




# *VinciDecoder*: Automatically Interpreting Provenance Graphs into Textual Forensic Reports with Application to OpenStack

Azadeh Tabiban<sup>1</sup>, Heyang Zhao<sup>1</sup>, Yosr Jarraya<sup>2</sup>,  
Makan Pourzandi<sup>2</sup>, and Lingyu Wang<sup>1</sup>

<sup>1</sup> CIISE, Concordia University, Montreal, QC, Canada

<sup>2</sup> Ericsson Security Research, Ericsson Canada, Montreal, QC, Canada

<sup>1</sup>{a\_tabiba, z\_heyang, wang}@ciise.concordia.ca

<sup>2</sup>{yosr.jarraya, makan.pourzandi}@ericsson.com

**Abstract.** The operational complexity and dynamicity of clouds highlight the importance of automated solutions for explaining the root cause of security incidents. Most existing works rely on human analysts to interpret provenance graphs for root causes of security incidents. However, navigating and understanding a large and complex cloud-scale provenance graph can be very challenging for human analysts. Without such an understanding, cloud providers cannot effectively address the underlying security issues causing the incidents, such as vulnerabilities or misconfigurations. In this paper, we propose VinciDecoder, an automated approach for generating natural language forensic reports based on provenance graphs. Our main observation is that the way nodes and edges compose a path in provenance graphs is similar to how words compose a sentence in natural languages. Therefore, VinciDecoder leverages a novel combination of provenance analysis, natural language translation, and machine-learning techniques to generate forensic reports. We implement VinciDecoder on an OpenStack cloud testbed, and evaluate its performance based on real-world attacks. Our user study and experimental results demonstrate the effectiveness of our approach in generating high-quality reports (e.g., up to 0.68 BLEU score for precision).

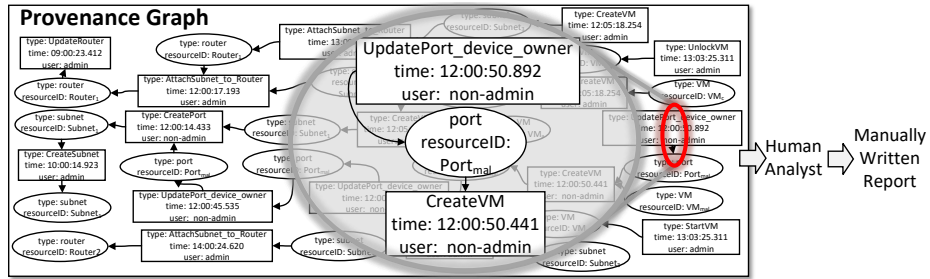
## 1 Introduction

With the recent worldwide surge in adopting cloud computing, there is an increasing need for explaining the root cause of security incidents in large scale cloud infrastructures [1]. Sharing detailed forensic reports about such root causes and attack techniques can raise cybersecurity awareness, and improve threat detection and attack prevention techniques [24]. However, most existing provenance-based solutions (e.g., [40, 47, 53]) would face a critical challenge in such a context, i.e., it would be impractical to rely on human analysts to interpret the large and complex provenance graphs produced by such solutions for a large cloud with tens of thousands of inter-connected virtual resources [35].

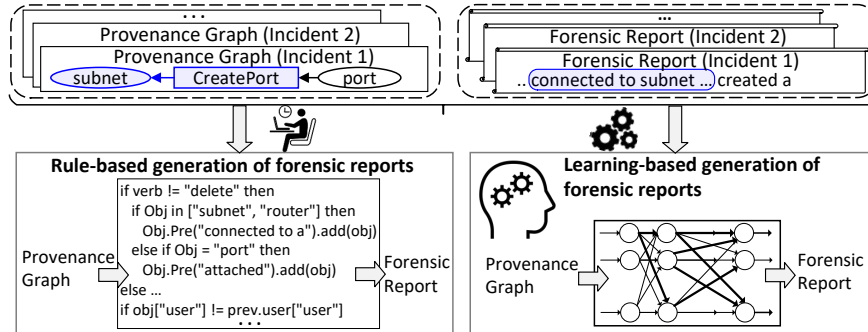
There exist rule-based techniques (e.g., [49]) for generating textual summaries of provenance graphs. However, only relying on a set of specified rules [49] would

not be sufficient, as the unpredictable nature of security incidents [57] will necessitate to constantly develop new rules, which may be costly especially for large clouds. We will further illustrate such limitations through the following example.

**Motivating example.** Fig. 1(a) depicts a provenance graph (left), and an analyst manually performing the task of creating a human-readable report (right) based on the provenance graph. Specifically, upon receiving an alert about the leakage of network traffic, the analyst begins investigating the suspicious paths of the provenance graph (left) generated by existing tools (e.g., [54]) to manually report the root cause as shown in Fig. 1a (right) (the exploit of a vulnerability [3] by updating the *device\_owner* field of a port attached to a created VM). However, such a task can be challenging to an analyst, especially as a real world cloud provenance graph may have tens of thousands of nodes and edges [47].



