How to Cover up Anomalous Accesses to Electronic Health Records

Xiaojun Xu
UIUC
xiaojun3@illinois.edu

Qingying Hao
UIUC
qhao2@illinois.edu

Zhuolin Yang
UIUC
zhuolin5@illinois.edu

Bo Li
UIUC
lbo@illinois.edu

David Liebovitz
Northwestern University
david.liebovitz@nm.org

Gang Wang
UIUC
gangw@illinois.edu

Carl Gunter
UIUC
cgunter@illinois.edu

Abstract

Illegitimate access detection systems in hospital logs perform post hoc detection instead of runtime access restriction to allow widespread access in emergencies. We study the effectiveness of adversarial machine learning strategies against such detection systems on a large-scale dataset consisting of a year of access logs at a major hospital. We study a range of graph-based anomaly detection systems, including heuristic-based and Graph Neural Network (GNN)-based models. We find that *evasion attacks*, in which covering accesses (that is, accesses made to disguise a *target access*) are injected during evaluation period of the target access, can successfully fool the detection system. We also show that such evasion attacks can transfer among different detection algorithms. On the other hand, we find that *poisoning attacks*, in which adversaries inject covering accesses during the training phase of the model, do not effectively mislead the trained detection system unless the attacker is given unrealistic capabilities such as injecting over 10,000 accesses or imposing a high weight on the covering accesses in the training algorithm. To examine the generalizability of the results, we also apply our attack against a state-of-the-art detection model on the LANL network lateral movement dataset, and observe similar conclusions.

1 Introduction

With the development of computer technology and recent incentives for its use in healthcare, the health records in hospitals have transitioned to electronic systems [19]. The electronic health record (EHR) system provides users (e.g., physicians, nurses, and pharmacists) with comprehensive chart access to prior encounters (i.e., visits of a patient to the hospital) and facilitates better information sharing among different departments.

One major security concern of EHR systems is the inappropriate internal user access to patient encounter information. For example, when famous actors visits a hospital, some users may access their information to satisfy curiosity or even sell information for economic gain [17]. It is not appropriate to block such accesses beforehand, because rapid access to medical information is essential in emergencies and therefore all the users are allowed to access any record without restriction. Therefore, the primary approach is to perform **post hoc analysis** to detect and penalize such anomalous accesses after they have been made.

Considering the large number of accesses made in a hospital, it is desired to automatically detect illegitimate accesses in the system. In the past, researchers have worked on EHR anomaly detection and found that Machine learning (ML) techniques can achieve a good performance in such tasks [30, 37, 43]. Commerical companies (e.g. Imprivata [1]) are also known to apply ML-based techniques in EHR management. More recently, researchers also show that graph neural networks (GNN) are highly effective in detecting insider threats and illegal accesses within a networked system [23].

Given the high-performance illegitimate access detection system, in this paper, we focus on the robustness of the system and ask: *is the system still reliable if a knowledgeable attacker intentionally injects accesses to bypass the detection?* We assume an adversary that wants to make a (malicious) target access which, without adversarial actions, will be detected by the system. We propose two types of adversarial attacks, which are based on different access constraints of attackers, as shown in Figure 1. In an **evasion attack** (orange arrows), the detection model is trained on the clean training dataset without interruption. The attacker will inject covering accesses to the testing dataset so that all the injected edges look benign to the model. In a **poisoning attack** (blue arrows), the attacker tampers with the training phase by poisoning the training set with covering accesses. The detection model trained on the poisoned dataset will view the target access as a benign one on the clean testing set.

We evaluate both attacks against different detection algorithms on our EHR dataset. For the evasion attack, we propose an iterative algorithm based on the detection model gradient. We show that for 60% of randomly chosen target accesses, the

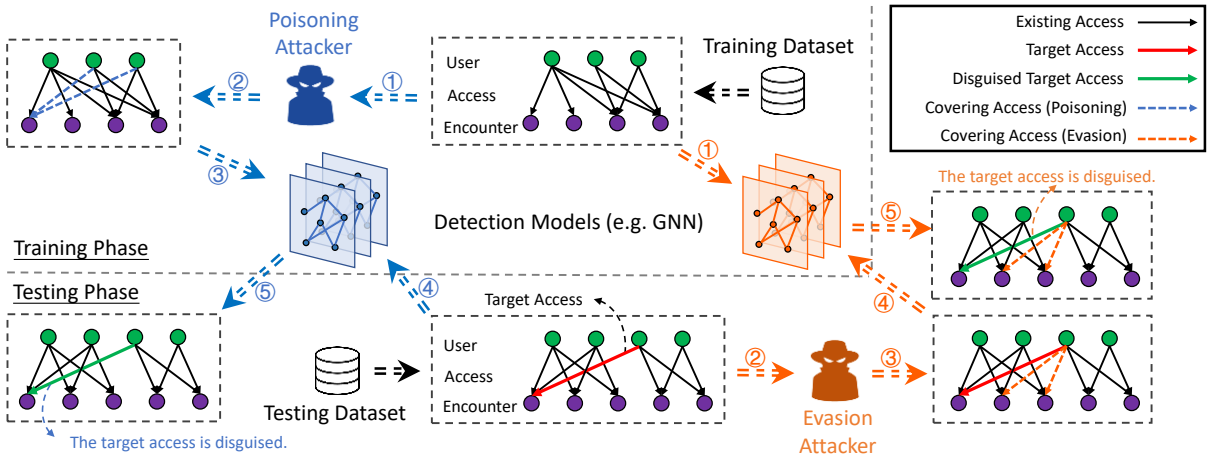

Figure 1: **Attack Overview**—The pipeline of our attacks against the illegitimate access detection system on the access log dataset. A 'user' refers to someone working in the hospital and an 'encounter' refers to a visit to the hospital made by a patient. The goal of the attacker is to disguise a target access in the testing phase. The double dashed arrows refer to the steps of the pipeline. In a poisoning attack (blue steps), the attacker will interfere with the training phase; in an evasion attack (orange steps), the attacker will interfere with the testing phase.

access can be disguised if the adversarial user makes 10 covering accesses to some encounter records in the system in a one-week period. In addition, the attack success rate increases to 77% with 50 covering accesses in a week. By comparison, a heuristic-based baseline attack cannot achieve a good attack performance.

For the poisoning attack, we propose a feature sensitivity-based algorithm to inject covering accesses. Surprisingly, we observe that even with strong knowledge and capability, the attacker is not able to perform an effective attack in usual cases. We show that the attack succeeds only under unrealistic settings, e.g., the user making 10,000 accesses within one week or changing the setting in the training process. Such a conclusion, i.e., the evasion attack is easier than the poisoning attack, differs from previous studies on other intrusion detection scenarios [36, 38]. We owe it to the reason that the data is highly dynamic and the nodes (patients) in the evaluation stage often does not appear in the training stage, so the poisoning attack cannot work well.

To examine the generalizability of the results, we also evaluate our attacks on a public dataset related to network lateral movement within an organization from the Los Alamos National Laboratory (LANL) [21]. We apply our attack against a state-of-the-art detection model Euler [23] which uses GNNs and recurrent neural networks (RNNs) for illegal access detection. We observe that our evasion attack works even better on this model: by injecting only 10 covering edges, it is sufficient to cover each of the malicious events. In addition, we confirm that poisoning is still more difficult than the evasion attack, even though poisoning works slightly better than that on the EHR dataset. Compared to the EHR graph, the LANL graph is less dynamic, i.e., the nodes in the graph (servers) appear in both the training and evaluation stages. The improved poisoning performance in turn confirms our earlier hypothesis, that is, the highly-dynamic property of EHR system makes it more difficult to launch poisoning attacks.

Our major contributions can be summarized as follows:

- This work proposes the first adversarial attacks against ML models for EHR illegal access detection and takes the stealthiness of the adversarial perturbation into consideration. We will open-source our code for the detection models and attacks.
- We propose evasion attacks against the detection system and show their effectiveness against the illegitimate access detection system for EHR.
- We show that it is difficult to launch the poisoning attack without unrealistic capabilities, which differs from previous conclusions on other intrusion detection systems.
- We also evaluate our attack against a state-of-the-art detection model outside of the EHR context (i.e., the LANL dataset on network lateral movement) and observe a similar conclusion.

2 Background

Anomaly Access Detection in EHR systems. In electronic health record systems, a user is not supposed to access arbitrary encounters for curiosity or other personal reasons. However, in medical systems it is problematic to restrict a user from accessing certain encounters. For example, if a pediatric nurse has to (for any reason) access the record of a geriatric patient in an emergency, the restriction may lead to a dangerous situation. Hence, the primary approach ([9, 18, 34, 43]) is to do post hoc detection, i.e. let the users access any encounter when needed, but audit whether these accesses are legitimate after they are made.

Graph Neural Network. Graph Neural Networks [45] are a class of neural networks that take attributed graph data as input and generate a feature vector for each node. Formally speaking, let $G = (V, E, X)$ denote an attributed graph where V is the node set, $E \subseteq V \times V$ is the edge set and $X \in \mathbb{R}^{|V| \times d}$ is the node feature matrix with d representing the feature dimension of each node. The edge features are usually not considered since they are less important than node features in most cases. The graph neural network $f(G) \in \mathbb{R}^{|V| \times h}$ takes the graph as input and calculates an h-dimensional embedding vector for each node using the graph structure and the input features. These node embeddings will be further processed according to different tasks. For example, on a node classification task one applies a linear layer over $f(G)$ to determine the class of each node; on a graph classification task the node features are aggregated (e.g., by taking the average) and sent through a linear layer to determine the class of the graph.

Adversarial Attacks on Machine Learning. In an adversarial attack, an attacker aims to tamper with the machine learning pipeline to produce some desired wrong outputs. In general, there are two classes of adversarial attacks. In a *poisoning attack*, the attacker interferes with the training phase. For example, [39] proposes to inject malicious training data which is similar to target input but an adversarial label to mislead the model. In an *evasion attack*, the attacker will interfere with the evaluation phase. A popular strategy [14] is to find and add to the input some imperceptible noise that mostly changes the model prediction via a gradient-based optimization technique.

3 Dataset Introduction

In our study, we will use the electronic health record dataset collected from a major hospital. The dataset consists of four major components and their relevant information as follows:

- **Patients**. A patient is a person who has received care from the hospital. The dataset includes patient information such as patient demographics (such as age and sex), diagnoses, and medications.
- **Encounters**. An encounter is a visit to the hospital made by a patient. The dataset includes encounter information such as its type (e.g., an inpatient), length of stay, and start and end date.
- **Users**. A user is a person working in the hospital (e.g., physicians and pharmacists) who provides services to the patients. The dataset includes user information such as the role and department.
- **Accesses**. An access is a logged event describing one user accessing the charts associated with one encounter. The dataset includes access information such as the access reason and date.

