information such as device types, locations, and device states to differentiate among different devices. Then, we match the device type and current status with the graph constructed from the deployed rules. In this way, we can remove the unrealistic "trigger-action" pairs such as the second rule (action) happening before the first rule (trigger). Thus, we can determine the unique **real-time interaction graph**.

The temporal dimension is not only integrated into the NLP embeddings but is also used for pruning graphs. First, we encode the temporal semantics (e.g., "sunset, at midnight") into embeddings, such that the ML models could learn the temporal features. Second, in the online stage, we use the triggers' chronological ordering information in event logs. For example, we can set a time interval (e.g., 3 hours) and use the timestamp to remove any unrealistic pairs. In this way, although two rules (nodes) could potentially interact, there is no edge between them in a real-time interaction graph because of the disjoined occurrence time. The temporal information allows the model to better learn "action-trigger" correlations, so as to reduce false alarms. Besides, with temporal information, we can remove unrealistic correlations and reduce the interaction graph edges, making it convenient for users to inspect device interactions.

3.3 Graph Representation Learning

In this section, we model the interactive threat analysis as a graph learning problem to automatically learn interaction patterns. Considering the different types of information linked to nodes, a single type of feature representation cannot represent the whole graph due to the differences in types and dimensionality. Existing GNN models [15, 19, 21, 45] either target homogeneous graphs or heterogeneous graphs with different designs. Besides, the interaction patterns could exist at different scales in a heterogeneous interaction graph, e.g., at the different subgraph levels. In this work, we design **ITGNN**, a unified model for **I**nteractive **T**hreat analysis based on **GNN**, which learns interactive threat patterns from different scales of an interaction graph.

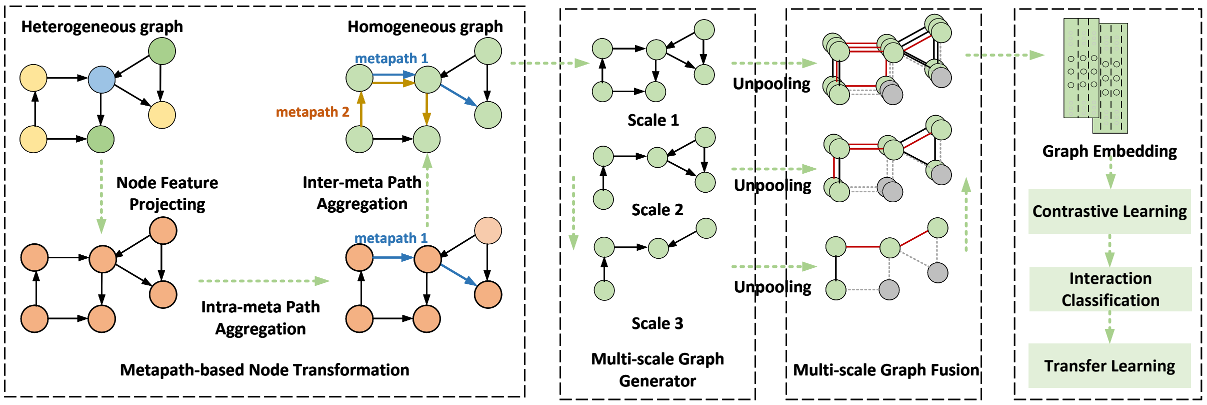


Figure 5: The proposed ITGNN model for interaction graph representation learning.

3.3.1 Graph Representation Model Design. We design the interaction graph representation learning process as shown in Figure 5, and the ITGNN model is detailed in Algorithm 2. Other input parameters in Algorithm 2 include node type set $\mathcal{A}$; the number of node V_A; node weight matrix W_A; node feature vector $x_v \in X$; metapath set $\mathcal{P}$; the number of layers L; the number of scales D; weight matrix for linear transformation of each metapath-specific node vectors M_A and b_A; activation function σ; aggregation function Φ. ITGNN takes as input a graph $G(V, E, X)$ and corresponding parameters, and outputs a d-dimensional graph representation z_g. ITGNN considers interaction features from both neighborhoods of a vertex and multiscale structures of a graph.

First, we project all nodes' features in heterogeneous graphs to the same feature space and then aggregate intra-metapath and inter-metapath information, called metapath-based node transformation. A metapath describes a composite relation between a series of node types, and the metapath instance is a sequence of nodes in a graph following the metapath schema [15]. Unlike homogeneous graph classification, we need to consider how to aggregate various types of nodes into the readout function. Inspired by MAGNN model [15], we first project heterogeneous node features to the same embedding space (line 3). For a node v of type A, we have the projected node embedding as h_v. Then, for each target node, we average the node features of metapath instances into a single vector (lines 5-7). In line 6, N_v^p is metapath-based neighborhoods of node v, $p(v, u_1)$ is a metapath instance, and v is the mapping target.

Second, we use the attention mechanism to aggregate the inter-metapath information (lines 9-11), where s_{p_i} is the summarized metapath of $p_i \in p_A$, q_A is the attention vector for node type A and β_{p_i} is the relative importance of each metapath p_i for the targeting node v. The interactive threats are usually caused by a subset of graph metapath instances. Different instances will contribute to the final graph embedding to different extents. Then, we transform the node embedding features while retaining the graph structure to form homogeneous-type graphs (line 13).

Third, we extract comprehensive features from multiple scales of a graph by incorporating the vertex infomax pooling (VIPool) module [24] into ITGNN, which is called a multi-scale graph generator and fusion. We input the homogeneous-type graph and apply the VIPool [24] to generate multi-scale graphs (lines 15-21), from which

Algorithm 2: ITGNN Graph Representation Learning

```
Input: interaction graph $G$ and other input parameters
Output: graph representation $z_g$
1  if $V_T > 1$ then
2      for $A \in \mathcal{A}$ do
3          $h_v = W_A \cdot X_v^A$
4          for $P \in \mathcal{P}_{\mathcal{A}}$ do
5              for $v \in V_A$ do
6                  $h_v^p = \frac{1}{|N_v^p|} \sum_{v \in N_v^p} (\frac{1}{|v|} \sum_{|v|} h_v), \forall v \in p(v, u_1)$
7              end
8          end
9          $s_{p_i} = \frac{1}{|V_A|} \sum_{v \in V_A} sigmoid(M_A \cdot h_v^{p_i} + b_A)$
10         $\beta_{p_i} = Softmax(q_A \cdot s_{p_i})$
11         $h_v^{p_A} = \sum_{p \in p_A} \beta_p \cdot h_v^p$
12     end
13     $G_m = G(V, E, \{h_v^{p_A}\})$
14 end
15 for $l = 1 \ldots L$ do
16     for $d = 1 \ldots D$ do
17         $h_v^d = VIPool(G_m)$
18         $h_v = h_v || h_v^d$
19     end
20     $h_v^{(L_i)} = \sigma(h_v^{(L_{i-1})}, \Phi(h_u^{(L_{i-1})}; u \in N_v))$
21 end
22 return $z_g = readout(h_v; v \in V)$
```

we extract the comprehensive features. The multi-scale graphs are propagated via the TAG [12] convolution layer, which does not need to approximate graph convolution. By vertex pooling, we select and keep vertices that contain high mutual information with neighborhoods, which can well express local subgraphs. We concatenate different scales of features as $h_v = h_v^0 || \cdots || h_v^{d-1} || h_v^d$ (line 18). h_v^d is the d_{th} scale of the graph features, $||$ represents concatenation. Fully connected layers are connected to fuse multi-scale features.