(a) Challenges of interpreting provenance graphs: excerpt of the provenance graph (left); an analyst manually creating a report based on the provenance graph (right).



(b) Our main idea: provenance graphs of several incidents (top left); existing reports (top right); automatic generation of forensic reports (bottom left and right).

Fig. 1: Motivating example.

- **Key ideas.** Fig. 1(b) shows the two main approaches adopted by our solution, namely *VinciDecoder*, for automatically interpreting provenance graphs into forensic reports. First, our rule-based approach generates customized forensic reports based on lexicons and grammar rules as illustrated in Fig. 1(b) (bottom left). Such rules are specified by the analyst according to his/her criteria (e.g., domain-knowledge) and understanding of the existing paired provenance graphs and forensic reports for similar types of future

attacks. Second, for use cases where such criteria are too dynamic (e.g., new types of attacks) for a rule-based approach to handle, we also propose a learning-based approach (bottom right) which automatically learns the correspondence between pairs of provenance graphs and forensic reports using Neural Machine Translation (NMT). Specifically, similar to words (e.g., verbs and object) of a sentence, there is a dependency between nodes (e.g., operations and their affected resources) in a provenance graph, which inspires us to train a translation model by applying NMT to provenance graphs (source language) paired with human-readable reports (target language), and automatically translate future provenance graphs into a natural language interpretation using the trained model.

- **Challenges.** Although our vision for adopting NMT seems plausible, realizing VinciDecoder requires addressing the following two main challenges. First, NMT is typically applied to textual data, whereas provenance graphs are usually stored as nodes and edges. To address this, VinciDecoder converts paths of provenance graphs into primitive sentences of node properties (detailed in 3.2). Second, it is challenging to generate high quality reports with a limited number of paired provenance graphs and reports for training. To address this, VinciDecoder leverages tens of thousands of CVE entries and their corresponding provenance graphs to train NMT (detailed in 4.2).

In summary, our main contributions are as follows:

- To the best of our knowledge, VinciDecoder is the first solution for generating forensic reports based on provenance analysis results using both rule-based and learning-based techniques. By reducing the reliance on human analysts to interpret and document large and complex provenance graphs, our approach can avoid the limitations, human error, and delay that are natural to such manual efforts, and thus improve the practicality of provenance analysis in large-scale cloud environments, enable automated documentation of root causes for security incidents, and allow for more timely incident-response.
- To automatically generate reports using NMT, we design several mechanisms as follows. VinciDecoder first converts provenance graph paths into primitive sentences representing properties of nodes, and removes instance-specific information to avoid mis-translation; it then learns a translation model based on the paired primitive sentences and reports; finally, when given target paths, VinciDecoder applies the learned model to the primitive sentences representing the paths to generate the forensic report. Optionally, our rule-based approach can form forensic reports by linking the node properties extracted from the target path based on pre-specified rules.
- We implement VinciDecoder on an OpenStack-based cloud testbed, and validate its effectiveness based on real-world security incidents. Our experiments and user study show that VinciDecoder generates high-quality results (e.g., up to 0.68 BLEU score for precision) with sufficient readability for human analysts (e.g., 92% of our participants agree that understanding the attack steps is much easier using VinciDecoder’s report than using provenance graphs).

The rest of this paper is organized as follows: Section 2 provides some background on data provenance and NMT. Section 3 details our methodology. Section 4 describes our implementation and presents the evaluation results. We discuss different aspects of our work and the related work in Section 5 and Section 6, respectively. We conclude the paper in Section 7.

## 2 Preliminaries

This section provides a background on data provenance, NMT and our assumptions.

### 2.1 Provenance Graph

As a powerful technique to capture the dependencies between data objects (e.g., virtual resources or operating system files) and events (e.g., management operations or system calls) in a graph representation, data provenance has been applied to clouds. We show an example of a cloud management-level provenance graph [47] in Fig. 2(a) consisting of two types of nodes: *entities* (shown as ovals) and *activities* (shown as rectangles), where entities represent virtual resources (e.g., a virtual port  $Port_{mal}$ ), and activities represent cloud management operations (e.g., an operation  $CreateVM$ ). Each node stores several properties such as the type of the operations/resources and the user who triggers the operations. Edges indicate the dependency between an operation and its affected resources. For example, in Fig. 2(a), the edge from  $CreateVM$  to  $Port_{mal}$  shows that this operation attaches  $Port_{mal}$  to the created VM  $VM_{mal}$ .