The dataset consists of access logs in the hospital over a period of one year. There are 309,096 patients, 944,385 encounters, 11,591 users, and 26,992,636 accesses in which a user visits the chart of a patient as part of an encounter. The hospital staff have performed standard data pre-processing procedures such as addressing malformed or missing entries. We view the dataset and its pre-processing to be representative of hospital EHR systems.

Our study focuses on in-patient encounters (i.e., records of patients who stay in a hospital while under treatment). Not surprisingly, inpatient encounters received the majority of the accesses—about 80.3% of all accesses were made to the inpatient encounters. We simulate the real-world scenario by extracting the data as 50 splits, where each split corresponds to the access logs in one week. We did not use the data at the beginning or end because less data is collected during these periods. The exact dates can not be revealed under the Safe Harbor policy. Within each period, if a user has accessed the same encounter more than once, we will only consider the access that appears for the first time. For example, if a user accessed an encounter twice in one split, the second access will not be included in the dataset. This is a simplifying assumption: if a user has a right to access the encounter once, then the user almost always has the right to access it a second time. On average, each split has 5,309 users, 2,916 encounters and 82,498 accesses.

We model an access log dataset as a bipartite graph where each user or encounter represents a node and each access is an edge connecting a user and an encounter. The patient information will come together with the encounter. We use $G = (V, E, X)$ to denote the graph, where $V = V_{user} \cup V_{enc}$ denotes the user node set and encounter node set respectively, E is the edge set where each edge $(u, v) \in E$, $u \in V_{user}$, $v \in V_{enc}$ connects one user node and one encounter node. $X \in \mathbb{R}^{|V| \times d_v}$ is the node feature matrix where d_v is the feature dimension of each node.

We extract the following features for a user: (1) User's department (udept): Each user belongs to 1 out of 196 departments. (2) User's role (urole): Each user belongs to 1 out of 172 roles.

We extract the following features for an encounter: (1) Length of stay (elos). We use numerical bins with 1,2,7,14,28,56 days as split values. (2) Patient's age (page). We use numerical bins with 3,10,20,30,40,50, 60,70 years old as split values. (3) Patient's gender (pgender). The gender is one out of 'Male', 'Female' and 'Unknown'. (4) Patient's diagnosis information (pdiag). This includes the ICD9 codes for diagnoses, procedures, medications during the period, and all the diagnoses of the patient in the history. There are 31,119 different types of diagnosis in total.

All these features are processed into a one-hot vector, except for the diagnosis information where we use multi-hot vectors because each encounter may have multiple diagnoses. We concatenate these one/multi-hot vectors together as the

feature vector for a node. This generates a 368-dimensional feature vector for a user and a 31,138-dimensional feature vector for an encounter.

4 Attack Motivation & Threat Model

In this section, we will first introduce the problem setup including current detection systems. Then we introduce the motivation and threat model for the proposed attacks.

4.1 Problem Setup

Given that graph-structured data is widely used in different applications such as recommendations in social networks and intrusion detection based on access patterns, there are different detection models set up for detecting abnormal graph patterns. In general, a detection model will take as input an untrusted graph and identifies a set of abnormal edges via different optimization strategies and prior knowledge. In particular, given a graph $G = (V, E, X)$, the detection model will be a function s such that given G and an edge $(u, v) \in V$, the model produces a score on whether the edge is benign or abnormal, i.e. $s(G, u, v) \in \mathbb{R}$. Usually, the score is calculated with the adjacency matrix in a differentiable manner. In other words, let A denote the adjacency matrix of graph G, then $s(G, u, v)$ can also be written as $s(A, X, u, v)$ and $\partial s/\partial A$ exists. Most popular edge classification algorithms satisfy this setting, such as Katz index, matrix factorization or graph auto-encoder [32]. In a special *poisoning attack* (which we will introduce later), we will also assume that the model first calculates a representation vector $g(G)_u$ for each node u, and then calculates $s(G, u, v)$ using simple operation (e.g. linear combination) based on $g(G)_u$ and $g(G)_v$. This is adopted in most ML-based models like matrix factorization, graph auto-encoder and Node2Vec [16]. In this work, we do not consider any other assumptions on the detection model.

4.2 Attack Motivation

Our goal as an adversarial user can be formalized as: given a malicious *target access* we would like to make, we will find a set of *covering accesses* so that 1) after injecting these accesses, the target access will seem benign to the system and 2) none of the covering accesses will be detected by the system. The attack is motivated by recent works [6, 14, 29] showing that machine learning systems will be fooled by adversarial attackers despite the high performance on the tasks. Hence, we want to act as an attacker who wants to make a malicious target access and evade the detection, and study the vulnerability of the illegitimate access detection system in hospitals. To fool the system, an attacker may change information about users, encounters, patients or accesses in the system, or add/delete accesses in the system. For a user in the hospital, the most applicable way is to make (i.e. add) extra accesses in the system.

Difference with existing Works. Our work is different from existing adversarial attacks against graph neural networks [7, 42, 47]. To evaluate the stealthiness of the attack, previous works manually set a budget on how many edges they can change. As long as the perturbation is within a certain budget, they view the attack as within small perturbations. In our work, we propose the attack with semantically meaningful perturbations against anomaly edge detection models - we require that *all* of the injected covering accesses should also be stealthy to the model.

As shown in Figure 1, we investigate two types of attack scenarios - evasion attack and poisoning attack. We show in detail how the covering accesses are injected in both attacks as in Figure 2:

A1: Evasion attack. In an evasion attack, the training process is already performed and the model is going to be applied to the testing graph to detect abnormal accesses on it. As an attacker, we would like to inject covering accesses into the testing graph to disguise our target access on the graph. For example, an adversary as a pediatric nurse would like to access the encounter of an elder woman out of curiosity, which is apparently very weird and will be easily detected as abnormal by the model. Therefore, the pediatric nurse can first access some pregnant women, which seems legitimate in common sense. After several accesses, it will be less suspicious for the adversary to access an elder woman based on her previous access patterns to various middle-aged women.

A2: Poisoning attack. In a poisoning attack, the model is not trained yet. The attacker will tamper with the training process by injecting covering accesses into the training graph so that the trained model will recognize our target access as benign in the testing graph. For example, if an adversarial pediatric nurse would like to access an elder woman during test time, she will find and inject some legitimate access from pediatric nurses to elder women in the training graph. After the model is trained, it will learn the (wrong) access pattern "pediatric nurses will access elder women" and view the target access as benign.

Note that these two attacks may appear together (e.g. the attacker injects covering edges in both training and testing graphs) or be mixed (e.g. certain ML algorithm trains and evaluates on the same graph) in practice. We will study the separate effect of each attack to see which one will be the more dangerous threat to the system.

4.3 Threat Model

We have the following assumptions about the attack:

Detection pipeline: We consider the inductive learning setting for the detection. That is, the model trainer will collect the data in a time period 1 and train the model. Then the

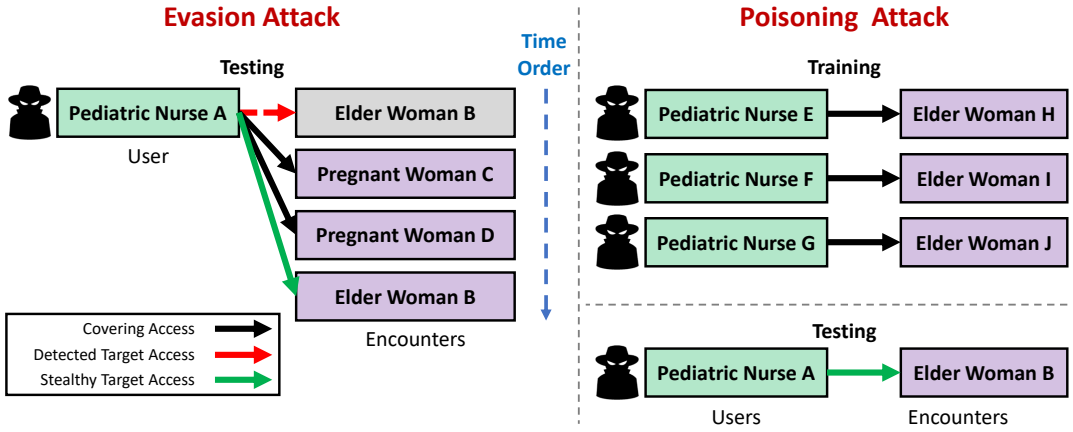

Figure 2: **Examples of the attack scenarios**—Black arrows refer to the covering accesses; the red arrow refers to the target accesses detected as malicious; the shaded encounter is rejected by the system and will not exist in the dataset; green arrows refer to the target accesses classified as benign ones. In an evasion attack, the attacker will inject covering accesses during testing time, so that the target access which may originally be detected by the system will look benign; in a poisoning attack, the attacker will inject covering accesses during training time so that the model trained on the data will view the target access as benign during testing time.

trainer collects the data in a time period 2 and applies the trained model to detect abnormal accesses. The training and evaluation data will not overlap in time. As we will show in Sec 7.4, it achieves better performance than transductive setting where training and evaluation data overlap.

Attacker's capability: With a *normal attacker capability*, the attacker (as the target user) is able to make access from the target user to any encounter; with a *strong attacker capability*, the attacker is able to make accesses from any user to any encounter. Such strong capability is not realistic, and we will only use it to illustrate the difficulty of poisoning attacks.

Attacker's knowledge: We have two dimensions of knowledge in the attack: the knowledge of the model and the knowledge of the data. We assume that the attacker always has white-box access to the model, including the structure, parameters and training algorithm. However, the attacker may have full (white-box) knowledge or partial (gray-box) knowledge of the data on which we train and evaluate the detection model. We made these assumptions because the model-level knowledge has been well studied [8, 28], and there are ways to get white-box model information via only black-box accesses [31]. On the other hand, the gray-box knowledge of data has received less attention.

5 Evasion Attack

In this section, we introduce our evasion attack algorithm. The main idea of the evasion attack is to iteratively add edges that can make the target edge look more "benign" to the model while keeping the added edges stealthy.