The proposed ITGNN graph representation learning can be used for different tasks. If the interaction graph is heterogeneous, it

Algorithm 3: Drifting Interaction Pattern Detection

```
Input: Training interaction graphs $g_i^{(j)}$, $i = \{0, 1\}$ is binary
       classes, $j = \{1, \cdots, n_i\}$, $n_i$ is the sample number in
       class $i$, testing data $x^{(m)}$, $m = \{1, \cdots, M\}$, $M$ is the
       total number of testing samples, ITGNN-C model $f$
Output: Drifting degree for each testing sample $A^{(k)}$
1  for $i \in \{0, 1\}$ do
2      for $j = 1$ to $n_i$ do
3          $z_i^{(j)} = f(g_i^{(j)})$   ▷ The latent representation of $g_i^{(j)}$
4      end
5      for $j = 1$ to $n_i$ do
6          $d_i^{(j)} = ||z_i^{(j)} - \frac{1}{n_i}\sum_{j=1}^{n_i} z_i^{(j)}||_2$
7      end
8      $MAD_i = median(|d_i^{(j)} - median(d_i^{(j)})|)$
9  end
10 for $m = 1$ to $M$ do
11     for $i \in \{0, 1\}$ do
12         $d_i^{(m)} = ||f(x^{(m)}) - \frac{1}{n_i}\sum_{j=1}^{n_i} z_i^{(j)}||_2$
13         $A_i^{(m)} = \frac{|d_i^m - median(d_i^{(j)})|}{MAD_i}$
14     end
15     return $min(A_i^{(k)}), i = 1, \cdots, N$
16 end
```

will first map node features from different feature spaces into the same space. Then, we can map the homogeneous-type graph into graph embedding. Next, we feed the generated graph embedding to the drifting sample detection module and classification module to identify the presence of an interactive threat. The concept drift is an important problem when modeling heterogeneous data from different IoT platforms. Since the classification models are mostly based on the i.i.d. assumption, the drifting samples should be filtered out. The detection and filtering of drifting samples could help reduce the false positives and false negatives of the classification model.

3.3.2 *Drifting Interaction Pattern Detection.*

Given that the interactive threat labeling is based on a set of known interactive threat patterns, it is conceivable that the learned model could have false negatives. This is because the testing interaction graph distribution may drift away from that of the training dataset due to the new smart device automation rules and the presence of various attacks. As a result, there will be a shift in the decision boundary [16] of the supervised or semi-supervised machine learning models, which will degrade the model performance.

Therefore, we need to detect the interaction graph samples that drift from existing classes. Here, we design ITGNN architecture with contrastive learning loss, called ITGNN-C, to learn a distance function to discover drifting samples. Taking advantage of our labeled graph dataset, contrastive learning can achieve better performance compared to unsupervised methods [48]. The basic idea of contrastive learning is to enlarge the distance among samples with different labels and reduce the distance among samples with the same label. Given a set of samples x_i and the corresponding label y_i, the contrastive loss takes a pair of samples (x_i, x_j) as input and tries to learn a function f_θ. The formula is written as follows:

$$\begin{aligned}\mathcal{L}_c(\mathbf{x}_i, \mathbf{x}_j, \theta) &= \mathcal{T}[y_i = y_j]||f_\theta(\mathbf{x}_i) - f_\theta(\mathbf{x}_j)||_2^2 \\ &+\mathcal{T}[y_i \neq y_j]\max(0, \epsilon - ||f_\theta(\mathbf{x}_i) - f_\theta(\mathbf{x}_j)||_2)^2,\end{aligned} \tag{1}$$

where $\mathcal{T}[y_i = y_j]$ means the value is 1 if two graph samples have the same label, otherwise is 0. $\mathcal{T}[y_i \neq y_j]$ means the value is 1 if two graph samples have different labels, otherwise is 0. ϵ is the upper bound distance to avoid the exceptional contribution of some dissimilar pairs. The ITGNN-C model augmented with contrastive loss can map samples from each class to a compact region in the latent space. After training the ITGNN-C model with labeled data, we can apply it to discover drifting samples. Given a list of testing samples x_t, with respect to the current classes in the training data, inspired by CADE [48], we evaluate if x_t is a drifting sample as shown in Algorithm 3.

We have binary classes "normal" and "threat", denoted as 0 and 1, respectively. We first generate the latent representations of training graph samples and compute the mean values of the representations as the centroid of each class (lines 2-4). We calculate the median of the absolute deviation MAD_i within each class i (lines 5-9). For testing, we generate representations of the testing samples and compute each test's drifting degree (lines 10-16). We recognize a potential drifting sample by comparing A^k with a threshold T_{MAD}, which is set as 3 empirically [23]. Users or security analysts can further investigate the drifting samples and conduct retraining.

3.3.3 *Vulnerable Interaction Classification.*

Besides designing contrastive learning, which learns to compare two samples, we design the supervised classifier loss function L for classifying graphs as normal or vulnerable:

$$\begin{aligned}L &= -\sum_{n-1}^{N} \frac{1}{\sum_{n=1}^{N} w_{y_n}} w_{y_n} x_{n,y_n} - \beta L_{pool}, \\ L_{pool} &= \frac{1}{n}\sum_i (t_i * \log(o_i) + (1 - t_i) * \log(1 - o_i)),\end{aligned} \tag{2}$$

where w_{y_n} is the weight of class labels to impose a high penalty when misclassifying the minority class. L_{pool} is graph pooling loss[24], β is a hyperparameter to balance the classification loss and pooling loss, x_{n,y_n} is the element of input embedding to the loss function, o_i is the logit output from each VIPool layer, and t_i is the binary label. In order to minimize false positives and false negatives, it is necessary to remove potential drifting samples and subsequently employ a classification model for detecting interactive threats. This approach can yield higher accuracy compared to solely relying on contrastive learning, as demonstrated in Table 5.

3.3.4 *Cross-Domain Graph Transfer Learning.*

Another challenge is that the semantic knowledge from different platforms varies. Some platforms contain less semantic knowledge, making it hard to train a robust GNN model. For example, the interaction rules of apps from the SmartThings platform are limited, with only 135 apps suitable for analysis. However, a key insight is that the apps' descriptions are mainly about the functions of device controls on the smart home platforms, which share common information. For example, the app on the SmartThings platform "*Turn on the air conditioner when the temperature is above 85 °F*" and the skill on the Alexa platform "*Alexa, turn on the air conditioner*" trigger a similar action. Interaction knowledge can be transferred across different smart home platforms as these platforms all target the same trigger-action paradigm. Therefore, we design a transfer learning module to transfer knowledge of interaction information across platforms.

Transfer learning has demonstrated its superiority in enhancing model performance and reducing the training time with less training data in computer vision [17, 49]. Deep learning models typically show excellent transferability properties. For example, the first layer features tend not to be specific to a particular task [49]. Similarly, the first few layers of embedding features of GNN contain some common information. The node features are both semantic-based embeddings. In addition, they share some properties that are transferable among smart home platforms. For example, the knowledge in the IFTTT interaction graph can be transferred to the SmartThings interaction graph. In turn, the learned features of the SmartThings graph model can be used to train a better IFTTT graph model. We can transfer embedding features from other pre-trained models to train a more robust model.

Formally, suppose D_S is source domain and D_T is target domain, where $D = \{X, P(X)\}$ and $X_S \neq X_T$. A domain feature space X is defined as a set of graph node features in a certain platform in the context of Glint. X_S is the feature space of the source domain, and X_T is the feature space of the target domain. The goal of transfer learning is to learn $P(Y_T|X_T)$ by gaining information from D_S and source task T_S. First, Glint trains a source domain model M_{T_S}. We take source domain graph node features as input to train M_{T_S}. Then, Glint transfers part of the layers of M_{T_S} to the target domain task model. Finally, Glint uses a small number of data in D_T to fine-tune parameters in the transferred layers and train parameters in new layers. The number of transferred layers depends on the specific graph model to yield better performance.