### 2.2 Neural Machine Translation

Neural Machine Translation (NMT) [46] builds a conditional probability model,  $P(Y|X)$ , such that the likelihood of a target sentence  $Y$  given a source sentence  $X$  is maximized [22]. As Fig. 2(b) shows, NMT usually consists of an encoder and a decoder, which typically utilize recurrent neural networks (RNN) such as a Long Short-Term Memory (LSTM) [23]. To initialize the training, LSTM cells are assigned with random *weights*, and the encoder captures the semantics of  $X$  by encoding it into a fixed-length vector  $H$ . Then, the decoder generates the target sentence given the computed vector  $H$ . NMT computes the deviation of the generated sentence from the reference sentence  $Y$  and improves the model by optimizing the assigned weights based on other pairs of sentences. In Section 3.4 and 3.5, we detail how VinciDecoder leverages this mechanism to generate forensic reports.

### 2.3 Assumptions

We assume the accuracy of provenance analysis results provided by existing tools (e.g., [54]), such as suspicious paths capturing the attack steps or malicious behavior. We also assume the correctness and completeness of provenance-based root cause analysis solutions (e.g., [21, 54]) in identifying suspicious paths capturing the attack steps. We assume that the provenance construction tool is not

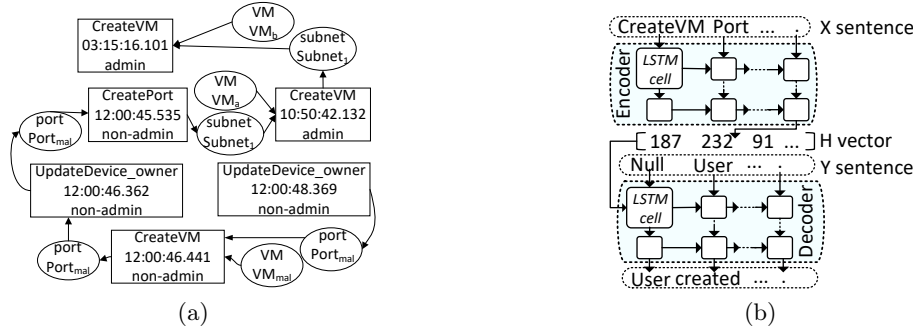


Fig. 2: An excerpt of a cloud management-level provenance graph (a); an example of NMT Encoder-Decoder model (b).

compromised. Finally, similar to most other learning-based data-to-text techniques (e.g., [42]), we assume the availability of a sufficient amount of training data (i.e., paired forensic reports and suspicious paths) for training our model<sup>1</sup>.

### 3 VinciDecoder

In this section, we provide an overview of VinciDecoder, detail its different modules, and describe the interaction between them.

#### 3.1 Overview

Fig. 3 shows an overview of VinciDecoder, which consists of two main phases: learning phase and automatic report generation phase. In the learning phase, VinciDecoder collects paired suspicious paths and reports for training, and then transforms suspicious paths into primitive sentences in our intermediary language (Section 3.2), which represents the properties of a node as a compound word, and removes the instance-specific information (Section 3.3). Next, it applies NMT to train a translation model profiling the correspondence between the obtained sentences and their forensic reports (Section 3.4). In the automatic report generation phase, VinciDecoder applies the trained translation model to generate forensic reports based on the suspicious paths of the provenance graph associated with the newly detected incident (Section 3.5). Optionally, VinciDecoder can generate reports using our rule-based mechanism (Section 3.5).

#### 3.2 Path to Intermediary Language Translation (PILT)

NMT is typically applied to textual sentences, which renders its application to provenance graphs challenging. To address this, the PILT module converts each suspicious path into a primitive sentence by querying the database to sequentially scan the nodes, extract their properties, and record them as one compound word of the sentence (see Fig. 3). Algorithm 1 details the mechanism of PILT as follows: PILT extracts the properties *type* and *user* from operation nodes and appends them to the created primitive sentence (line 3-5). Moreover, it calculates

<sup>1</sup> In Section 4.2, we discuss how we obtain more pairs of reports and paths for training.

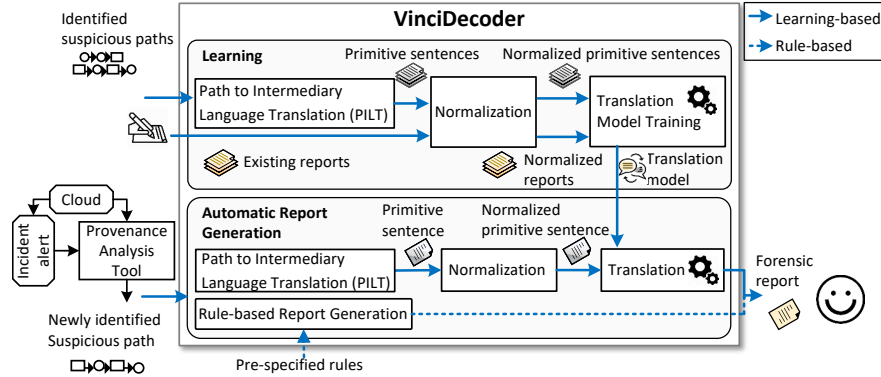


Fig. 3: Overview of VinciDecoder.

the *elapsed time* between the timestamp properties stored at two consecutive operation nodes (line 6-7) and appends the calculated value with a proper postfix (e.g., “-millisecond”, “-hours”, etc.) to the sentence (line 8-9). The elapsed time may be interesting for reporting the incidents where the attacker attempts to issue a large number of operations in a short period of time, e.g., to launch race condition or DoS attacks. PILT also records the *type* and the identifier of resources (line 10-13). In the next section, we detail how obtained sentences are modified and leveraged to train the translation model.