5.1 Attack Goal

In an evasion attack, we have a trained model f with related score function s applied to detect illegitimate access on a test dataset $G = (V, E, X)$. The attacker aims to make an illegitimate target access (u^*, v^*), but the access will be easily detected by the model if it is directly injected, i.e.,

$$s((V, E \cup \{(u^*, v^*)\}, X), u^*, v^*) < \theta$$

Therefore, the attacker's goal is to make a number of covering accesses $E_{cov} = \{(u_i^c, v_i^c)\}_{i=1}^K$, so that the target access will be viewed as benign after the covering accesses are made. In addition, we require that all the covering accesses should also seem benign and not be detected by the model:

$$s((V, E \cup \{(u^*, v^*)\} \cup E_{cov}, X), u^*, v^*) > \theta$$
$$s((V, E \cup \{(u^*, v^*)\} \cup E_{cov}, X), u_i^c, v_i^c) > \theta \quad \forall (u_i^c, v_i^c) \in E_{cov}$$

5.2 Attack Approach

We first assume a white-box knowledge of the data and a strong attacker capability. Our strategy of evasion attack is to find the covering accesses in an iterative manner, which is similar to the approach proposed in [7] but further requires stealthiness of covering accesses. Let $E_{cov}^t = \{(u_i^c, v_i^c)\}_{i=1}^t$ be the covering accesses that we already find in the t-th step and we would like to find the next covering edge (u_{t+1}^c, v_{t+1}^c). Let $s^t = s((V, E \cup \{(u^*, v^*)\} \cup E_{cov}^t, X), u, v)$ denote the current score of the target access and A^t be the adjacency matrix of $E \cup \{(u^*, v^*)\} \cup E_{cov}^t$. Note that we assume that the score is differentiable w.r.t. the adjacency matrix. We denote:

$$\nabla^t = \frac{\partial s^t}{\partial A^t}.$$

Algorithm 1: Evasion atk

Input: Detection model s, Test graph $G = (V, E, X)$ Target access (u^*, v^*), number of covering edges K

Output: E_{cov}: a set of K covering edges.

$E_{cov} \leftarrow []$;
for $t = 1, \ldots, K$ **do**
  $A^t \leftarrow \mathcal{A}(V, E \cup E_{cov})$;
  $s^t \leftarrow s(A^t, X, u^*, v^*)$;
  $\nabla^t \leftarrow \frac{\partial s^t}{\partial A^t}$;
  Calculate u_t, v_t as in Eqn. 1;
  $E_{cov} \leftarrow E_{cov} \cup \{(u_t, v_t)\}$;
return E_{cov}

Intuitively, a large value at ∇^t_{ij} means that adding the pair (i, j) will increase the prediction score of s^t, thus making the edge "more benign" and should be added as the covering edge. On the other hand, our choice is restricted so that the added edge should not exist in the original graph, should satisfy the bipartite graph constraint and should seem benign to the model. Hence, the next covering edge to add (u^c_{t+1}, v^c_{t+1}) is found by:

$$\begin{aligned} & \arg\max_{u,v} \nabla^t_{uv} \qquad (1) \\ \text{s.t.} \quad & (u, v) \notin E \cup \{(u^*, v^*)\} \cup E^t_{cov}, \\ & u \in V_{user}, v \in V_{enc} \\ & s((V, E \cup \{(u^*, v^*)\} \cup E^t_{cov}, X), u, v) > \theta \end{aligned}$$

We will repeat this process K times to find the covering access set. The algorithm is shown in Alg. 1.

In an attack with only gray-box knowledge of the graph, the attacker only knows part of the graph G'. To achieve the attack goal, the attacker will perform the evasion attack algorithm on G' and add the resulting covering edges into G. In an attack with normal attack capability, the accesses should only start from the target user, so we always require that $u = u^*$ in Eqn. 1.

6 Poisoning Attack

In this section, we introduce our poisoning attack algorithm. The idea of the attack is to inject the edges that are most similar to the target edge so that the model will learn such patterns as benign.

6.1 Attack Goal

In the poisoning attack, the attacker will interfere with the training dataset so that the trained model is misled and produce a wrong result on the chosen target access during evaluation, while the overall performance (on non-target accesses) remains similar. Formally speaking, suppose the model f is trained on $G_1 = (V_1, E_1, X_1)$ and evaluated on $G_2 = (V_2, E_2, X_2)$. The attacker aims to make a target access (u^*, v^*) but will be detected by the model, i.e.:

$$s((V_2, E_2 \cup \{(u^*, v^*)\}), u^*, v^*) < \theta$$

Unlike in an evasion attack, the strategy here is to inject some covering accesses in the training data as $G'_1 = (V_1, E_1 \cup E_{cov}, X_1)$, so that the trained model s' wrongly classifies the target access as benign:

$$s'((V_2, E_2 \cup \{(u^*, v^*)\}), u^*, v^*) > \theta$$

Note that this setting is different from many "poisoning attack" setting in other works on graph tasks, which assumes a transductive learning setting in which the covering accesses exist in both training and testing periods [42, 47, 48]. Our setting is usually more difficult because the evaluation graph remains clean and does not contain any covering accesses.

Algorithm 2: Poisoning atk

Input: Train graph $G_1 = (V_1, E_1, X_1)$, Test graph $G_2 = (V_2, E_2, X_2)$, Target access (u^*, v^*), number of covering edges K, maximum edge per node N

Output: E_{cov}: a set of K covering edges.

$E_{cov} \leftarrow []$;
Train the detection model on G_1 and get the encoding model g;
$n \leftarrow \{\}$;
foreach $u \in V_1$ **do**
  $n[u] \leftarrow 0$;
for $t = 1, \ldots, K$ **do**
  $E_{candidate} \leftarrow \{(u \in V_{user}, v \in V_{enc}) \mid n[u] < N \wedge n[v] < N \wedge (u, v) \notin E_1 \cup E_{cov}\}$;
  $u_t, v_t \leftarrow \arg\min_{(u,v) \in E_{candidate}} d(u, v, G_1; u^*, v^*, G_2)$;
  $E_{cov} \leftarrow E_{cov} \cup \{(u_t, v_t)\}$;
  $n[u] \leftarrow n[u] + 1$;
  $n[v] \leftarrow n[v] + 1$;
return E_{cov}

6.2 Attack Approach

We assume a strong attacker capability. Moreover, we do not require stealthiness on the covering access and we assume that the attacker already has full knowledge of the test data G_2 when the poisoning data is being injected in the training data G_1. These strong capabilities are clearly unrealistic — the idea is that if the attacker fails the poisoning attack even under such strong assumptions, there would be no need to test normal attacker capabilities.

The idea of the poisoning attack is to inject the training data which has the most similar representation vector with the

target victim data [39]. In our case, we want to add the covering access whose user and encounter has similarly encoded node representations as those of the target user and encounter. Formally speaking, consider a (trained) representation model $g(G)$, the attacker wants to inject covering accesses (u_i^c, v_i^c) into G_1 if the following value is small:

$$d(u_i^c, v_i^c, G_1; u^*, v^*, G_2) = ||g(G_1)_{u_i^c} - g(G_2)_{u^*}||_2^2 + ||g(G_1)_{v_i^c} - g(G_2)_{v^*}||_2^2$$

where $g(G)_u$ represents the representation of node u calculated by model g in graph G.

Based on this intuition, our poisoning attack algorithm is shown in Alg. 2. We will first train a model using the clean training data G_1 and use the model g to calculate representation similarity d. We hope that the node representation vector will not change dramatically after we add the covering edges, so we will set a limit N on the maximum number of covering edges added to one user/encounter. We inject the covering edges in an iterative way so that in each step we add the edge (u_t, v_t) which is valid and whose representation is the most similar to that of the target access in the test graph.

7 Evaluation on Malicious Access Detection

In this section, we will show the performance of the illegitimate access detection without attacks. The attacks will be later evaluated over these detection methods.

7.1 Abnormal Access Detection System

We evaluate a variety of anomaly edge detection algorithms in our experiments, including preferential attachment (Pref), Katz index (Katz), non-negative matrix factorization (NMF), matrix factorization (MF), factorization machine (FM), Node2Vec (N2V), Node2Vec with feature (N2V-F), one-class graph neural network (OCGNN) [40] and graph neural network (GNN) [44]. Among these approaches, we find GNN to be the best approach for our task and we will introduce it as below. Other approaches are introduced in Appendix A.

Graph Neural Network-based Anomaly Edge Detection. We will use the structure of Graph Auto-encoder [32], which is a type of GNN designed for edge classification. The model $s(G,u,v)$ will take as input a graph G and a pair of nodes (u,v), and output the score that there should be an edge between (u,v). Let $\mathcal{A}(V,E) \in \{0,1\}^{|V|\times|V|}$ denote the operation of generating the adjacency matrix from the node set V and the edge set E. Given a graph $G = (V,E,X)$, the model first encodes the graph into the node embedding matrix using a GNN-based node encoder g:

$$A = \mathcal{A}(V,E) \tag{2}$$

$$Z = g(A,X) \in \mathbb{R}^{|V|\times h} \tag{3}$$

then it decodes Z into a reconstructed adjacency matrix $\hat{A}$:

$$\hat{A} = \sigma(S) \tag{4}$$

$$S = ZWZ^\intercal \tag{5}$$

and thus $f(G,u,v) = \hat{A}_{uv}$, where $W \in \mathbb{R}^{h\times h}$ is the trainable parameter, σ is chosen as the sigmoid function to ensure that each element $\hat{A}_{ij} \in (0,1)$. The score function $s(G,u,v) = S_{uv} \in \mathbb{R}$ is the prediction score before the sigmoid function.

Note that this process can also be viewed as generating a binary classification result $f(G,u,v) \in (0,1)$ for each edge (u,v), where a higher value indicates a higher probability of being benign. Following conventions, we use "positive" to refer to the illegitimate accesses in detection (although its ground truth is 0 in the adjacency matrix) and "negative" to refer to the benign ones. We will have a threshold θ so that edges with $s < \theta$ are considered positive and otherwise negative. We will use a separate validation set to find this threshold.

To train the model, we will optimize the loss function between generated adjacency matrix and the ground truth:

$$L = \sum_{u\in V_{user}, v\in V_{enc}} H(A_{uv}, \hat{A}_{uv}) \tag{6}$$

where $H(x,y) = -x\log y - (1-x)\log(1-y)$ is the binary cross entropy loss. Directly applying the loss function would lead to the problem of an unbalanced positive/negative ratio since the graph is usually sparse, so we will first randomly sample a set of positive node pairs E_{pos} with size $|E_{pos}| = |E|$ which are not connected, and optimize the loss function on:

$$L' = \sum_{(u,v)\in E\cup E_{pos}} H(A_{uv}, \hat{A}_{uv}) \tag{7}$$

We will re-sample E_{pos} every time we optimize the loss function.

Transductive Learning vs. Inductive Learning. Note that most edge classification tasks assume a *transductive learning* setting. Given an untrusted graph G, the model will be trained on this graph despite the existence of a small proportion of malicious data and applied to the graph to determine which accesses are illegitimate. In our task, an *inductive learning* setting is also possible, where training and evaluation will happen on non-overlapping datasets. We can train our model using one graph G_1 and apply it to detect the malicious edges in another graph G_2. We will consider both scenarios in the evaluation.

7.2 Model Setting

In the experiments, we use a two-layer Graph Convolutional Network [25] as the GNN encoder g with a hidden size 64 and a dropout rate 0.2. The decoder is chosen as a bilinear model as shown in Eqn. 5. Since we do not have ground truth illegitimate accesses, we will simulate them with the technique to

be introduced in Section 7.3. The simulation number is 10% of the overall access number within the period. Intuitively, with a larger number of illegitimate accesses, the detection task will be more difficult since the data becomes noisier. We view 10% illegitimate accesses as a large proportion in most cases and will vary the value to see the impact. We train the model using Adam optimizer [24] with a learning rate 0.01 and weight decay rate 5×10^{-4}. The details of baseline detection implementation are introduced in Appendix A.