Glint uses a two-pronged approach for transfer learning. **First**, Glint transfers semantic knowledge from one platform to another, such as from the SmartThings interaction graph to the IFTTT interaction graph. There are common features in semantic knowledge from different platforms. Thus, Glint can transfer semantic knowledge in the form of feature embeddings. **Second**, Glint transfers interaction graph knowledge from homogeneous to heterogeneous interaction graph. Different platforms provide certain similar functions even though the usage is different. For example, an Alexa voice command or a SmartThings app can directly control the light status. The embeddings of GNN layers can transfer such interaction knowledge. Because the first several layers' features learned by GNNs are generic to be applied in many tasks [31], Glint can freeze the first several layers close to the input layer to preserve the transferred knowledge.

4 IMPLEMENTATION AND EVALUATION

We first evaluate the rule correlation discovery models for graph dataset construction. Then, we present and label the constructed interaction graphs. Next, for interactive threat analysis, we evaluate the performance of our designed ITGNN model on homogeneous and heterogeneous graph datasets. Then, we explore the possible interactive threats in user-designed automation rules. Finally, we evaluate Glint with a real-world testbed.

Table 2: The number of rules from 5 platforms.

IFTTT	SmartThings	Alexa Skill	Google Assistant	Home Assistant
316,928	185	5,506	5,292	574

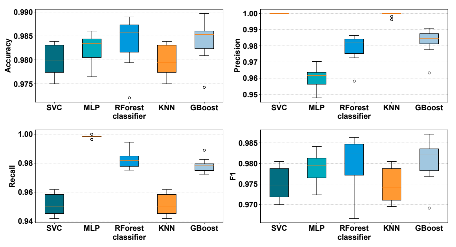


Figure 6: The performance of five classification models.

4.1 Rule Correlation Discovery Evaluation

We use Scrapy [33] to crawl publicly available rule descriptions from 5 smart home platforms, as shown in Table 2. We randomly choose and manually label 5,600 pairs that have action-trigger correlations and another 8,000 pairs that have no action-trigger correlations. Part of the labeled data is directly from [43], and we manually label the interaction pairs in newly crawled data. We use the labeled dataset to train classifiers to recognize whether a sentence pair has an action-trigger correlation. Then, we use the well-trained classifiers to label whether a sentence pair has action-trigger correlations for unlabeled rule pairs. Besides, we manually check the classification results to ensure the correctness of action-trigger correlations. In this way, we can efficiently filter out the impossible and unrealistic correlations among different rules.

We compare five different classification models: C-Support Vector (SVC), Multi-layer Perceptron (MLP), Random Forest (RForest), K-nearest Neighbors (KNN), and Gradient Boosting (GBoost). We use Scikit-learn [28] library to implement the experiments. The input of a model is the features of action-trigger pairs. The output is true or false. True means there is a correlation between "action" to "trigger". To deal with the issue of class imbalance, we adjust class weights inversely proportional to class frequencies in the training data. For instance, we use a weight balance in the MLP loss function to give a high cost for the misclassification of "true" flows. We apply the grid search method to find the most effective hyperparameters and use 10-fold cross-validation to test the generalization of models. We compute the accuracy, recall, precision, and F1 to evaluate the models. The recall R represents the percentage of positive samples in the test set which are predicted correctly. A high recall means a low false-negative rate. The precision P shows the percentage of positive predictions which are truly positive. The high precision means a low false-positive rate. The F1 score is the harmonic average of precision and recall: $F1 = \frac{2 \cdot P \cdot R}{P+R}$.

Can ML aid in rule correlation discovery? As shown in Figure 6, all five models achieve excellent correlation classification performance, demonstrating the utility of our extracted features. MLP and Random Forest achieve 98.2% and 98.4% accuracy, respectively. Both SVC and KNN achieve the highest average precision rates of 100%. The high precision ensures that the true connection predictions have a high possibility of being truly positive. MLP

Table 3: The number of interaction graph datasets

Type	Platforms	Label	Num. of Total Graph	Num. of Unsafe Graph
Homo.	IFTTT	labeled	6,000	1,473
		unlabeled	10,000	*
	SmartThings	labeled	165	36
Hetero.	5 platforms	labeled	12,758	3,828
		unlabeled	19,440	*

and Random Forest models achieve 99.8% and 98.2% recall, respectively. Besides, Random Forest achieves the highest 98% F1 scores. A higher F1 score for a classification model means better performance. With deep and large hidden layers, MLP can better characterize the hidden features in the dataset. The Random Forest model is based on bagged trees, but it uses a random subspace method to prevent overfitting. Therefore, we chose MLP, RandomForest, and KNN to collaboratively predict the remaining 20,000 unlabeled pairs based on the highest precision, recall, and F1 value, so we can uncover the interaction correlations between actions and triggers. If the predictions of the three models differ, we manually review the prediction results to determine the final correlation labeling result.

4.2 Interaction Graph Construction

Due to the large volume of rule combinations, we randomly select and chain different rules that formulate "trigger-action" correlations to build interaction graphs, of which the number of nodes is from 2 to 50. In an interaction graph, we build an edge between two rules that have a "trigger-action" correlation, and each rule is a node.

For the IFTTT interaction graph dataset construction, we use rules from both the dataset (315,393 applets) from [43] and our newly crawled 1,535 applets to construct interaction graphs among different applets. Considering descriptions could be lengthy and contain multiple sentences, we extract word phrases for text descriptions of IFTTT applets. Similarly, we build the SmartThings graph dataset by analyzing the apps' descriptions provided by the developers. For IFTTT and SmartThings homogeneous interaction graph datasets, we use the *en_core_web_lg* model in spaCy [35] to get the averaged word embeddings of each phrase in every rule. The dimension of embedding is 300, which is used as node features. As shown in Table 3, on the IFTTT platform, we build and manually check 6,000 interaction graphs to label whether graphs contain interactive threats or not. Finally, we label 1,473 graphs that contain interactive threats. Besides, we build 10,000 unlabeled IFTTT interaction graphs for discovering interactive threats in the smart home platform. On the SmartThings platform, we cross-check our 165 SmartThings inter-app interaction graphs with existing graphs [10] to ensure accuracy. We label 36 unsafe graphs out of 165 graphs.

We build a heterogeneous interaction graph dataset on IFTTT, SmartThings, Alexa, Google Assistant, and Home Assistant platforms by building interaction correlations across platforms. Specifically, we use IFTTT, SmartThings, and Alexa to build 12,758 heterogeneous interaction graphs with labels. We have heterogeneous graphs, among which 3,828 graphs are labeled as vulnerable graphs. For voice assistant rules, we use the *Universal Sentence Encoder* [8] to obtain sentence embeddings of rules, which are more suitable for identifying sentence features. The dimension of each embedding is 512, which is used as the node feature. Besides, we generate 19,440 unlabeled graphs based on the above five platforms. The Google Assistant and blueprints (rules) from Home Assistant are used in unlabeled graphs. These blueprints are created and discussed from December 2020 to January 2022.

We use DGL [42] to build the graph datasets. The stored labeled graph dataset file for IFTTT is 21.8G, the one for SmartThings is 0.018G, and the heterogeneous graph amounts to 81.6G. The large dataset file is partly due to the storage of all vertex information with DGL. We apply the DGL with Pytorch to build and train the GNN model. We run experiments on TensorEX Ubuntu 20.04 Deep Learning Stack with 256GB DDR4 memory, Intel(R) Xeon(R) Gold 5218R 2.10GHz CPUs and RTX A6000 GPUs.

In the labeling phase, two students who study IoT security volunteer to label interaction graphs. They first learn the security policies and IoT interactive threats identified in the literature [2, 7, 10, 11, 14, 39, 43]. For example, IoTSafe [11] specifies safety and security policies such as "All electrical appliances should be turned off when smoke is detected". Then, two volunteers follow the six types of threats [43] as criteria for labeling, where the specific rule settings are in Table 4:

(1) **Condition bypass.** Users may configure different granularity of settings with the same aim on different platforms. As a result, some conditions in fine-grained settings may be bypassed due to existing coarse-grained settings. For example, the users may have SmartThings setting 1 and have a concise setting 2 with Alexa. Such settings will allow condition bypass, which will bring threats.