---

**Algorithm 1** Path to Intermediary Language Translation
 

---

**Input:** path  $\leftarrow$  Suspicious path identified by the provenance analysis tool

**Output:** SenRepresentingPath

1. **foreach** node  $\in$  path **do**
  2.   **if** isOperation(node) **then** %Appending the operation properties to sentence
  3.     OperationType  $\leftarrow$  node["data"]["OperationType"]
  4.     User  $\leftarrow$  node["data"]["user"]
  5.     SenRepresentingPath.append("type:" + OperationType + "user:" + User)
  6.     %Appending the approximate elapsed time between operations to sentence
  7.     **if** isNotFirstNode(node) **then**
  8.       ElapsedTime  $\leftarrow$  ThisOperation – PreviousOperationTime
  9.       ElapsedApprox  $\leftarrow$  ElapsedTimeApproximator (ElapsedTime)
  10.       SenRepresentingPath.append(ElapsedApprox)
  11.     **else if** isResource(node) **then**%Appending resource properties to sentence
  12.       ResourceType  $\leftarrow$  node["data"]["ResourceType"]
  13.       ResourceID  $\leftarrow$  node["data"]["ID"]
  14.       SenRepresentingPath.append("type:" + ResourceType + "ID:" + ResourceID)
  15. **return** SenRepresentingPath
- 

*Example 1.* Fig. 4 shows the translation of a path (left) into a primitive sentence (right) in our intermediary language. As we can see, the properties of each node (e.g., the node representing *CreatePort* operation) are represented by a word (e.g., “*type:CreatePort,user:non-admin*”) in the obtained sentence.

### 3.3 Normalization

To allow NMT to focus on generic words in the primitive sentences instead of application-specific ones (which may lead to mis-translation), VinciDecoder

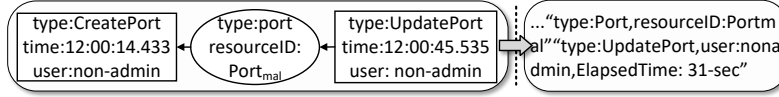


Fig. 4: Simplified example path (left) translated into a primitive sentence (right).

needs to remove instance-specific information from the dataset before feeding it to NMT. Specifically, the forensic reports and their corresponding primitive sentences used for training may contain values (e.g., the name/ID of resources) that are related to semantics of the specific scenarios (which NMT is not aware of). Retaining such values is known to reduce the quality of the reports generated by the trained neural translation model [43]. Therefore, VinciDecoder identifies and replaces all instance-specific values (e.g., the number preceding the string “-milliseconds”<sup>2</sup>) with a placeholder (i.e., \0), and the name of the applications or software platforms with the word “*platform*” based on our specified rules.

### 3.4 Translation Model Training

This module builds a translation model to profile the correspondence between the existing forensic reports and their associated suspicious paths. To this end, we leverage NMT [26], as it automatically captures the *context* of words and nodes (i.e., the dependencies between words in a report and nodes in a path) using embeddings. By applying NMT, VinciDecoder first projects words of a report and the words of the obtained primitive sentences (i.e., properties of nodes) into a high-dimensional numerical vector space such that words/nodes with similar contexts have closer vector representations. Next, VinciDecoder builds a translation model based on the derived vectors that optimally maps each provided forensic report to its paired primitive sentence.

*Example 2.* Fig. 5 shows an excerpt of the training dataset composed of the primitive sentences obtained from the suspicious paths (left) and their corresponding manually created reports (right). The semantically related information on each side are illustrated with the same type of lines.

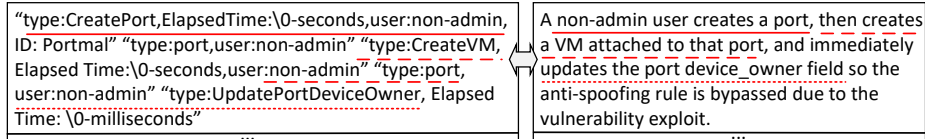


Fig. 5: Example paths in our intermediary language (left) and their corresponding manually written reports (right). Semantically related information are highlighted by the same type of lines.

### 3.5 Automatic Report Generation

Once a new security incident is detected, VinciDecoder automatically generates forensic reports based on the suspicious path identified by existing tools (e.g., [21, 54]) using our learning-based and rule-based techniques.