As introduced in Sec. 3, we extract 50 subsets from 50 consecutive weeks from the EHR dataset. In an inductive learning setting, we will train the model using the i-th graph and evaluate on the $(i+1)$-th graph, and report the averaged result over $i \in \{1, 2, \ldots 49\}$. We will randomly set aside 10% of the edges in the training graph as the validation set. In a transductive learning setting, the model is trained and evaluated using the $(i+1)$-th graph, and we use the i-th graph as the validation set. In both settings, the validation set is used to determine the threshold for detection so that the true negative rate on the validation set is over 0.9.

The metrics we use to evaluate detection performance are true positive rate (TPR), true negative rate (TNR), precision among top-1000 positive predictions (Prec@1k), and Area Under ROC Curve (AUC). The ROC curve shows the true positive rate and the false positive rate when varying the threshold in binary classification.

7.3 Illegitimate Access

Hospitals generally do not share data on illegitimate accesses for evaluating the model performance. Therefore, simulating illegitimate accesses based on domain knowledge (i.e., known undesired behaviors) is a commonly used evaluation methodology in prior studies on EHR access logs [9, 30, 37, 43]. We use four different types of illegitimate access to evaluate model performance: (1) Sample a set of users based on their access frequency and let each of them access one randomly chosen encounter. Different sampled users access different encounters. (2) Sample a set of users based on their access frequency and let each of them access the most frequently accessed encounter (e.g. some curious accesses to a VIP patient who is taken care of by many users). (3) Sample a set of users based on their access frequency and let each of them access the least frequently accessed encounter. (4) Randomly choose one user and let this user access a variety of different encounters.

We show the evaluation with the first simulation in the following and put other simulation results in Appendix C. Later in Section 10, we will also show that the attack works similarly on a public dataset where ground-truth attacks are available.

Approach	AUC	Prec@1k	TPR	TNR
GNN-trans	0.9041	0.9479	0.7597	0.9193
GNN-ind,noisy	0.9058	0.9579	0.7706	0.9024
GNN-ind,clean	**0.9073**	**0.9629**	**0.7751**	0.8991
pref	0.6808	0.2903	0.3356	0.8987
katz	0.8413	0.4674	0.5613	0.8989
NMF	0.8504	0.4802	0.5697	0.8997
MF	0.8736	0.8944	0.7016	0.8944
FactM	0.8690	0.9046	0.6969	0.9001
N2V	0.7320	0.5226	0.4466	0.9000
N2V+nodef	0.7523	0.7198	0.5085	0.9000
OCGNN	0.5759	0.3265	0.3188	0.8167

Table 1: **Detection Performance**—Detection performance for different approaches. 'trans' refers to the transductive learning setting and 'ind' refers to the inductive learning setting on noisy or clean training set. We emphasize that the TNR metric should not be compared because we manually tune the threshold in the validation set to achieve around 0.9 TNR.

7.4 Detection Performance

The detection performance of our model is shown in Table 1. We first consider three scenarios for our GNN pipeline: a transductive learning setting (GNN-trans) where the model is trained and evaluated on the same graph which contains the malicious edges; an inductive learning setting with noisy training set (GNN-ind,noise) where the model is trained on a graph with the malicious edges and evaluated on another one; an inductive learning setting with clean training set (GNN-ind,clean) where the model is trained on a clean graph without malicious edges and evaluated on an untrusted one. We observe that the inductive learning leads to a better detection performance. This matches our intuition because in transductive learning, the same illegitimate accesses will appear in the training and evaluation so that the model will learn to recognize these accesses as benign during training. On the other hand, the inductive model trained with noisy data can achieve similar performance compared with that trained with clean data. This shows that our pipeline is resilient to the existence of potential malicious accesses in the training set. In later discussions, we will use GNN-ind,clean as our setting if not specified.

We also compare our approach with baseline approaches in Table 1. We can see that our GNN-based approach outperforms all other detection algorithms. The Katz index and matrix factorization-based approaches achieve good detection performance, while other methods are relatively bad for our task. In addition, the OCGNN trained with a one-class optimization goal has poor performance in the detection. This shows the importance of having positive data in the training process. We have also examined the detection performance by varying the training and evaluation periods (see Appendix B). We observe that the time difference has little impact.

We emphasize that the goal of this section is to find well-

performing models against which we can evaluate our attacks, but not to propose a state-of-the-art detection model for the EHR system. In the following sections, we will mainly use the GNN-based model as the detection model and also consider other models that achieve good performance, such as katz index and matrix factorization-based approaches.

8 Evaluation on Evasion Attacks

In this section, we will evaluate our evasion attack algorithm under different scenarios and show that evasion attack is effective against various detection models. Apart from the results here, we also perform experiments on public datasets in Section 10, in which we observe a similar conclusion to the results to be presented in this section.

8.1 Attack Setting

We perform our evasion attack experiment as discussed in Section 5 over the detection models trained in Section 7. Following the previous setting, we will apply the model trained on the i-th graph and perform the evasion attack on the data of the $(i+1)$-th graph, and report the averaged result over $i \in \{1,2,\ldots,49\}$. Unless specified, we assume that the attacker has white-box knowledge of the graph; the scenario of gray-box knowledge of the graph will be discussed in Section 8.5.

For each time period, we randomly choose 100 target accesses in the graph. We show the distribution of prediction scores of the chosen target accesses in Appendix C. For each target access, we will perform the evasion attack with $K \in \{2,5,10,20,50\}$ covering edges. Let G_{atk} denote the graph after injecting the covering accesses and target access. We will report the following metrics for the attacks:

- r_{tgt}: the average evasion rate of the target accesses, i.e. $\mathbb{1}(s(G_{atk}, u^*, v^*) > 0)$.
- r_{cov}: the average evasion rate of the covering accesses, i.e. $\mathbb{1}(s(G_{atk}, u_i^c, v_i^c) > 0)$.
- r_{atk}: the average success rate of the attack. An attack is successful only if its target access and all the covering accesses evade the detection, i.e.:

$$\mathbb{1}(s(G_{atk}, u^*, v^*) > 0 \wedge s(G_{atk}, u_i^c, v_i^c) > 0 \ \forall i \in [K]).$$

8.2 Baseline Attacks

No previous researches focus on the evasion attack that considers the stealthiness of covering accesses. In order to compare the performance with attacks that also consider covering access stealthiness, we also propose a heuristic-based attack for normal attacker capability. Our baseline is based on the intuition that in order to find stealthy covering accesses to disguise the target access, we hope that the accessed encounters are 1) similar to the target encounter (so that they can disguise the target access) and 2) similar with the encounters accessed by the target user (so that they are stealthy). Therefore, Given the target access (u^*, v^*), we calculate a score for each encounter v in the graph as:

$$M(v) = (1-\alpha) \cdot \text{sim}(v, v^*) + \alpha \cdot \frac{1}{|\mathcal{N}(u)|} \sum_{v_i \in \mathcal{N}(u)} \text{sim}(v, v_i)$$

where $\text{sim}(\cdot,\cdot)$ is a similarity metric between two nodes and α is a parameter balancing the tradeoff between covering access stealthiness and the ability to disguise. In practice, we use the Jaccard index as the similarity metric $\text{sim}(u,v) = \frac{\mathcal{N}(u) \cap \mathcal{N}(v)}{\mathcal{N}(u) \cup \mathcal{N}(v)}$ and $\alpha = 0.75$. The target user will access the encounters with top-K score as the covering accesses.

	Baseline			Our attack		
K	r_{tgt}	r_{cov}	r_{atk}	r_{tgt}	r_{cov}	r_{atk}
0	25.49%	-	25.49%	25.49%	-	25.49%
2	48.89%	19.80%	11.77%	40.16%	100%	**40.16%**
5	65.96%	33.40%	12.58%	50.55%	100%	**50.55%**
10	78.12%	47.25%	13.50%	60.33%	100%	**60.33%**
20	89.00%	60.20%	12.37%	69.80%	100%	**69.80%**
50	96.33%	72.43%	8.54%	77.45%	100%	**77.45%**

Table 2: Evasion attack performance.

8.3 Evasion Attack Performance

We report the attack performance in Table 2. If no attack is performed, the evasion rate of a target access is around 25%, which is the false negative rate of our system. We observe that the attack performance is good. 60% of the target accesses can successfully evade the system when disguised with 10 covering edges. With 50 covering edges, the evasion rate further improved to around 77%. In addition, with the full knowledge of the graph, the attacker will always add covering edges that are not detected by the model, hence achieving a 100% evasion rate on the covering edges. By comparison, the baseline attack only considers the evasion rate of target access. So the injected covering accesses are easy to detect and the attack success rate is low. These results show the viability of our strategy for attacking the access detection model based on graph structure. We show a case study of the evasion attack in Appendix E. We can observe that the attacker will gradually make accesses that both look plausible and can disguise the target access. Such behaviour is coherent with our attack intuition.

8.4 Attack against Different Detectors

We show the results of the evasion attack against different approaches in Figure 3. The detailed numbers are provided in Table 11 in Appendix D. In the upper part, we show the

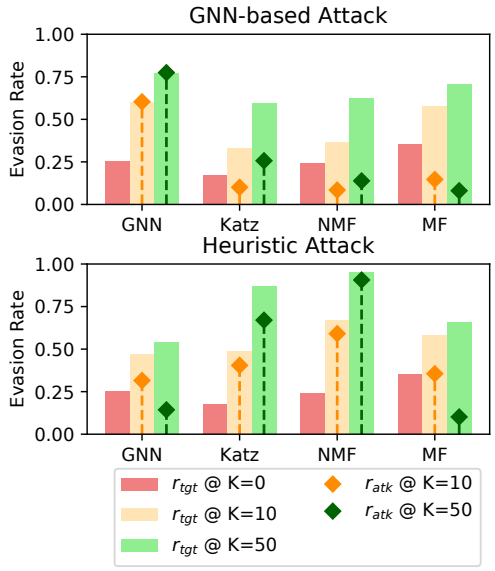

Figure 3: **Evasion Attack Performance**—The evasion attack performance of our GNN-based attack and the heuristic-based attack against different detection algorithms. r_{tgt} refers to the evasion rate of the target access while r_{atk} refers to the attack success rate (i.e. the rate of attacks where target access and all the covering accesses evade the system). Note that r_{atk} does not necessarily increase with larger K, since a larger number of covering access will increase the risk of being detected.

performance of GNN-based attacks against other detection approaches. Specifically, we will 1) generate the covering accesses w.r.t. the target access and the GNN system using previous algorithms, 2) inject the covering accesses into the graph and 3) evaluate the graph using other detection approaches. We only evaluate Katz, NMF and MF since pref and N2V-based approaches do not have a good detection performance and FactM takes too much time. We observe that our generated accesses, although generated specifically for attacking the GNN model, can also fool other algorithms to some extent. With $K = 50$ covering accesses, around 60% of the target accesses can be disguised for all other approaches. The main problem is that the attack cannot guarantee the stealthiness of the covering accesses. Therefore, the attack success rate is only around 10% to 20%.