(2) **Condition block.** With different smart home settings, the condition may be blocked because some commands may cancel the execution condition of another command. For instance, suppose we have settings 3, 4, and 5; when motion is detected at the door after 7 p.m., no notification is sent because the home state is disarmed.

(3) **Action revert.** The same device may be correlated to multiple environmental factors, and the action could be reverted if some environmental factors change. For example, with settings 6 and 7, the air conditioner is turned on and then turned off. Such interactions may cause action to revert.

(4) **Action conflict.** Some actions may conflict with each other in multiple smart home platforms. For instance, with settings 8 and 9, the action is conflicted if smoke is detected after 10 p.m.

(5) **Action loop.** The interactions among devices and the environment could trigger each other, which will introduce a loop in the interaction graph. For example, with settings 10 and 11, when the home state is away, the actions will loop forever.

(6) **Goal conflict.** There are cases when a user makes misconfigurations, such that the actions' goals conflict with each other. Suppose the user turns on the heater to keep a room warm while in the bathroom; however, given the settings 12 and 13, the window opens, resulting in the temperature drop of the room.

We develop the Python scripts to show the graph and rule contents, so the first volunteer can quickly check the graph edges and derive the conclusion. If a type of defined threat is found, then the volunteer stops checking and labels the graph as vulnerable. Otherwise, it is labeled as normal. Especially we cross-check our 165 SmartThings inter-app interaction graphs with the existing inter-app interaction chain graph of SmartThings [10] to ensure accuracy, from which we label 36 unsafe graphs out of 165 graphs.

Table 4: Rule examples from different platforms.

Platform Setting	Content
SmartThings Setting 1	If outside temperature is above 70°F and time is 11 a.m., then open windows.
Alexa Setting 2	If outside temperature is above 70°F, then open windows.
IFTTT Setting 3	If motion is detected at the door and home is in armed state, then send a notification.
IFTTT Setting 4	When light is on, disarm home state.
SmartThings Setting 5	Turn on the light at 7 p.m.
Alexa Setting 6	Turn on the air conditioner when temperature is above 100°F.
IFTTT Setting 7	When humidity is below 30%, turn on humidifier and turn off air conditioner.
SmartThings Setting 8	If smoke is detected, unlock the door.
Alexa Setting 9	Lock the door at 10 p.m. every day
IFTTT Setting 10	Turn off the living-room light when bedroom light is on.
IFTTT Setting 11	If the living-room light is turned off and the homestate is away, then turn on bedroom light.
Alexa Setting 12	Turn on a heater
SmartThings Setting 13	Open windows if indoor temperature is above 80°F.

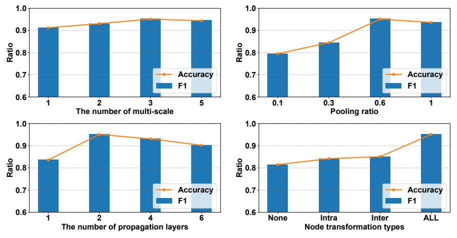


Figure 7: The ablation study of ITGNN model.

The other volunteer double-checks the labeling result to avoid mislabeling. If they have disagreements on the labels, they will have a further discussion based on the security policies identified in the literature. For large and complex interaction graphs, such as a graph with 50 nodes, it takes about 6 minutes to examine the possible interactive threats in a graph. For simple graphs with two nodes, we just need to check whether they violate the security policies, which takes about 20 seconds. On average, it takes about 1 minute to label each graph at the beginning. After the two volunteers get familiar with the labeling process, it takes about 40 seconds to label and double-check each graph. In total, it takes about 8 weeks to label the entire graph dataset.

4.3 Ablation Study of ITGNN

In this section, we implement an ablation study focusing on the ITGNN model. We evaluate the hyper-parameters and components of ITGNN on the heterogeneous dataset as shown in Figure 7. Specifically, we consider: (i) The number of scales, which is in the multi-scale graph generator. We can see that ITGNN achieves the best performance when the number of scales is 3 in Figure 7. When the number of scales is small (e.g., 1, original scale), ITGNN underestimates local information in graphs while global information is underestimated when the scale number is large (e.g., 5). (ii) Pooling ratio, which is graph pooling in VIPool. When the value is 1, the model keeps all vertices and yields VIPool ineffective. When the value is 0.6, ITGNN can select vertices that can better express neighborhoods' information, which helps achieve the best performance.

(iii) The number of propagation layers in the multi-scale graph generator. We test the influence of the number of layers in the propagation process. As shown in Figure 7, when the number of layers becomes large (e.g., 6), the model achieves worse performance compared with the number of layers being 2. The GNN model suffers from over-smoothing and becomes less expressive when the layer number increases. (iv) We also remove some modules in metapath-based node transformation. As we can see from Figure 7, when we remove both intra-meta path and inter-meta path aggregation modules, the model can only achieve an accuracy of 81.5%. This is because, for heterogeneous graphs, the model ignores the various types of node information in different metapath instances. As a result, the model cannot effectively learn the graph structure information in heterogeneous graphs. In contrast, the complete design of ITGNN achieves the best accuracy of 95.1%.

Table 5: Results of homogeneous graph classification

Dataset	Model	Accuracy	Precision	Recall	F1
IFTTT	GCN[21]	89.5	100	89.5	94.5
	GXN[24]	78.7	79.0	76.4	76.3
	GIN[46]	95	94.7	94	94.4
	IFG[36]	69.8	75.5	70.2	67.4
	SVC[29]	84.1	84.1	84	83.9
	KNN[3]	89.5	90.9	89.5	89.6
	ITGNN-C	95.4	95.3	94.9	95
	ITGNN-S	**95.7**	**95.9**	**95.7**	**95.8**
SmartThings	GCN[21]	90.9	82.6	90.9	86.6
	GXN[24]	88.2	89.9	88.2	87.2
	GIN[46]	89.7	85.9	89.5	87.7
	IFG[36]	86.1	89.3	87.5	85.9
	SVC[29]	84.4	87.3	84.8	81.3
	KNN[3]	84.8	83.8	84.8	83.2
	ITGNN-C	76.5	69	70.6	69.5
	ITGNN-S	**88.2**	**89.9**	**88.2**	**87.2**

4.4 Homogeneous Graph Evaluation

With the interaction graph datasets, we can train the GNN-based interactive threats detection model. We run experiments in 5 trials. For each trial, we split the dataset into the training set and testing set by 8:2. However, there are only 1,473 vulnerable interaction graphs out of 6,000 interaction graphs, as shown in Table 3. The imbalanced classes will degrade the performance of the model. Thus, for the training set, we first increase the number of vulnerable graphs by random oversampling until the number of vulnerable graphs is doubled. Second, we assign unequal costs to the misclassification class. The class weight is inversely proportional to the number of class examples. We use the weighted F1 score to measure the performance of graph models. We calculate the F1 value for each label and then compute the average weights by the number of occurrences of each class. Thus, the F1 value may be beyond the range of precision and recall.

We compare our proposed model ITGNN with four state-of-the-art graph models and two classification models, including: (i) graph convolutional network (GCN) [21], which is a convolutional network operating on graphs; (ii) graph cross-network (GXN) [24] model, which extracts graph features by learning from multiple scales of a graph; (iii) GIN [46] model, a graph isomorphism network, which is used to distinguish certain graph structures; (iv) InfoGraph (IFG) [36], which uses graph-level representation learning via mutual information maximization. (v) Two classical ML methods: SVM and KNN. We compute the average of the node embeddings as the graph sample feature. (vi) ITGNN-C uses the contrastive learning loss in Eq. (1) and ITGNN-S uses the classification loss in Eq. (2) to differentiate between vulnerable and normal interaction graphs. We train the eight models and fine-tune hyperparameters on interaction graph datasets.