<sup>2</sup> Despite removing the numbers, the range of the elapsed time (e.g., milliseconds vs. hours) retains useful information about the incidents.

**Learning-based report generation.** After building the translation model in the learning phase, VinciDecoder can be applied to generate forensic reports based on the detected suspicious path. Specifically, VinciDecoder converts the suspicious path into a primitive sentence in our intermediary language, and removes the instance-specific information by following the same techniques as mentioned in Section 3.2 and Section 3.3. Next, it applies the translation model to each normalized primitive sentence to automatically generate the corresponding forensic report. To improve the quality of generated reports, VinciDecoder also allows the analyst to conduct post-editing [28] by identifying the instance-specific information using the primitive sentences and adding them to the reports.

**Rule-based report generation.** To ensure the applicability of our approach when there is a lack of a sufficient number of reports for training, VinciDecoder is also equipped with a rule-based mechanism, which enables translation without training data. Specifically, VinciDecoder sequentially scans nodes on each path, and extracts the following properties stored at each node: the *type* and *ID* of resources/operations, the *user* triggering an operation, and the *elapsed time* between the timestamp values stored in two consecutive operation nodes. Then, it creates an ordered list, where each item represents the properties of a node. Next, VinciDecoder generates sentences based on rules specified by the analyst, and it sequentially links the items such that the extracted user, resource, operation and elapsed time are included as the subject, object, verb and propositional phrase in generated sentences, respectively (detailed in Appendix). Finally, VinciDecoder generates an introductory and a concluding sentence to describe an overview of the incident (e.g., describing the time of the detection).

*Example 3.* Fig. 6 shows the report related to the incident in our motivating example (Section 1). The report starts with explaining the number of operations in the suspicious path, continues with describing the attack steps, and concludes with indicating the ID of nodes in the suspicious paths.

<p>By the detection time, there are 4 operations performed in 1 minute corresponding to the resource vmml. A nonadmin user created a port named portmal on a subnet. Once done, this user modified portdeviceowner after less than a minute. (S)He also created a vm named vmml on that port after less than a second. Then, (s)he modified that portdeviceowner after less than a second. More details can be found in the provenance graph in path [ 416 - 419 - 422 - 425 ].</p>
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Fig. 6: Automatically generated report on the incident discussed in our motivating example (Section 1).

## 4 Implementation and Evaluation

In this section, we detail the implementation of VinciDecoder and evaluate our solution.

### 4.1 Evaluation using Cloud Management-level Provenance Graphs

To evaluate VinciDecoder under different scenarios (e.g., various lengths of suspicious paths), we apply VinciDecoder to cloud management-level provenance graphs generated in our testbed cloud.



**4.1.1 Implementation and Data Collection** We implement VinciDecoder in a cloud testbed based on OpenStack [8] (a popular open-source cloud platform). We note that only our PILT module (Section 3.2) and our rules (Appendix) are platform-specific, and the modular design of VinciDecoder makes it easily portable to other platforms or provenance models (e.g., OS-level provenance [25]). We export provenance graphs from Neo4j [7] into JSON format for processing. We leverage Open-Source Toolkit for Neural Machine Translation (ONMT) [26] (a popular tool for language translation). Similar to some other solutions (e.g., [54]), we choose the default options for embedding paths (i.e., 500 dimensional vector) as well as the batch size and the dropout rate (i.e., 64 and 0.3, respectively). We leverage the metrics in [45] to evaluate our approach. We run VinciDecoder on an Ubuntu 20.04 server equipped with 128 GB of RAM. We generate the provenance graphs through deploying and updating different types of virtual resources. Moreover, we enrich our training dataset by leveraging the rule-based mechanism (detailed in Section 3.5). To simulate reports authored based on various writing styles, we specify rules capturing the writing styles of our different authors.

Table 1: Statistics of our testbed datasets.

Dataset	Training								Testing	
	D <sub>tr-size1</sub>	D <sub>tr-size2</sub>	D <sub>tr-size3</sub>	D <sub>tr-size4</sub>	D <sub>tr-len1</sub>	D <sub>tr-len2</sub>	D <sub>tr-len3</sub>	D <sub>tr-len4</sub>	D <sub>ts1</sub>	D <sub>ts2</sub>
# of paths	2000	4000	6000	8000	2000	2000	2000	2000	2000	2000
l <sub>min</sub>	4	4	4	4	4	8	12	16	4	8

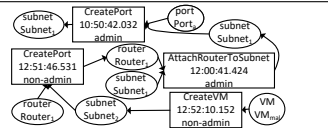
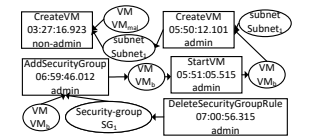
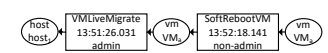