In the lower part of Figure 3, we show the performance of heuristic-based baseline attacks on different algorithms. We have two observations from the results. First, the heuristic attack achieves a better performance on baseline detection algorithms. This shows that our intuition makes sense and the heuristic attack can indeed fool some detection systems. Second, the heuristic attack does not work well against our GNN system. We owe it to the fact that GNN-based system is more complicated than others so the simple heuristic cannot fool the system easily. This shows the superiority of GNN-based system compared with other ones - only our specifically designed

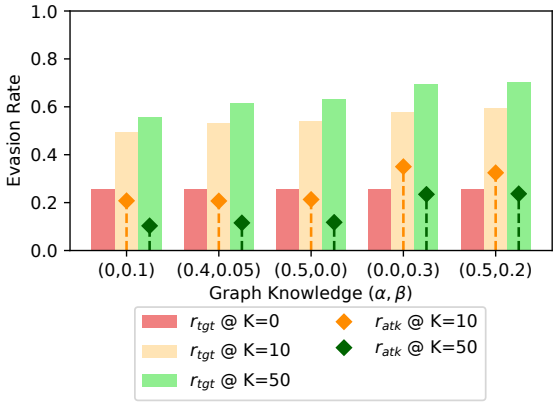

Figure 4: **Gray-box Evasion Results**—The gray-box evasion attack performance with different knowledge of the graph. The ratio of edges known by the attacker, ρ, are 0.10/0.12/0.13/0.30/0.30 for the five settings respectively. r_{tgt} refers to the evasion rate of the target access while r_{atk} refers to the attack success rate (i.e. the rate of attacks where target access and all the covering accesses evade the system). Note that r_{atk} does not necessarily increase with larger K, since a larger number of covering access will increase the risk of being detected.

algorithm can achieve a good evasion attack performance.

Note that our evasion attack can be applied specifically to attack other models. We show an example of attacking Katz in Appendix D and can observe that it also works.

8.5 Gray-box Attack

In the previous analysis, we focus on the attack with white-box knowledge of the testing graph and observe that the attacker is able to achieve a good attack performance. However, in practice the attacker may have limited knowledge of the accesses made by other users. In this section, we consider this scenario where the attacker only knows part of the graph. We assume that the attacker knows all the nodes in the original graph but only part of the edges. That is, the attacker wants to perform the attack on the graph $G = (V, E, X)$ but he/she only has access to the graph $G' = (V, E', X)$, where $E' \subset E$. We let $\rho = |E'|/|E|$ denote the ratio of edges known by the attacker. To achieve the attack, the attacker will perform the evasion attack algorithm on G' and add the resulting covering edges into G.

We first introduce the way to simulate gray-box knowledge of the graph. Intuitively, the attacker has two ways to know the accesses on the graph: those related to the attacker's job, or those learned from usual patterns. To model these two types of partial knowledge, we propose an (α, β)-*subgraph* by sampling the edges of the original graph. For the first way, we model it by calculating the distance between the user and an arbitrary edge. An edge with distance d to the user has a probability of α^d to be known by the user. For the second way, we model it by a uniform sampling over the graph so

that each edge has a probability of β to be known by the user.

The result is shown in Figure 4. We can see that the less knowledge the attacker has on the testing graph, the more difficult it will be to achieve a successful evasion attack. The evasion rate of target access dropped to around 70% with 30% knowledge of the graph and 50 covering edges added (compared with 77% in Table 2). Moreover, the attacker cannot guarantee a 100% evasion rate on the covering edges in this case. With 10% knowledge of the graph, the attack success rate is almost never better than doing nothing (which is around 25% according to Table 2). With more knowledge of the graph, injecting around 10 covering access will help with the attack. The detailed table of the results is shown in Appendix F.

9 Evaluation on Poisoning Attack

In this section, we evaluate the poisoning attack algorithm under different scenarios and show that the poisoning attack is difficult to succeed against the detection model even with very strong attack capability.

9.1 Attack Setting

We perform the poisoning attack experiment as introduced in Section 6. We will inject poisoning data on the i-th graph and evaluate the corrupted model on the data of the $(i+1)$-th graph, and report the averaged result over $i \in \{1,2,\ldots,49\}$. The target accesses are chosen as the same ones as in an evasion attack. Let G_1 denote the poisoned training graph and G_2 denote the testing graph with target access. We will report the following metric for the attacks:

- s_{tgt}: the average score of the target accesses, i.e. $s(G_2, u^*, v^*)$.
- r_{tgt}: the average evasion rate of the target accesses, i.e. $\mathbb{1}(f(G_2, u^*, v^*) > 0)$.
- d_{cov}: the average difference between the embedding vectors of covering access and target access, i.e. $d(u_i^c, v_i^c, G_1; u^*, v^*, G_2)$.
- AUC: the detection AUC of the model on G_2 after trained on the poisoned dataset.

9.2 Poisoning Attack Performance

We report the attack performance in Table 3. We observe that the poisoning attack is much more difficult compared with the previous evasion attack scenario. The attacker almost never succeeds even with 100 covering edges injected into the training set. Even with 10,000 covering edges injected, which is more than 10% of the edges in the original data, the attack success rate is still not high. The failure of the poisoning attack can be explained by our observation in Section 7.4 that the model is able to learn a good pattern even with 10% of noisy data in the training process. Hence, it achieves "self-correction" and ignores those poisoning covering accesses. We empirically find that most covering accesses are still classified as malicious even if we label them as benign accesses during the training process. Therefore, we conclude that the poisoning attack is difficult to succeed. We provide a case study of a poisoning attack in Appendix G and observe that the injected edges are indeed similar to the target edge.

K	s_{tgt}	r_{tgt}	d_{cov}	AUC
0	-1.583	26.32%	-	0.9073
10	-1.579	26.32%	2.435	0.9073
100	-1.503	26.54%	2.586	0.9073
1000	-1.183	29.25%	2.656	0.9073
10000	-0.308	40.57%	2.716	0.9049

Table 3: Poisoning attack performance.

λ	K	s_{tgt}	r_{tgt}	d_{cov}	AUC
100	10	-0.7791	36.23%	2.4201	0.9068
	20	-0.5927	36.54%	2.4513	0.9065
	50	-0.4716	40.54%	2.5075	0.9057
	100	-0.4166	42.08%	2.5364	0.9044
1000	10	-0.5727	40.92%	2.4110	0.9039
	20	-0.4880	45.00%	2.4589	0.9023
	50	-0.4595	50.54%	2.5059	0.8991
	100	-0.4163	59.15%	2.5501	0.8930

Table 4: Poisoning attack with a larger weight λ on the injected covering accesses during training.

Note that our observation is very different from those of prior studies on network intrusion detection systems (IDS)—prior studies commonly suggest that poisoning attack is highly effective against IDS [36, 38]. We suspect one of the contributors is the *high dynamism* of the EHR systems. Note that most patients only stay in the hospital for a short period of time (e.g., less than a week). Therefore, the *patient nodes* in consecutive graphs (i.e., weekly snapshots) can be highly different. In addition, poisoning edges can only exist in the training process but not in the evaluation process. Such dynamics (on nodes and edges) could make poisoning challenging.

9.3 Large Training Weight on Covering Edges

As we discussed above, the failure of the poisoning attack may be also due to the model does self-correction and automatically ignoring the poisoning edges during the training process. To validate the speculation, we evaluate an unrealistic scenario where the model is trained to focus more on the injected covering edges. In particular, we put a weight $\lambda > 1$ on the loss terms of covering edges during the training process of the model. The results are shown in Table 4. We can observe that when we apply a large weight on these covering accesses, the attack is able to achieve a higher attack performance. In

particular, when we use $\lambda = 1000$ and add $K = 100$ covering accesses, the attack success rate is around 60%. This confirms our previous assumption that the covering accesses are similar to the target access and can indeed lead the model to learn the desired pattern under certain circumstances, but only with unrealistic conditions.

10 Beyond EHR: Attack against LANL

To examine the generalizability of our results, we apply our attacks to a public dataset that is outside of the context of EHR but is within the broad context of illegal access detection.

10.1 LANL Dataset and Detection Model

We evaluate our attack methodology using the dataset from Los Alamos National Labs (LANL) [21]. LANL contains 58 consecutive days of log files collected using their lab's internal computer network with over 12,000 users and 17,000 computers. The dataset contains APT-style attacks that compromise internal hosts and perform lateral movement within the network. We focus on the authentication logs which contain ground-truth labels for benign and malicious authentication events. Out of 45M events, 518 are labeled as anomalous/malicious. The dataset has been used as a ground-truth dataset for anomalous event detection by prior works [4, 23, 27].

To build a good intrusion detection model for LANL, we use a state-of-the-art model EULER [23]. EULER stacks a model agnostic GNN upon a recurrent neural network (RNN) and builds a temporal graph for link prediction (i.e., link in this context represents an "access" to a server). EULER constructs the graph using the authentication logs by taking the source and destination servers as the nodes. The edges are created between the source and destination servers for all authentication records within a time window of 1,800 seconds. It uses an encoder-decoder framework to generate the embedding and reconstruction for the temporal graph and learns a probability function to predict the likelihood of an edge appearing based on previous temporal states. It assumes that an anomalous edge (non-edge) will occur with a low probability in the future. EULER outperforms prior works such as [15, 33] in dynamic link prediction.

We take the same approach as EULER and construct a temporal graph using authentication logs from LANL dataset with their source code[1]. In order to launch the attack, we transform the decentralized model trained by the model into a centralized model; the performance is still the same after the transformation: both models achieve a detection AUC of 0.997.

10.2 Evasion attack

For the evasion attack, we use the malicious events in the ground truth as our target edges. We can adopt the same attack approach as against the EHR system, since the overall detection model is also differentiable. We show the results of the evasion attack in Table 5, where the metrics are defined in the same way as in EHR datasets. We observe that the evasion attack performance is even better here. We can disguise each of the malicious events with at most 10 covering accesses and ensure that the covering accesses are also stealthy.

[1] https://github.com/iHeartGraph/Euler

K	r_{tgt}	r_{cov}	r_{atk}
0	13.2%	-	13.2%
2	48.5%	100%	48.5%
5	99.8%	100%	99.8%
10	100%	100%	100%
20	100%	100%	100%
50	100%	100%	100%

Table 5: Evasion attack performance on the LANL dataset.