Why not use traditional ML models? As shown in Table 5, on the IFTTT dataset, the two traditional classifiers, SVC and KNN, achieve 84.1% and 89.5% accuracy, respectively. Nevertheless, the graph models can achieve better performance. For example, compared to KNN, GIN improves accuracy by 5.5% to 95%, and ITGNN-S improves accuracy by 6.2% to 95.7%. Traditional ML models cannot mine the graph structure information, which is essential for threat analysis, while GNN is designed to learn graph information, and it can better mine the interactive threat patterns.

Why not use the contrastive loss for classification? Compared with models trained on the IFTTT dataset, models trained on the SmartThings dataset have worse performance. For instance, ITGNN-C on the SmartThings dataset only achieves 76.5% accuracy because contrastive learning needs a large number of data to learn to compare two samples. However, the size of the SmartThings dataset (165 graphs) is much smaller than that of the IFTTT dataset.

4.5 Heterogeneous Graph Evaluation

We evaluate the performance of our proposed ITGNN model on the heterogeneous graph dataset. We compare ITGNN with three state-of-the-art heterogeneous graph learning methods. (i) HGSL [52] performs heterogeneous graph structure learning for classification. The MAGNN [15] is designed for heterogeneous node classification and link prediction. Due to the different downstream tasks in the original models, we construct two heterogeneous graph models. (ii) MAGCN model is an adapted GCN [21] model with a MAGNN graph converter. (iii) MAGXN model is then adapted GXN [24] model with a MAGNN graph converter.

Why not use existing GNN models? With the evaluations on both homogeneous and heterogeneous graph datasets, we show the benefits of ITGNN as a unified framework for both homogeneous and heterogeneous graph representation learning. Besides, ITGNN is designed for both drifting sample detection and interactive threat classification problems. On the IFTTT dataset, as shown in Table 5, ITGNN achieves the best accuracy, recall, and F1 score. Specifically, ITGNN-S achieves 95.7% recall and F1 score on the IFTTT dataset. Achieving a better recall is necessary because it reduces false negatives and will avoid safety and security problems by better predicting possible interactive threats. Even though GIN performs better on the SmartThings dataset, it achieves lower precision than ITGNN-S. Other complex GNN architectures entangle the whole interaction graph information, which are not suitable for our interactive threat detection task.

On the heterogeneous graph dataset, as shown in Figure 8, our proposed ITGNN model can achieve the best average accuracy

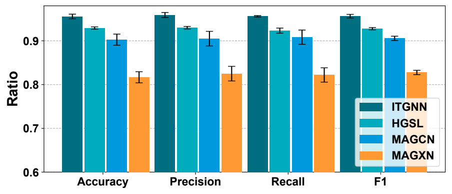


Figure 8: The results of heterogeneous graph classification.

(95.5%), precision (95.9%), recall (95.6%), and F1 score (95.6%). The HGSL achieves 92.9% accuracy, the MAGCN achieves 90.2% accuracy, while MAGXN achieves 81.7% accuracy for threat detection. The MAGXN has a more complex architecture with more parameters, resulting in a slow training process. By contrast, the MAGCN achieves acceptable performance even though it has a simpler structure, which confirms the "No free lunch" theorem [44] for machine learning. Overall, the proposed ITGNN outperforms both the homogeneous and heterogeneous models. ITGNN is highly effective since it can not only preserve heterogeneous node feature information but also leverages the multi-scale graph structure features to generate graph representations.

4.6 Transfer Learning Evaluation

From previous results in Table 5, we find that the simple GCN architecture can be easily trained, while other models such as GXN and ITGNN have relatively complex architecture, which can be overfitting on an insufficient dataset (e.g., SmartThings). This indicates that one model may best perform for some datasets, but not in all cases. To apply Glint in a more diverse type of heterogeneous smart home platforms, we need to enhance the adaptability of GNN models. Transfer learning applies the learned knowledge from one domain to a different but related domain, improving modeling performance with less training data and time.

As shown in Table 6, we not only evaluate the transfer learning between homogeneous (IFTTT)-homogeneous (SmartThings) datasets but we also evaluate homogeneous (IFTTT)-heterogeneous datasets. We take IFTTT and SmartThings platforms as examples of homogeneous graph learning because these two are prevalent platforms in IoT interaction studies. Besides, the IFTTT dataset contains a large number of graphs, while SmartThings contains a small number of graphs, which are representative datasets for transfer learning problems. The size of the SmartThings dataset is much smaller than the IFTTT dataset. Therefore, given the IFTTT dataset as the source domain and the SmartThings dataset as the target domain, we only fine-tune the fully connected layer for classification and freeze the other layers of a model trained on the IFTTT dataset. Then, we let the SmartThings dataset be the source domain and the IFTTT dataset be the target domain; now, we can freeze the earlier two layers of the model trained on the SmartThings dataset. The earlier layers capture the basic feature of interaction graphs. Finally, we can train the rest of the layers using the IFTTT dataset.

Can transfer learning improve model performance? The performance of all models improves with transfer learning techniques as shown in Table 6. For example, the accuracy of the GIN model on the SmartThings dataset improves by 6%. One interesting phenomenon is that if a model performs well in the source

Table 6: Detailed performance of transfer learning.

Model	Target Domain	Source Domain	No trans.	Trans.	Improved
GIN	SmartThings	IFTTT	89.7%	92.3%	2.6%
GIN	IFTTT	SmartThings	95.0%	95.2%	0.2%
GCN	SmartThings	IFTTT	90.9%	94.1%	3.2%
GCN	IFTTT	SmartThings	89.5%	93.9%	4.4%
ITGNN	SmartThings	IFTTT	88.2%	100%	11.8%
ITGNN	IFTTT	SmartThings	95.7%	96.4%	0.7%
ITGNN	IFTTT	Heterogeneous	95.7%	96.1%	0.4%
ITGNN	Heterogeneous	IFTTT	95.1%	95.5%	0.4%

domain, it will also work well in the target domain. For instance, the ITGNN model achieves 88.2% accuracy on the target domain (SmartThings). With knowledge of the source domain (IFTTT), the model can achieve 100% accuracy. Moreover, even though the source domain (SmartThings) knowledge is limited, it can still provide helpful knowledge across platforms. For example, the accuracy on the IFTTT platform grows from 95.7% to 96.4%. Moreover, we check if the transfer learning techniques could bring negative effects. As shown in Table 6, the applied transfer learning techniques consistently enhance the model by achieving better accuracy in the target domain. Therefore, models trained on datasets from multiple smart home platforms do not incur negative transfer problems.

4.7 Drifting Interaction Pattern Evaluation

We implement the Algorithm 3, and we use principal component analysis (PCA) to project the graph embeddings from the original 256-dimensional into a 2-dimensional space. We use the K-means method to cluster the projected embeddings as shown in Figure 9, where the centroid of each class is the white cross.

We test the samples in the unlabeled dataset to detect the potential drifting samples (the samples in the red circle in Figure 9). The drifting samples are examined manually to summarize the potential interaction patterns and can be used for retraining. Among 10,000 unlabeled IFTTT graphs, we found 63 potential drifting samples. For 19,440 unlabeled heterogeneous graphs, we found 104 potential drifting samples. For example, the open and close of a lawn sprinkler valve is a rare example, and only exists in the unlabeled dataset, so it is regarded as a potential drifting sample. The ratios of vulnerable interaction graphs on the randomly generated unlabeled IFTTT dataset and heterogeneous dataset are 8.3% and 16.7%, respectively. To recognize whether there are new threat cases, we manually examine all drifting samples and randomly check 50% predicted vulnerable interaction graphs. Primarily, in drifting samples, we discover four new types of interactive threats related to user-designed blueprints (rules) [5] across multiple platforms. The reported new interactive threat types are more complex and easily overlooked, while the previously discovered threat types from existing work are all rooted in a single platform.