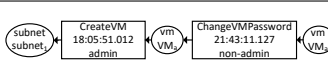
Table 1 shows the statistics of our datasets. To evaluate the effect of length and number of available samples (i.e., the suspicious paths) on the performance, we conduct our experiments based on two groups of training datasets: 1) varying the number of paths: four datasets (D<sub>tr-size1</sub> to D<sub>tr-size4</sub>) each consisting of a different number of paths with the same minimum length; 2) varying the length of paths: four datasets (D<sub>tr-len1</sub> to D<sub>tr-len4</sub>) consisting of the same number of paths with different specified minimum lengths. We randomly select 70% and 30% of the paths from each training dataset to build and validate (used by NMT to automatically tune the hyperparameters in training [22]) the models, respectively. Our training and testing datasets are selected from disjoint parts of the provenance graph, so we can evaluate the ability of VinciDecoder in handling unseen datasets. We also evaluate our approach based on two testing datasets with paths of different minimum lengths as shown in Table 1.

**4.1.2 Effectiveness Evaluation** We reproduce in our testbed eight real-world incident scenarios that involve cloud management operations, and apply VinciDecoder to generate reports based on the captured provenance graphs. Table 2 shows those scenarios and corresponding incidents. Most of those scenarios are discussed in previous works (e.g., [14, 34, 48, 50, 55]) focusing on security verification. For all cases, our generated reports capture all operations that led to the incidents. Table 3 demonstrates the effectiveness of VinciDecoder based on five scenarios. We also showcased our result for the sixth scenario in Section 3.5. The other two scenarios (Table 2, seventh and eighth rows) involve fewer types of operations and are thus omitted due to space limitation.

Table 2: Attack scenarios used to evaluate the effectiveness of VinciDecoder.

Index	Root cause	Incident
1	Improper authorization [55]	Port Scanning
2	Failed update of security groups [48]	Data leakage
3	Soft-rebooting migrated VM [5]	Data corruption
4	Deleting resized VM [4]	Disk utilization
5	Incorrect role assignment [50]	Data leakage
6	Race condition in update port [48]	Data leakage
7	Wrong VLAN ID [14]	Data leakage
8	Excessive VM creation on a host [34]	Disk utilization

Table 3: Reports generated by VinciDecoder for five scenarios in Table 2. The sixth scenario is showcased in Section 3.5.

Index	Provenance graph path	Automatically generated report
1		An admin user created a port named porta on a subnet. This admin user attached that subnet to a router after around 1 hours. A nonadmin user created a port on that subnet and on that router, previously affected by a different user. After that, (s)he created a vm named vmmal, which is associated to the alert.
2		A nonadmin user created vm named vmmal on a subnet. An admin user created a vm named vmb on that subnet. Once done, (s)he started that vm vmb. Then attached a securitygroup named SG1 on that vm. The administrator deleted securitygroup rule from that SecurityGroup after around 1 minutes.
3		An admin user livemigrated a vm named vma to a host. A nonadmin user softbooted that vm, previously affected by a different user.
4		A nonadmin user created a vm named vma on a host. Later, (s)he resized that vm after less than a minutes. Next, (s)he deleted that vm after less than a minute.
5		An admin user created a vm, named vma on a subnet. A nonadmin user changed password a vm, previously affected by a different user after around 3 hours.

**Example of verifying the captured information.** Fig. 7(a) shows the automatically generated report explaining the operations (i.e., the creation of a rogue port on a router created by a different user) to exploit a vulnerability [2] that led to the attack on  $VM_a$  (Table 2, first row). VinciDecoder correctly details the steps described by the manually created report shown in Fig. 7(b).

**4.1.3 Performance Evaluation** We showcase the high quality of generated reports with different number and length of paths in training datasets based on well known translation metrics BLEU and ROUGE [45]. BLEU (*precision*) measures the fraction of the generated information that are relevant to the manually written reports and ROUGE (*recall*) indicates the fraction of information from the reference reports that are included in automatically generated reports. As VinciDecoder proposes the first learning-based provenance translation solution, we cannot directly compare our results to existing works, while we note that scores above 0.5 are generally known to reflect high quality translations [29].

**Number of training samples.** Fig. 8(a) shows that, in most cases, there is a minor variation in the evaluated performance as the number of the training samples increases. This can be explained by the possibility that our translation models trained by larger datasets may become more biased [33] due to the

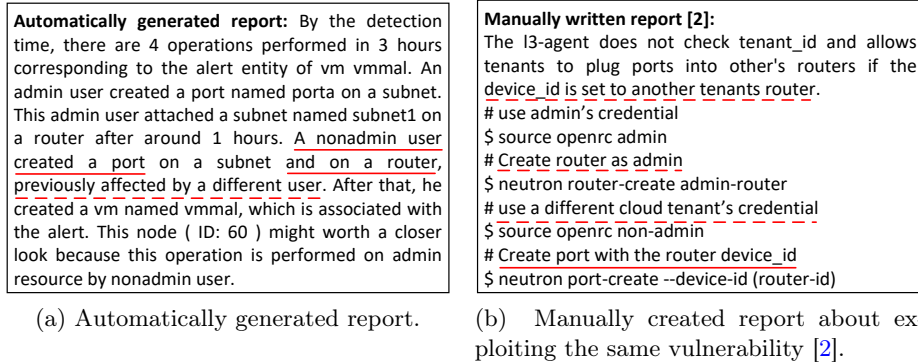


Fig. 7: Verifying the information captured by our generated report. The semantically relevant information are highlighted with the same type of line.