K	r_{tgt}	r_{cov}	r_{atk}
0	55.9%	-	55.9%
2	63.0%	90.4%	59.5%
5	74.8%	95.6%	64.7%
10	83.0%	92.2%	47.1%
20	94.5%	91.4%	37.6%
50	97.9%	97.6%	67.6%

Table 6: Evasion attack performance on the LANL dataset when transferring from EULER to Katz.

We also study the transferability of our evasion attack as shown in Table 6. Similar to the setting in the EHR dataset, we first launch the attack against the EULER system and then apply the covering edges to other detection models to see whether they can disguise the target edge. We only evaluate Katz here, since the number of nodes is large and MF-based methods cannot be executed efficiently. We observe a similar observation as before — the covering edges can indeed transfer to disguise the target accesses, but their stealthiness cannot be guaranteed, so that the attack success rate may not always improve with more covering edges.

10.3 Poisoning attack

For each poisoning attack, one access randomly selected from the ground truth malicious events is used as the target access to evaluate whether the attack is successful or not. The evaluate is repeated for 100 random malicious events from the ground truth. The results are shown in Table 7. We can observe that although the poisoning attack is still more difficult to launch compared with the evasion attack, it is more applicable here than in the EHR case. This is because the nodes here are static — nodes in the testing set have already appeared in the training set. Therefore, the poisoning attack can be launched by injecting the edges during the training stage that are connected to the target edge. This confirms our previous hypothesis that the highly-dynamic property of EHR system

K	r_{tgt}	AUC
0	12%	0.997
10	17%	0.996
100	56%	0.988
1000	64%	0.979
10000	75%	0.977

Table 7: Poisoning attack performance on the LANL dataset.

makes it more difficult to launch poisoning attacks.

11 Related Work

Graph Edge Classification. In practice, many real-world applications are related to the edge classification task on graphs. The link prediction problem has been investigated in many areas such as social networks and citation networks [3, 44]. The recommendation system can be viewed as a bipartite graph between users and items, and people can apply GNNs to recommend items to users [13]. The fraud detection task on graphs [2, 20] also focuses on detecting abnormal edges in graphs.

Malicious Medical Records Access Identification. Several supervised learning approaches such as logistic regression have been applied to detect malicious access for medical records [5, 22]. Considering the privacy concerns and domain knowledge required for the labeling information, later several unsupervised learning strategies have been studied [11, 12]. However, these approaches do not take accessing graph-structured information into account explicitly. In this paper we for the first time leverage graph neural networks to model the EMR accessing patterns with deep networks.

Adversarial Attacks on GNNs and Edge Classification Models. Various researches have been conducted on attacking graph neural networks for node classification or graph classification. From the perspective of attack goal, [10, 47] perform targeted attack which aims to fool the model on certain inputs; [42, 48] perform untargeted attack and the goal is simply to degrade model performance. From the perspective of attack setting, [10, 41] perform evasion attack where the adversarial perturbation is only applied at the evaluation phase; [42, 47, 48] perform transductive attack where model training, evaluation and attack happen on the same graph. Our attack is working on the targeted attack.

Several works also focus on the attack against edge detection models. [35] proposes an optimization-based attack on knowledge graphs. [46] proposes a game theory-based attack against similarity-based link prediction. The most similar works are [7] and [26] which also attack against GNN-based models. [7] proposes a gradient-based evasion attack while [26] proposes a searching-based evasion attack. In both works, however, the perturbation budget is manually chosen and the attack will not consider the stealthiness of injected covering accesses. In addition, their link prediction task focuses on non-existing node pairs, while our access detection task focuses on existing edges. For all the works mentioned here, the model is trained in a transductive setting, while our work focuses on the inductive setting where the model is trained and evaluated on different graphs.

12 Conclusion

In this paper, we propose adversarial attacks against abnormal access detection systems for EHR data. We show that despite the good performance of detection systems on normal data, an attacker with knowledge of the system can disguise his illegitimate access with covering accesses in the system. In addition, injecting covering accesses during the evaluation phase will be much more effective than in the training phase. Such a conclusion helps with further investigation in robust EHR systems as well as in other intrusion detection tasks.

Ethics Concerns

The EHR dataset is de-identified using HIPAA Safe Harbor rules and our study is conducted with Institutional Review Board (IRB) approval. Both users and patients are de-identified in the dataset. We believe that we have demonstrated significant adversarial threats to advanced methods for detecting illegitimate access to medical records. However, we further believe that revealing these threats in this publication does not create an immediate concern for currently-deployed EHR systems. Our investigation is similar to other research on adversarial machine learning in many ways [6, 10, 26]. In particular, we aim to provide foundations for understanding risks and providing mitigation in a timely manner. For example, our findings suggest that adversarial techniques will be less effective if hospitals do not reveal the periods over which learning and prediction are carried out. This provides a simple countermeasure that can be used immediately to bolster defenses. Also, the deployment of advanced detection techniques is nascent, so there is time to continue studying mitigation techniques without risking widespread compromises in the wild.

Acknowledgements

This project would not have been possible without the generous support from Northwestern Memorial Hospital and the Northwestern Medicine Enterprise Data Warehouse. Our research was supported by NSF CNS grants 09-64392, 19-10100, 20-46726, and 20-55233. It was also supported by HHS grant 90TR0003-01 and a grant from C3 AI. Views expressed in the paper are those of the authors only.

References

[1] What's ahead for ai and machine learning in healthcare?, https://www.imprivata.com/blog/whats-ahead-ai-and-machine-learning-healthcare.

[2] Leman Akoglu, Hanghang Tong, and Danai Koutra. Graph based anomaly detection and description: a survey. *Data mining and knowledge discovery*, 29(3):626–688, 2015.

[3] Mohammad Al Hasan and Mohammed J Zaki. A survey of link prediction in social networks. In *Social network data analytics*, pages 243–275. Springer, 2011.

[4] Benjamin Bowman, Craig Laprade, Yuede Ji, and H. Howie Huang. Detecting lateral movement in enterprise computer networks with unsupervised graph ai. In *RAID*, 2020.

[5] Aziz A Boxwala, Jihoon Kim, Janice M Grillo, and Lucila Ohno-Machado. Using statistical and machine learning to help institutions detect suspicious access to electronic health records. *Journal of the American Medical Informatics Association*, 18(4):498–505, 2011.

[6] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *2017 ieee symposium on security and privacy (sp)*, pages 39–57. IEEE, 2017.

[7] Jinyin Chen, Ziqiang Shi, Yangyang Wu, Xuanheng Xu, and Haibin Zheng. Link prediction adversarial attack. *arXiv preprint arXiv:1810.01110*, 2018.

[8] Pin-Yu Chen, Huan Zhang, Yash Sharma, Jinfeng Yi, and Cho-Jui Hsieh. Zoo: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, pages 15–26, 2017.

[9] You Chen, Steve Nyemba, and Bradley Malin. Detecting anomalous insiders in collaborative information systems. *IEEE transactions on dependable and secure computing*, 9(3):332–344, 2012.

[10] Hanjun Dai, Hui Li, Tian Tian, Xin Huang, Lin Wang, Jun Zhu, and Le Song. Adversarial attack on graph structured data. In *International conference on machine learning*, pages 1115–1124. PMLR, 2018.

[11] Daniel Fabbri and Kristen LeFevre. Explanation-based auditing. *Proceedings of the VLDB Endowment*, 5(1):1–12, 2011.

[12] Daniel Fabbri, Kristen LeFevre, and David A Hanauer. Explaining accesses to electronic health records. In *Proceedings of the 2011 workshop on Data mining for medicine and healthcare*, pages 10–17, 2011.

[13] Wenqi Fan, Yao Ma, Qing Li, Yuan He, Eric Zhao, Jiliang Tang, and Dawei Yin. Graph neural networks for social recommendation. In *The World Wide Web Conference*, pages 417–426, 2019.

[14] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[15] Palash Goyal, Sujit Rokka Chhetri, and Arquimedes Canedo. dyngraph2vec: Capturing network dynamics using dynamic graph representation learning. *Knowledge-Based Systems*, 2020.

[16] Aditya Grover and Jure Leskovec. Node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864, 2016.

[17] Ash har Quraishi. At least 50 northwestern hospital employees fired for accessing smollett's profile, records: Sources, https://www.nbcchicago.com/news/national-international/northwestern-memorial-hospital-employees-fired-jussie-smsollett-records/2383/.

[18] Monica Hedda, Bradley A Malin, Chao Yan, and Daniel Fabbri. Evaluating the effectiveness of auditing rules for electronic health record systems. In *AMIA Annual Symposium Proceedings*, volume 2017, page 866. American Medical Informatics Association, 2017.

[19] Ashish K Jha, Catherine M DesRoches, Eric G Campbell, Karen Donelan, Sowmya R Rao, Timothy G Ferris, Alexandra Shields, Sara Rosenbaum, and David Blumenthal. Use of electronic health records in us hospitals. *New England Journal of Medicine*, 360(16):1628–1638, 2009.

[20] Meng Jiang, Peng Cui, Alex Beutel, Christos Faloutsos, and Shiqiang Yang. Catchsync: catching synchronized behavior in large directed graphs. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 941–950, 2014.

[21] Alexander D. Kent. Cybersecurity Data Sources for Dynamic Network Research. In *Dynamic Networks in Cybersecurity*. Imperial College Press, June 2015.

[22] Jihoon Kim, Janice M Grillo, Aziz A Boxwala, Xiaoqian Jiang, Rose B Mandelbaum, Bhakti A Patel, Debra Mikels, Staal A Vinterbo, and Lucila Ohno-Machado. Anomaly and signature filtering improve classifier performance for detection of suspicious access to ehrs. In *AMIA Annual Symposium Proceedings*, volume 2011, page 723. American Medical Informatics Association, 2011.

[23] Isaiah J. King and H. Howie Huang. Euler: Detecting network lateral movement via scalable temporal graph link prediction. In *Proc. of NDSS*, 2022.

[24] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR (Poster)*, 2015.

[25] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *ICLR (Poster)*, 2017.

[26] Wanyu Lin, Shengxiang Ji, and Baochun Li. Adversarial attacks on link prediction algorithms based on graph neural networks. In *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security*, pages 370–380, 2020.

[27] Fucheng Liu, Yu Wen, Dongxue Zhang, Xihe Jiang, Xinyu Xing, and Dan Meng. Log2vec: A heterogeneous graph embedding based approach for detecting cyber threats within enterprise. In *Proc. of CCS*, 2019.

[28] Yanpei Liu, Xinyun Chen, Chang Liu, and Dawn Song. Delving into transferable adversarial examples and black-box attacks. *arXiv preprint arXiv:1611.02770*, 2016.

[29] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations*, 2018.

[30] Robert Mitchell and Ray Chen. Behavior rule specification-based intrusion detection for safety critical medical cyber physical systems. *IEEE Transactions on Dependable and Secure Computing*, 12(1):16–30, 2014.

[31] Seong Joon Oh, Bernt Schiele, and Mario Fritz. Towards reverse-engineering black-box neural networks. In *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*, pages 121–144. Springer, 2019.