• **Action block**: Automation is blocked by non-automation settings. Users create automation blocker rules that block some automation from running. For example, users can set rule 1, "If the light is set in manual mode, then keep the light brightness to 100%.". Thus, rule 2, "dimming lights when turning on the TV" will become ineffective. Rules 1 and 2 are encoded into two nodes in an interaction graph, and they target the same device, "light", which will be labeled as a vulnerable interaction because of the block of the action.

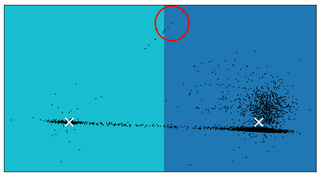


Figure 9: K-means clustering on graph embeddings learned with contrastive loss (PCA-reduced data).

• **Action ablation**: Automation action can be reverted over time. A device can have multiple attributes that are triggered by different factors. For example, a blueprint turns on the air conditioner (AC) when the temperature is above 95°F. Another blueprint has the rule that turns on the humidifier and turns off the AC when humidity is below 30%. The status of the AC between the two rules is conflicted and will be reverted when the humidity becomes low.

• **Trigger intake**: Automation is caused by unexpected triggers. A user applies a rule to send the camera a snapshot notification when motion is detected at the door. There is another rule that starts the vacuum cleaner at 9 pm, which could accidentally trigger the motion sensor. As a result, the user may frequently receive false snapshot notifications. Without a deep analysis, the user can hardly notice this is caused by the event that the vacuum cleaner triggers the motion sensor, especially in a bad light condition.

• **Condition duplicate**: A fake automation condition is generated by rules from other platforms. For instance, a rule that reports the room is occupied when any of the conditions are met: motion sensor detects motion, the door is shut, or media is playing on devices in the room. A rule from IFTTT can play music in the room from 3 pm to 4 pm. Due to the trigger from the played music, the room is reported to be occupied. Another rule that starts the heating when the room is occupied and when the temperature is below 60°F is subsequently triggered. Because of the applet from IFTTT, the rule condition is accidentally satisfied in the platform of Blueprint.

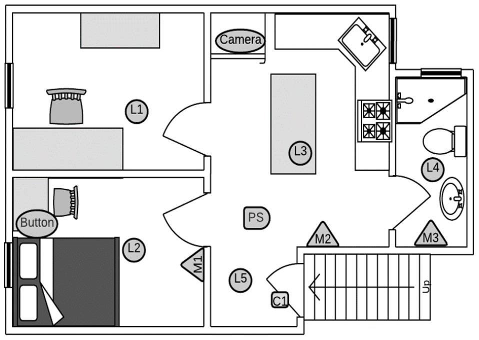
Note that action block involves user intervention, and action ablation happens over a long time. They are neither action revert nor action conflict defined in iRuler [43]. Besides, existing work such as HAWatcher [14] cannot handle these user interventions and long-term correlations. Similarly, trigger intake and condition duplicates are not pre-defined in iRuler or mined in HAWatcher. As a result, the existing work cannot detect these new types of interactive threats.

4.8 Evaluation on a Real-life Testbed

We construct real-time interaction graphs with event logs collected from a real-world home testbed, as shown in Figure 10. A volunteer deploys the SmartThings and Alexa systems in his house following the setup in HAWatcher [14] with off-the-shelf devices and apps. The volunteer uses the desired automation and spends a week collecting 1,813 event logs related to home automation.

4.8.1 *Efficacy Comparison.*

Following the setup in HAWatcher [14], we simulate five types of attacks by modifying event logs or manually interfering with the home automation: (i) Targeted compromise: fake commands, stealthy commands. (ii) Interaction abuse: fake events, event losses. (iii) Misconfiguration: command failure. For



Abbr.	Device Name	Attributes
L	Light Bulb	switch
M	Motion Sensor	motion
C	Contact Sensor	contact
T	Temperature Sensor	Temperature
PS	Presence Sensor	presence
Camera	Camera	snapshot
Button	Smart Button	button

Figure 10: The layout of smart home devices.

example, we simulate "fake commands" by manually turning off lights during normal operation, and "stealthy commands" by manually starting a robot vacuum to trigger motion sensors. We build 600 graphs as the test set. Out of 300 graphs, 150 graphs contain binary-correlation threats (BCT). Among another 300 graphs, 150 graphs contain complex-correlation threats (CCT). BCT means interactive threats are caused by two nodes, while CCT is caused by more than two nodes.

We choose three white-box anomaly detection methods for comparison: **(i)** HAWatcher [14], which extracts binary correlations and verifies them with run-time event logs. The input is the device states extracted from event logs. The inconsistencies are reported as anomalies. We reuse the refined correlations reported in the paper [14] to check the actual states of devices. **(ii)** One-class Support Vector Machine (OCSVM) [32], which is a unsupervised anomaly detection method. **(iii)** IsolationForest [25], which is also widely used for anomaly detection. We capture all devices' states as a frame when a new event happens. Four consecutive frames compose a data vector, which is the input of OCSVM and IsolationForest. The output is -1 for threats and 1 for normal cases. We use Scikit-learn [28] to implement OCSVM and IsolationForest. Note that for many other systems, it is unfair to directly compare the performance of existing methods and ours when considering the big gap between the analysis of open-source and closed-source platforms.

As shown in Figure 11, OCSVM and IsolationForest achieve worse performance compared to Glint. For instance, for complex graphs, the OCSVM can only achieve 66.9% precision and 63.3% recall. This is because they only use time-series event log data, without the consideration of rich information in neighbor nodes and graph structure. The HAWatcher can achieve 97.8% precision and 94.1% recall taking 21 days of training for binary-correlation threat detection [14]. By contrast, Glint takes no more than 1 hour to train the model and apply transfer learning to improve model performance,

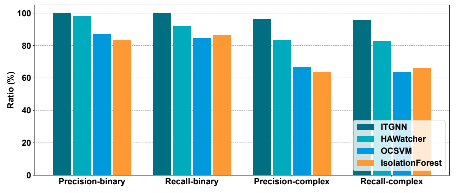


Figure 11: Comparison with different detection methods.

which achieves 100% precision and recall for binary-correlation threat detection. For complex-correlation graphs, HAWatcher cannot check interactive threats caused by goal conflict, action revert, and condition bypass because the three types of threats are not covered. For such cases, we set HAWatcher to generate results by the Bernoulli distribution with the probability of a single success of 0.5. Finally, HAWatcher achieves 83.2% precision and 82.7% recall. By contrast, Glint achieves 96% precision and 95.3% recall, which outperforms HAWatcher in heterogeneous platform scenarios by 12.8% precision and 12.6% recall rate.

4.8.2 *Efficiency Comparison.*

Given that time efficiency is affected by many factors, such as code implementation, third-party libraries, and hardware setup, we qualitatively compare the efficiency of different systems. HAWatcher [14] needs to traverse all defined correlations. Suppose the number of correlations is n, then the time complexity is n^2 for their extracted binary correlations. The time complexity will be n^N when the length of the interaction chain is N. Thus, such search-based methods are retarded by the path explosion when dealing with large and complex graphs. iRuler [43] applies satisfiability modulo theories (SMT) solver to check interactive vulnerabilities. However, the existence of heterogeneous platforms indicates that there will be more complex rules for smart home management. With the fact that the SMT problem is typically NP-hard, such symbolic execution has poor scalability when dealing with large and complex interaction graphs. It is thus unlikely that a symbolic execution engine can exhaustively explore all the possible states within a reasonable amount of time. By contrast, our proposed Glint is a learning-based system, which achieves a high efficiency for real-time interactive threat prediction. For each graph, the prediction time is related to the graph size, and the average prediction for a heterogeneous graph takes about 0.61s. The ITGNN model trained on heterogeneous graphs is only 6.13 MB.