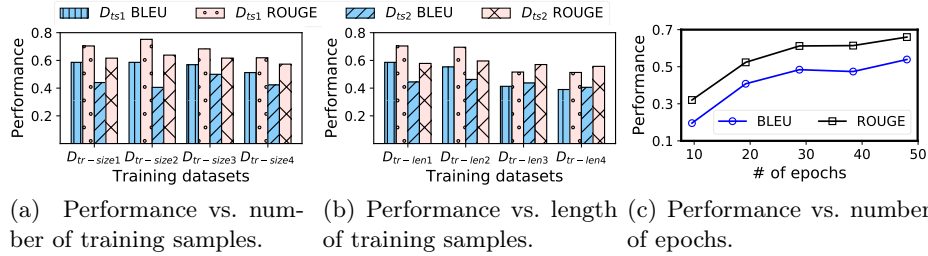


Fig. 8: Evaluation with cloud management-level provenance graphs.

frequent appearances of similar patterns of cloud management operations. To further illustrate this effect, Fig. 9 compares an excerpt of a manually written report of a path with the ones generated by VinciDecoder based on the four training datasets. As we can see, larger training datasets (e.g.,  $D_{tr-size4}$ ) cause more extra or missing information in the generated reports. We conclude that our approach remains useful even with a limited number of training samples.

**Length of training samples.** Fig. 8(b) shows that the performance decreases when the length of paths in the training dataset increases, which may be due to the degraded performance of NMT for longer sentences [18]. The reduction is more significant for  $D_{ts1}$  due to the difference between the length of paths in this testing dataset (with minimum four nodes) and that of paths in the training datasets,  $D_{tr-len3}$  and  $D_{tr-len4}$  (with minimum 12 and 16 nodes). The training datasets  $D_{tr-len1}$  and  $D_{tr-len2}$  (with paths of minimum four and eight nodes) cause a noticeably higher performance for  $D_{ts1}$  than for  $D_{ts2}$  due to the general positive impact of shorter paths of  $D_{ts1}$  on the performance and the similarity between training and testing datasets regarding the lengths of paths.

**Number of epochs.** Fig. 8(c) shows that the performance is significantly improved with the number of epochs (i.e., the number of times NMT iterates through a training dataset). We also measure the perplexity (i.e., the extent a trained model could predict a newly provided data [15]) and the accuracy for different numbers of epochs and training datasets. Fig. 10(a) shows that the

<b>Path (converted to a sentence in the intermediary language):</b> "type:CreateServers,user:Admin, ElapsedTime:\0-seconds" "type:port" "type:UpdatePorts,user:Admin,ElapsedTime:\0-seconds"	
<b>Manually written report:</b> An admin user created a server attached to a port, named \0. He later updates that port after less than a minute.	
<b>D<sub>tr-size1</sub>:</b> An Admin user created a server named \0 attached to that port. Once done, he updated a port named \0 [missing elapsed time].	<b>D<sub>tr-size2</sub>:</b> An Admin user created a server named \0 attached to that port. [not using pronoun] The Admin user modified a port named \0, after less than a minute.
<b>D<sub>tr-size3</sub>:</b> [missing create server] An Admin user updated a port named \0. <del>The administrator updated a port named \0, after around \0minutes.</del>	
<b>D<sub>tr-size4</sub>:</b> An Admin user <del>created a port named \0 connected to that subnet. Then created a port named \0 connected to that subnet. This Admin user created a port named \0 connected to that subnet after around \0 minutes. Once done, he created a port named \0 connected to that subnet. He</del> created a server named \0 attached to that port after less than a minute. [not using pronoun] This Admin user modified a port named \0 after less than a minute. <del>Then updated a port named \0.</del>	

Fig. 9: Comparing reports generated by training datasets with different numbers of samples (irrelevant parts of the generated reports are crossed out).

perplexity decreases to around 1.2 after 20 epochs, and Fig. 10(b) shows that the accuracy increases to around 97% after 40 epochs. We conclude that the perplexity and accuracy of VinciDecoder improve with the number of epochs, and reach almost constant values after training over maximum 40 epochs.