[32] S Pan, R Hu, G Long, J Jiang, L Yao, and C Zhang. Adversarially regularized graph autoencoder for graph embedding. In *IJCAI International Joint Conference on Artificial Intelligence*, 2018.

[33] Aldo Pareja, Giacomo Domeniconi, Jie Chen, Tengfei Ma, Toyotaro Suzumura, Hiroki Kanezashi, Tim Kaler, Tao Schardl, and Charles Leiserson. Evolvegcn: Evolving graph convolutional networks for dynamic graphs. In *Proc. of AAAI*, 2020.

[34] Jordan J Pellett, Olufemi Omitaomu, Mohammed M Olama, Ozgur Ozmen, Hilda Klasky, Laura Pullum, Teja Kuruganti, Merry Ward, Angela Laurio, Jeanie Scott, et al. Detection of anomalous events in electronic health records. Technical report, Oak Ridge National Lab.(ORNL), Oak Ridge, TN (United States), 2020.

[35] Pouya Pezeshkpour, Yifan Tian, and Sameer Singh. Investigating robustness and interpretability of link prediction via adversarial modifications. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 3336–3347, 2019.

[36] Ya-guan QIAN, Hong-bo LU, Shou-ling JI, Wu-jie ZHOU, Shu-hui WU, Jing-sheng LEI, and Xiang-xing TAO. A poisoning attack on intrusion detection system based on svm. *ACTA ELECTONICA SINICA*, 47(1):59, 2019.

[37] Soumi Ray, Dustin S McEvoy, Skye Aaron, Thu-Trang Hickman, and Adam Wright. Using statistical anomaly detection models to find clinical decision support malfunctions. *Journal of the American Medical Informatics Association*, 25(7):862–871, 2018.

[38] Benjamin IP Rubinstein, Blaine Nelson, Ling Huang, Anthony D Joseph, Shing-hon Lau, Satish Rao, Nina Taft, and J Doug Tygar. Antidote: understanding and defending against poisoning of anomaly detectors. In *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement*, pages 1–14, 2009.

[39] Ali Shafahi, W Ronny Huang, Mahyar Najibi, Octavian Suciu, Christoph Studer, Tudor Dumitras, and Tom Goldstein. Poison frogs! targeted clean-label poisoning attacks on neural networks. In *Advances in Neural Information Processing Systems*, pages 6103–6113, 2018.

[40] Xuhong Wang, Baihong Jin, Ying Du, Ping Cui, Yingshui Tan, and Yupu Yang. One-class graph neural networks for anomaly detection in attributed networks. *Neural Computing and Applications*, pages 1–13, 2021.

[41] Huijun Wu, Chen Wang, Yuriy Tyshetskiy, Andrew Docherty, Kai Lu, and Liming Zhu. Adversarial examples on graph data: Deep insights into attack and defense. In *IJCAI International Joint Conference on Artificial Intelligence*, 2019.

[42] Kaidi Xu, Hongge Chen, Sijia Liu, Pin Yu Chen, Tsui Wei Weng, Mingyi Hong, and Xue Lin. Topology attack and defense for graph neural networks: An optimization perspective. In *28th International Joint Conference on Artificial Intelligence, IJCAI 2019*, pages 3961–3967. International Joint Conferences on Artificial Intelligence, 2019.

[43] Prosper K Yeng, Muhammad Ali Fauzi, and Bian Yang. Workflow-based anomaly detection using machine learning on electronic health records' logs: A comparative study. In *2020 International Conference on Computational Science and Computational Intelligence (CSCI)*, pages 753–760. IEEE, 2020.

[44] Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 5171–5181, 2018.

[45] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI Open*, 1:57–81, 2020.

[46] Kai Zhou, Tomasz P Michalak, and Yevgeniy Vorobeychik. Adversarial robustness of similarity-based link prediction. In *2019 IEEE International Conference on Data Mining (ICDM)*, pages 926–935. IEEE Computer Society, 2019.

[47] Daniel Zügner, Amir Akbarnejad, and Stephan Günnemann. Adversarial attacks on neural networks for graph data. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2847–2856, 2018.

[48] Daniel Zügner and Stephan Günnemann. Adversarial attacks on graph neural networks via meta learning. In *International Conference on Learning Representations*, 2018.

A Other Detection Models

A.1 Model Details

We compare our GNN-based detection system with other popular anomaly edge detection algorithms introduced in [44], including preferential attachment (Pref), Katz index (Katz), non-negative matrix factorization (NMF), matrix factorization (MF), factorization machine (FM), Node2Vec (N2V), Node2Vec with feature (N2V-F) and one-class graph neural network (OCGNN). All these approaches except for OCGNN are designed in the transductive learning setting and we run these algorithms in the same setting as our transductive setting for GNN. We run OCGNN in the inductive setting. **Pref:**

This is a heuristic score for linkage probability calculated by $|\mathcal{N}(u)| \cdot |\mathcal{N}(v)|$ where $\mathcal{N}(u)$ denotes the neighborhood of u. A higher score indicates a larger probability of linkage.

Katz: This is also a heuristic score for linkage probability calculated by $\sum_{l=1}^{L_{max}} \beta^l |\text{walks}^{<l>(u,v)}|$, where $\text{walks}^{<l>(u,v)}$ is the set of length-l walks between u and v. A higher score indicates a larger probability of linkage. We use $\beta = 0.001$ and L_{max} in practice.

NMF: Given the adjacency matrix $A \in \mathbb{R}^{|V|\times|V|}$, NMF will look for two nonnegative matrix $U, V \in (\mathbb{R}^+)^{|V|\times h}$ to minimize $||A - UV^\intercal||_F$. Then we use the recovered adjacency matrix $UV^\intercal$ to determine which edges are illegitimate. We use the scikit-learn toolkit for solving the problem and use the default parameters, except that we set $h = 100$ for better detection performance.

MF: This algorithm is mostly the same with NMF except that we no longer require U, V to have only non-negative values. We use the fastFM package for solving the problem and use the default ALS solver.

FM: In the factorization machine, the information of each edge is processed into a vector x, including the one-hot representation of the two nodes and the feature vectors of the two nodes. Then the model will fit a prediction model of linkage score $y(x)$ by:

$$y(x) = w_0 + \sum_i w_i x_i + \sum_{i<j} < v_i, v_j > x_i x_j$$

where w and v are the trainable parameters. We use the fastFM package for solving the problem and use the default ALS solver.

N2V: Node2Vec proposes to generate an embedding vector for each node in a graph via random walk. After the node embedding vector is generated, we will concatenate the vectors for the node pairs in an edge to get the edge embedding vector and fit a logistic regression model to predict linkage. We use the node2vec package for embedding vector generation.

N2V-F: This algorithm is mostly the same with N2V except that we concatenate the node feature vectors in addition

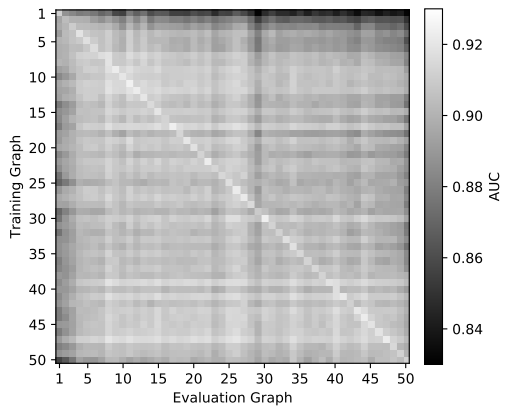

Figure 5: Detection AUC of the pipeline when trained on the data of one period and evaluated on the data of another period. We can see that the detection performance is usually good even when the model is trained and evaluated on different time periods.

to the generated node embedding to get the edge embedding vector.

OCGNN: This approach uses the same graph neural network structure as in our main text. The key difference is that the training process involves only benign data and the model is trained with a one-class optimization goal. We follow the same optimization process and parameters as in [40], except that we use the concatenation of both node vectors as the representation vector of the edge (in their paper they are working on node classification, so no edge vector is needed).

B Detection Performance Over Time

We show the model detection performance on different time periods as shown in Figure 5. We can observe that the detection AUCs on the diagonal are usually the highest (with the lightest color). This matches the intuition that a model trained on the period should also have the best evaluation performance on the same period. On the other hand, we also observe that the transferability of models is good. In most cases the model achieves over 0.88 detection AUC.

C Addition Detection Results

Varied Ratio of Illegitimate Accesses. In previous detection experiments we assume a 10% of illegitimate accesses during evaluation. We will evaluate how the detection performance changes with a different proportion. In Table 8, we show the result when we vary the proportion. We observe that the detection performance (AUC, TPR) indeed improves by a small margin with fewer illegitimate accesses. This is expect-able since more illegitimate accesses will yield to more confusion for detection. The Prec@1k metric drops because it is a metric related to the number of injected accesses - the more injected, the metric will be higher. On the other hand, the difference in the metric is indeed small. Therefore, we will use 10% in the overall experiments.

Approach	AUC	Prec@1k	TPR	TNR
GNN-ind,clean-10%	**0.9073**	**0.9629**	**0.7751**	0.8991
GNN-ind,clean-5%	0.9121	0.9069	0.8104	0.8757
GNN-ind,clean-2%	0.9135	0.6524	0.8146	0.8747
GNN-ind,clean-1%	0.9155	0.4004	0.8178	0.8747

Table 8: Detection performance with different proportion of illegitimate accesses injected.

Model Type	AUC	Prec@1k	TPR	TNR
GNN	**0.9073**	**0.9629**	**0.7751**	0.8991
DNN	0.7897	0.7580	0.5659	0.8815
GNN (none)	0.7554	0.8280	0.4670	0.9035
GNN (udep)	0.8214	0.7640	0.6225	0.9066
GNN (urole)	0.8584	0.8574	0.6683	0.9130
GNN (elos)	0.8220	0.7481	0.5957	0.9071
GNN (page)	0.8505	0.8576	0.6776	0.9093
GNN (pgender)	0.8114	0.7353	0.5804	0.9026
GNN (pdiag)	0.8743	0.8621	0.7376	0.8693
GNN (-udep)	0.9015	0.9576	0.7756	0.8843
GNN (-urole)	0.8997	0.9233	0.7684	0.8955
GNN (-elos)	0.9057	0.9589	0.7754	0.8956
GNN (-page)	0.9050	0.9603	0.7771	0.8987
GNN (-pgender)	0.9055	0.9613	0.7699	0.9010
GNN (-pdiag)	0.8915	0.9603	0.7395	0.9049

Table 9: Detection performance of our pipeline. (none) represents the model without any node information; (*) represents the model only with that feature; (-*) represents the model with all node features except the specified one. The feature name abbreviations were introduced in Section 3.