4.8.3 *Strengths and Limitations.*

While the above evaluation results are encouraging, we consider this work as a first step toward data-driven IoT interaction analysis. We discuss the strengths and limitations of Glint with examples.

Time factor. The IoT device influence can occur in the long term. For example, turning on/off the heater will change the temperature over a relatively long period. The existing system, HAWatcher, only considers interactions that arise within a short time frame, which cannot mine long-term correlations. By contrast, Glint discovers interactions based on automation rule semantics and real-time event logs. We encode the time-related semantics into node embeddings. One strength of Glint is that by annotating such long-term correlations in graph datasets, the GNN models can effectively learn such patterns manifested over a long time interval. For example, settings

12 and 13 in Table 4 compose a vulnerable interaction graph, which is reported by Glint, but the HAWatcher generates a false negative.

User factor. The automatic interactions could be disturbed by user-activity deviations. An example is shown in the drifting interaction pattern "Action block" in Section 4.7. Existing systems such as iRuler [43] use pre-defined vulnerability patterns, which cannot report such interactive threats. In comparison, Glint can detect abnormal patterns with concept drift detection. According to users' feedback, Glint can fine-tune the model and adapt to user preferences. How regularly we should update the interaction graphs to accommodate changes in user actions and IoT devices poses an important future research problem.

NLP techniques. We acknowledge that the current NLP techniques applied in this work may overestimate or underestimate physical channel properties, especially when involving different locations. The physical properties of two devices at different locations can vary. For instance, the temperature of the oven in the kitchen can hardly influence the temperature in the living room. As a result, one limitation of Glint is its lack of complete understanding of the physical channel properties, which may lead to wrong predictions about the correlation between the two rules. For example, "Set temperature of the oven to 350°F for preheating at 7:00 pm" should have no correlation to "Open the window if the indoor temperature is above 80°F". The construction of an interaction graph will affect the GNN prediction result. We may request users' feedback on the correctness of the built graphs, so we can fine-tune the graph builder model. Besides, more advanced NLP techniques such as large language models [27, 53] can be applied to enhance the accuracy of IoT rule correlation recognition in the future.

5 RELATED WORK

Many studies have focused on interactions among apps in smart homes. Most systems rely on the source code of apps [2, 7, 11, 39]. For example, IoTGuard [7] applies code instrumentation to collect runtime information and checks against pre-defined security policies. However, most platforms such as IFTTT and Alexa are closed-source, which makes these methods inapplicable. Instead, Glint fuses NLP-based multi-domain information without relying on code analysis.

Other systems [7, 11] use dynamic testing to discover interactive threats. For instance, IoTSafe [11] combines static analysis and dynamic testing to predict risky conditions. However, the deployment of devices in real-life costs a lot. They need to carefully control their test cases to avoid causing unexpected safety issues. Moreover, it is well-known that dynamic testing suffers from code coverage issues. Many factors, including the time and seasonal impacts, are not considered. HAWatcher [14] extracts normal correlation patterns from smart apps and event logs, but their method fails to deal with long-term correlations and user activity deviations. Glint mines interaction correlations based on NLP techniques by fusing rule semantics and real-time event logs. Glint extracts interactive threat patterns with GNN models, which can avoid the overhead of dynamic testing.

Furthermore, many systems [7, 14, 43] need to pre-define the security policies or threat patterns. For example, the rule parser in iRuler [43] only considers interactions within a single platform. iRuler needs expert knowledge to accurately pre-define complex threat space for the rule-based model checker. However, it is unknown how to pre-define diverse types of interactive threats across heterogeneous closed-source platforms. Therefore, the rule-based methods can only detect well-known interactive threats, which causes high false negatives. Moreover, iRuler needs to consider every combination of device states. Meanwhile, the definition of rules, device status, and the search depth setting can influence both the accuracy and efficiency of the analysis. For instance, a larger search depth may lead to a more time-consuming analysis.

6 CONCLUSION

To detect interactive threats across heterogeneous closed-source platforms, we propose the graph learning-based system Glint to learn interaction patterns. We design the unified ITGNN model for multi-scale graph representation learning. To cope with the interaction pattern coverage issue, we propose the drifting interaction pattern detection algorithm. In addition, we leverage transfer learning to adapt to different data volumes in heterogeneous platforms. To train ITGNN, we build the first interaction graph dataset on five high-profile platforms. Our evaluation shows that the proposed model can achieve high accuracy in detecting interactive threats across heterogeneous platforms. We identify four new types of interactive threats in the user-defined Home Assistant platform. In the future, we will continue exploring the explainability of GNN models to better understand the root causes of interactive threats. In conclusion, our proposed system, Glint, is capable of effectively detecting interactive threats in IoT apps, potentially inspiring novel solutions for IoT data management.

ACKNOWLEDGEMENT

We would like to thank the anonymous reviewers for their insightful comments on our work. This work was supported in part by National Science Foundation grants CNS-1950171 and CCF-2200621 and Cisco Research Funding for building AI-driven smart home IoT platforms.

REFERENCES

[1] Hadi Abdullah, Washington Garcia, Christian Peeters, Patrick Traynor, Kevin RB Butler, and Joseph Wilson. 2019. Practical hidden voice attacks against speech and speaker recognition systems. *arXiv preprint arXiv:1904.05734* (2019).

[2] Mohannad Alhanahnah, Clay Stevens, Bocheng Chen, Qiben Yan, and Hamid Bagheri. 2022. IoTCOM: Dissecting Interaction Threats in IoT Systems. *IEEE Transactions on Software Engineering* (2022).

[3] Naomi S Altman. 1992. An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician* 46, 3 (1992), 175–185.

[4] Amazon. 2022. Amazon Alexa Skills. https://www.amazon.com/alexa-skills/b?ie=UTF8&node=13727921011.

[5] Home Assistant. 2022. Home Assistant. https://www.home-assistant.io/.

[6] Steven Bird, Ewan Klein, and Edward Loper. 2009. *Natural language processing with Python: analyzing text with the natural language toolkit.* " O'Reilly Media.".

[7] Z Berkay Celik, Gang Tan, and Patrick D McDaniel. 2019. IoTGuard: Dynamic Enforcement of Security and Safety Policy in Commodity IoT.. In *NDSS*.

[8] Daniel Cer, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, et al. 2018. Universal sentence encoder. *arXiv preprint arXiv:1803.11175* (2018).

[9] Haotian Chi, Chenglong Fu, Qiang Zeng, and Xiaojiang Du. 2022. Delay Wreaks Havoc on Your Smart Home: Delay-based: Automation Interference Attacks. In *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, 1575–1575.

[10] Wenbo Ding and Hongxin Hu. 2018. On the safety of iot device physical interaction control. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 832–846.

[11] Wenbo Ding, Hongxin Hu, and Long Cheng. 2021. IOTSAFE: Enforcing Safety and Security Policy with Real IoT Physical Interaction Discovery. In *Proceedings of the 2021 Network and Distributed Systems Security (NDSS) Symposium*.

[12] Jian Du, Shanghang Zhang, Guanhang Wu, José MF Moura, and Soummya Kar. 2017. Topology adaptive graph convolutional networks. *arXiv preprint arXiv:1710.10370* (2017).

[13] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. 2017. Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*. 1285–1298.

[14] Chenglong Fu, Qiang Zeng, and Xiaojiang Du. 2021. Hawatcher: Semantics-aware anomaly detection for appified smart homes. In *30th USENIX Security Symposium*.

[15] Xinyu Fu, Jiani Zhang, Ziqiao Meng, and Irwin King. 2020. Magnn: Metapath aggregated graph neural network for heterogeneous graph embedding. In *Proceedings of The Web Conference 2020*. 2331–2341.

[16] João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. 2014. A survey on concept drift adaptation. *ACM computing surveys (CSUR)* 46, 4 (2014), 1–37.

[17] Xiang Gu, Jian Sun, and Zongben Xu. 2020. Spherical space domain adaptation with robust pseudo-label loss. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 9101–9110.

[18] Justin Huang and Maya Cakmak. 2015. Supporting mental model accuracy in trigger-action programming. In *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*. 215–225.

[19] Xiao Huang, Qingquan Song, Fan Yang, and Xia Hu. 2019. Large-scale heterogeneous feature embedding. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 33. 3878–3885.

[20] IFTTT. 2022. IFTTT Applets. https://ifttt.com/explore/.

[21] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).

[22] Palanivel A Kodeswaran, Ravi Kokku, Sayandeep Sen, and Mudhakar Srivatsa. 2016. Idea: A system for efficient failure management in smart iot environments. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*. 43–56.

[23] Christophe Leys, Christophe Ley, Olivier Klein, Philippe Bernard, and Laurent Licata. 2013. Detecting outliers: Do not use standard deviation around the mean, use absolute deviation around the median. *Journal of experimental social psychology* 49, 4 (2013), 764–766.

[24] Maosen Li, Siheng Chen, Ya Zhang, and Ivor Tsang. 2020. Graph cross networks with vertex infomax pooling. *Advances in Neural Information Processing Systems* 33 (2020), 14093–14105.

[25] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. 2008. Isolation forest. In *2008 eighth ieee international conference on data mining*. IEEE, 413–422.

[26] Dongsheng Luo, Wei Cheng, Dongkuan Xu, Wenchao Yu, Bo Zong, Haifeng Chen, and Xiang Zhang. 2020. Parameterized explainer for graph neural network. *Advances in neural information processing systems* 33 (2020), 19620–19631.

[27] OpenAI et al. 2023. GPT-4 Technical Report. *arXiv preprint arXiv:2303.08774* (2023).

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

[29] John Platt et al. 1999. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. *Advances in large margin classifiers* 10, 3 (1999), 61–74.

[30] Thomas Prätzlich, Jonathan Driedger, and Meinard Müller. 2016. Memory-restricted multiscale dynamic time warping. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 569–573.

[31] Luana Ruiz, Luiz Chamon, and Alejandro Ribeiro. 2020. Graphon neural networks and the transferability of graph neural networks. *Advances in Neural Information Processing Systems* 33 (2020).

[32] Bernhard Schölkopf, John C Platt, John Shawe-Taylor, Alex J Smola, and Robert C Williamson. 2001. Estimating the support of a high-dimensional distribution. *Neural computation* 13, 7 (2001), 1443–1471.

[33] Scrapy. 2022. Web Crawling Framework. https://scrapy.org/.

[34] SmartThings. 2022. SmartThings Developer. https://smartthings.developer.samsung.com/docs/api-ref/capabilities.html.

[35] Spacy. 2022. Industrial-Strength Natural Language Processing. https://spacy.io/.

[36] Fan-Yun Sun, Jordon Hoffman, Vikas Verma, and Jian Tang. 2020. InfoGraph: Unsupervised and Semi-supervised Graph-Level Representation Learning via Mutual Information Maximization. In *International Conference on Learning Representations*.

[37] Milijana Surbatovich, Jassim Aljuraidan, Lujo Bauer, Anupam Das, and Limin Jia. 2017. Some recipes can do more than spoil your appetite: Analyzing the security and privacy risks of ifttt recipes. In *Proceedings of the 26th International Conference on World Wide Web*. 1501–1510.

[38] Yuan Tian, Nan Zhang, Yueh-Hsun Lin, XiaoFeng Wang, Blase Ur, Xianzheng Guo, and Patrick Tague. 2017. Smartauth: User-centered authorization for the internet of things. In *26th USENIX Security Symposium (USENIX Security 17)*.

[39] Rahmadi Trimananda, Seyed Amir Hossein Aqajari, Jason Chuang, Brian Demsky, Guoqing Harry Xu, and Shan Lu. 2020. Understanding and automatically detecting conflicting interactions between smart home IoT applications. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1215–1227.

[40] Blase Ur, Melwyn Pak Yong Ho, Stephen Brawner, Jiyun Lee, Sarah Mennicken, Noah Picard, Diane Schulze, and Michael L Littman. 2016. Trigger-action programming in the wild: An analysis of 200,000 ifttt recipes. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*. 3227–3231.

[41] Guangjing Wang, Hanqing Guo, Anran Li, Xiaorui Liu, and Qiben Yan. 2023. Federated IoT Interaction Vulnerability Analysis. In *2023 IEEE 39th International Conference on Data Engineering (ICDE)*. IEEE.

[42] Minjie Wang, Da Zheng, Zihao Ye, Quan Gan, Mufei Li, Xiang Song, Jinjing Zhou, Chao Ma, Lingfan Yu, Yu Gai, Tianjun Xiao, Tong He, George Karypis, Jinyang Li, and Zheng Zhang. 2019. Deep Graph Library: A Graph-Centric, Highly-Performant Package for Graph Neural Networks. *arXiv preprint arXiv:1909.01315* (2019).

[43] Qi Wang, Pubali Datta, Wei Yang, Si Liu, Adam Bates, and Carl A Gunter. 2019. Charting the attack surface of trigger-action IoT platforms. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 1439–1453.

[44] David H Wolpert. 1996. The lack of a priori distinctions between learning algorithms. *Neural computation* 8, 7 (1996), 1341–1390.

[45] Feng Xia, Ke Sun, Shuo Yu, Abdul Aziz, Liangtian Wan, Shirui Pan, and Huan Liu. 2021. Graph learning: A survey. *IEEE Transactions on Artificial Intelligence* 2, 2 (2021), 109–127.

[46] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2018. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826* (2018).

[47] Lin Yang, Junjie Chen, Zan Wang, Weijing Wang, Jiajun Jiang, Xuyuan Dong, and Wenbin Zhang. 2021. Semi-supervised log-based anomaly detection via probabilistic label estimation. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 1448–1460.

[48] Limin Yang, Wenbo Guo, Qingying Hao, Arridhana Ciptadi, Ali Ahmadzadeh, Xinyu Xing, and Gang Wang. 2021. {CADE}: Detecting and explaining concept drift samples for security applications. In *30th USENIX Security Symposium (USENIX Security 21)*. 2327–2344.

[49] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. 2014. How transferable are features in deep neural networks? *arXiv preprint arXiv:1411.1792* (2014).

[50] Hao Yuan, Haiyang Yu, Jie Wang, Kang Li, and Shuiwang Ji. 2021. On explainability of graph neural networks via subgraph explorations. In *International Conference on Machine Learning*. PMLR, 12241–12252.

[51] Guoming Zhang, Chen Yan, Xiaoyu Ji, Tianchen Zhang, Taimin Zhang, and Wenyuan Xu. 2017. Dolphinattack: Inaudible voice commands. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 103–117.

[52] Jianan Zhao, Xiao Wang, Chuan Shi, Binbin Hu, Guojie Song, and Yanfang Ye. 2021. Heterogeneous graph structure learning for graph neural networks. In *35th AAAI Conference on Artificial Intelligence (AAAI)*.

[53] Ce Zhou, Qian Li, Chen Li, Jun Yu, Yixin Liu, Guangjing Wang, Kai Zhang, Cheng Ji, Qiben Yan, Lifang He, et al. 2023. A comprehensive survey on pretrained foundation models: A history from bert to chatgpt. *arXiv preprint arXiv:2302.09419* (2023).

[54] Pengpeng Zhou, Yang Wang, Zhenyu Li, Xin Wang, Gareth Tyson, and Gaogang Xie. 2020. Logsayer: Log pattern-driven cloud component anomaly diagnosis with machine learning. In *2020 IEEE/ACM 28th International Symposium on Quality of Service (IWQoS)*. IEEE, 1–10.