Detection Performance with Different Feature Sets. In Table 9, we compare the model with a 2-layer neural network (DNN) which does not consider graph structure, as well as GNN models without any node features (denoted by none), with one type of feature (denoted by feature name) and with all but one type of feature (denoted by '- feature name'). The feature names were introduced in Section 3. We observe that if we only consider either node features (DNN) or graph structures (GNN (none)), the performance will not be good. This indicates the importance of combining node information and graph structure in the system. We observe that all features help with the detection performance. The patient diagnosis information is the most important feature in the detection, as removing it will lead to the largest performance drop and we can use only the diagnosis information to build a good detection model.

Different Anomaly Simulations. We introduced four simulations of illegitimate accesses in Section 7.3. Here we show the evaluation results under these different simulations as in Table 10. For simulations 2-4, we inject 100 illegitimate accesses instead of 10% of the overall access number, as too many accesses to the same encounter/from the same user

Simulation	AUC	TPR	TNR
Sim1	0.9073	0.7751	0.8991
Sim2	0.7276	0.4694	0.8739
Sim3	0.9197	0.8041	0.8738
Sim4	0.9204	0.8020	0.8738

Table 10: Detection performance using different simulations.

Table 11: The evasion attack performance of our GNN-based attack and the heuristic-based baseline attack against different detection algorithms.

Model		GNN	Katz	NMF	MF
Init r_{tgt}		25.49%	17.38%	24.23%	35.23%
GNN Attack	r_{tgt} @ $K=10$	60.33%	32.92%	36.62%	57.77%
	r_{cov} @ $K=10$	100.0%	36.00%	48.70%	76.82%
	r_{atk} @ $K=10$	60.33%	10.08%	8.46%	14.69%
	r_{tgt} @ $K=50$	77.45%	59.38%	62.62%	71.00%
	r_{cov} @ $K=50$	100.0%	66.59%	70.50%	82.06%
	r_{atk} @ $K=50$	77.45%	25.69%	13.85%	8.08%
Heuristic Attack	r_{tgt} @ $K=10$	46.69%	48.92%	67.08%	58.31%
	r_{cov} @ $K=10$	76.45%	72.14%	87.30%	84.23%
	r_{atk} @ $K=10$	31.57%	40.46%	59.08%	35.54%
	r_{tgt} @ $K=50$	53.77%	87.15%	95.15%	65.92%
	r_{cov} @ $K=50$	77.59%	94.26%	98.44%	86.98%
	r_{atk} @ $K=50$	14.31%	67.00%	90.62%	10.15%

is not acceptable. We can observe that the model performs consistently except for the second simulation, i.e. from different users to the most frequently accessed encounter. This is intuitively understandable - since a wide variety of users have accessed the counter, it is rather difficult to detect one illegitimate access out of them. This also suggests a weak point in the automatic detection of illegitimate access.

D Evasion Attack

Evasion Attack against Different Approaches. We show the detailed number of evasion attacks against different approaches in Table 11.

K	r_{tgt}	r_{cov}	r_{atk}
0	17.38%	-	-
2	20.29%	100%	20.29%
5	25.29%	100%	25.29%
20	38.57%	100%	38.57%
50	58.65%	100%	58.65%

Table 12: Designed evasion attack for Katz.

Specific Evasion Attack for Katz. In our evasion attack, we repeatedly pick covering accesses based on the gradient of prediction score w.r.t. the adjacency matrix. This idea can be generalized to attack other algorithms as long as the prediction score is calculated by the adjacency matrix in a differentiable way. Hence, we can apply the same idea to attack the Katz index-based system. The calculation of Katz function can be

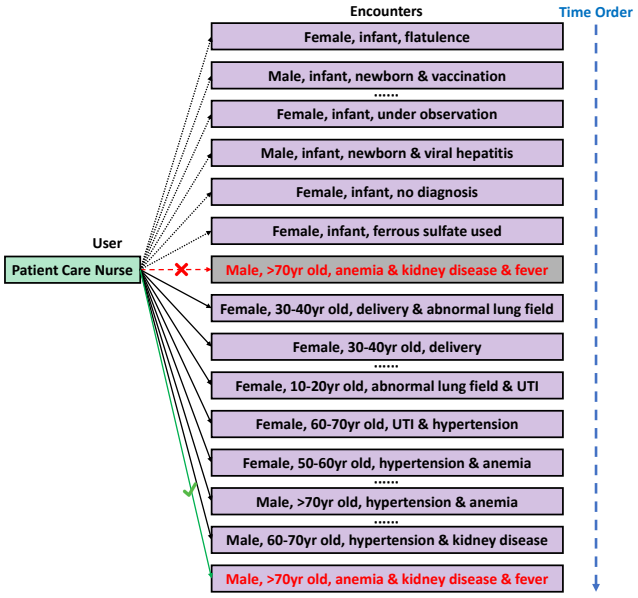

Figure 6: **An Example of Evasion Attack**—An example of the evasion attack. The encounter in red color is the target encounter to access. The black dashed arrows are the benign accesses that originally exist; the red arrow is the target access that will be detected and the shaded encounter is a rejected one which will not exist in the dataset; the black solid arrows are the injected covering accesses which seem benign to the model; the green arrow is the target access which seems benign to the model after the injection of covering accesses.

formalized as:

$$\sum_{l=1}^{L_{max}} \beta^l |\text{walks}^{<l>(u,v)}|$$
$$= \sum_{l=1}^{L_{max}} \beta^l (A^l)_{uv}$$

which is a differentiable function w.r.t. the adjacency matrix A. Therefore, we can apply the same evasion attack algorithm as in Alg. 1 to generate covering accesses against Katz. The result is shown in Table 12. We can observe that our gradient-based approach also applies to attack Katz-based detection systems. With $K = 50$ edges injected the evasion rate achieves around 60% and all the covering accesses are stealthy to the system.

E Case Study of Evasion Attack

We provide an example of the evasion attack in Figure 6. We omit some encounter details in the figure and only keep the most important ones. We do not include a LIME-based explanation here because our attack does not change the node features, so only the structural prior will significantly change in the interpretation.

α, β	ρ	K	s_{tgt}	r_{tgt}	r_{cov}	r_{atk}
0.0,0.1	0.10	2	-1.167	37.38%	81.27%	26.54%
		5	-0.795	44.46%	84.41%	23.46%
		10	-0.517	49.31%	86.68%	20.77%
		20	-0.300	53.31%	89.06%	16.00%
		50	-0.157	55.69%	90.85%	10.31%
0.4,0.05	0.12	2	-1.085	38.00%	81.77%	26.85%
		5	-0.673	47.23%	84.30%	24.15%
		10	-0.349	53.15%	86.72%	20.69%
		20	-0.099	57.69%	88.64%	16.15%
		50	0.067	61.38%	93.00%	11.54%
0.5,0.0	0.13	2	-1.070	39.08%	81.85%	28.31%
		5	-0.649	47.69%	84.50%	26.46%
		10	-0.322	54.23%	86.01%	21.31%
		20	-0.065	58.46%	88.08%	16.08%
		50	0.116	63.31%	90.27%	11.77%
0.0,0.3	0.30	2	-0.959	41.15%	90.38%	33.92%
		5	-0.466	50.69%	91.71%	36.08%
		10	-0.081	57.92%	92.85%	35.00%
		20	0.240	64.46%	93.97%	29.85%
		50	0.461	69.54%	95.21%	23.46%
0.5,0.2	0.30	2	-0.930	41.08%	90.46%	34.00%
		5	-0.423	51.62%	91.54%	36.08%
		10	-0.023	59.38%	92.06%	32.46%
		20	0.299	65.38%	93.98%	30.00%
		50	0.530	70.38%	94.84%	23.69%

Table 13: The performance of gray-box evasion attack with different knowledge of the graph. The attacker's knowledge is measured by ρ (the ratio of edges in the graph known by the attacker). The edge sampling is controlled by α and β.

In this example, the attacker is a patient care nurse who wants to access a male elderly (marked as the red encounter) who suffers from anemia, kidney disease and fever. However, the earlier accesses made by the attacker are all related with the encounters of infants (the black dashed arrows). Therefore, the model easily recognizes the target access as malicious with a high score: $s \approx -6.98$ (the red arrow).

In order to disguise the target access, the attacker needs to inject some covering accesses (the black solid arrows). The first several accesses go to encounters of pregnant women since it is natural for a nurse who works with infants to also access the charts of the mothers of the infants. After that, the attacker will access some females with other diagnoses, usually generic and not gender-specific. Gradually, the model will view it as benign for this user to access the elders with a generic diagnosis regardless of gender. After injecting 20 covering accesses, the attacker succeeds in accessing the target encounter (the green arrow) without raising an alert.

F Gray-box Attack

We show the detailed number of gray-box evasion attacks in Table 13.

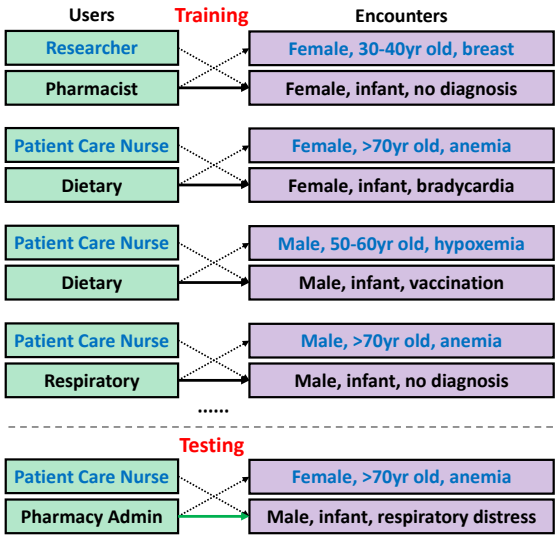

Figure 7: **An Example of Poisoning Attack**—The users and encounters in blue color represent the most frequently appeared features in the users accessing a related encounter or in the encounters accessed by a related user. The black dashed arrows are the benign accesses that originally exist; the black solid arrows are the injected covering accesses in the training data; the green arrow is the target access in the testing data.

G Case Study of Poisoning Attack

We show an example of the poisoning attack in Figure 7. We omit some encounter details and only keep the most important ones. The accesses actually being injected are shown in solid arrows and with users and encounters in black color. We also provide information of the most frequently appeared features in the users accessing an encounter/encounters accessed by a user in blue color and by dashed arrows. The four covering accesses at the top are injected into the training data; the target access at the bottom is performed during the testing phase.

In this example, the adversary aims to make an access from a pharmacy admin to a male infant. However, we observe that most of the previous accesses go to elder people, so this access will look weird. To disguise the target access, the adversary will inject several covering accesses during the training process. We observe that most covering accesses are from users that used to access encounters of the elder man and go to encounters of infants. These users and encounters are similar to the ones in the target access and the adversary hopes that the model will be trained to recognize such access patterns as benign. However, such an attack is weaker than the evasion attack and it cannot succeed unless a larger weight is put on the injected covering accesses during training.