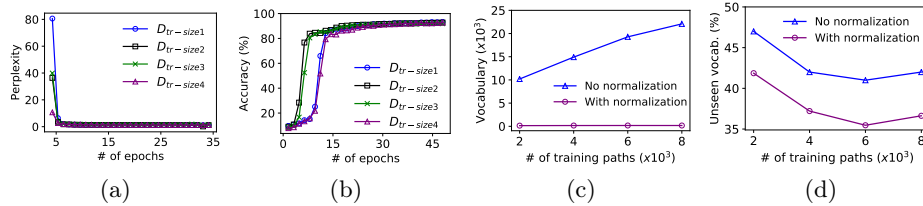


Fig. 10: (a) Perplexity (the smaller is better) and (b) accuracy at different epochs; (c) the growth of vocabulary size, and (d) the proportion of unseen words.

**Out-of-vocabulary evaluation.** Fig. 10(c) shows that, without normalization (Section 3.3), the size of the vocabulary significantly grows with the size of the dataset, which may subsequently reduce the performance. Furthermore, Fig. 10(d) shows that, on average, the proportion of unseen words in the testing dataset (i.e., words that do not exist in the training datasets, and thus may be translated incorrectly) is around 6% less after conducting normalization. This shows that our normalization technique effectively increases the applicability of the trained models for describing new provenance graphs in testing datasets. In summary, our results demonstrate the feasibility and quality of the produced reports for datasets with different number and length of paths.

## 4.2 Large Scale Experiments using CVE-based Provenance Graphs

As our evaluation in Section 4.1 is limited to the data collected from our testbed, to evaluate our approach based on more realistic and larger scale datasets, we apply VinciDecoder to CVE-based provenance graphs in this section.

**Data Collection.** The performance of NMT may be adversely affected by the scarce available pairs of input data [19]. Therefore, to enrich our dataset, we adopt an approach similar to recent works (e.g., [13, 20, 44]) on extracting provenance graphs from cyber threat intelligence (CTI) reports such as vulnerability databases [6]. Similar to such solutions, we leverage a combination of rule-based and machine learning techniques (e.g., Part-of-Speech Tagging [17]) to extract different components of provenance graphs (e.g., affected systems, attackers’ activities, and the impact of attacks), which allows us to generate a large number of provenance graphs paired with their CTI reports to train our translation model. To this end, we processed 60,000 CVE entries. Inspired by existing solutions (e.g., [44]), to decrease the verbosity of CVE entries and facilitate extracting provenance information, we apply a summarization technique<sup>3</sup> to the entries, and subsequently, extract provenance metadata. Finally, we clean the dataset by removing the entries from which the attackers’ activities and impact cannot be extracted, and we obtain six datasets with the total number of 40,151 entries as shown in Table 4. We randomly select 80%, 10% and 10% of entries in each dataset for training, testing and validation, respectively.

Table 4: Statistics of our datasets prepared with CVE entries.

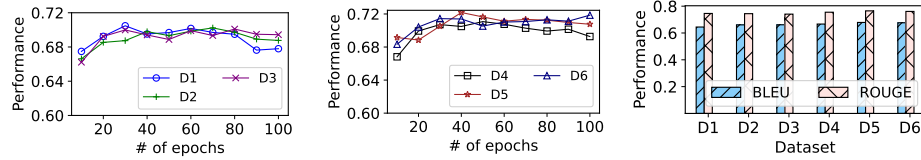
	D <sub>1</sub>	D <sub>2</sub>	D <sub>3</sub>	D <sub>4</sub>	D <sub>5</sub>	D <sub>6</sub>
<b>Total before cleaning</b>	30000	36000	42000	48000	54000	60000
<b>Total after cleaning</b>	20626	25188	28575	32333	36283	40151
<b>Training</b>	16600	20271	22997	26022	29201	32314
<b>Validation</b>	2060	2514	2852	3227	3621	4007
<b>Testing</b>	1966	2403	2726	3084	3461	3830

**Number of epochs.** We showcase the high quality of our generated reports by first identifying the number of epochs that yields the highest performance (average BLEU and ROUGE scores) for each dataset. Fig. 11(a) shows that VinciDecoder achieves higher performance with smaller datasets after a fewer number of epochs (e.g., 30 epochs for D1). This can be explained by the possibility that training on smaller datasets for a larger number of epochs would cause overfitting [38], which decreases the performance. On the other hand, the performance related to our larger datasets (D4, D5 and D6 in Fig. 11(b)) remains high for a larger number of epochs. For instance, we maximise the performance by training on our largest dataset (D6) for 100 epochs.

**Number of training samples.** We measure the performance of VinciDecoder trained with different datasets for the number of epochs that achieved the highest performance in Fig. 11(a) and 11(b) (e.g., 30 and 100 epochs for D1 and D6, respectively). Fig. 11(c) shows that both the BLEU and ROUGE scores remain almost similar and above 0.68 and 0.74, respectively, for all datasets. This shows that despite the complex content and various writing styles that are natural to CVE reports, VinciDecoder performs well in generating such reports<sup>4</sup>.

<sup>3</sup> <https://pypi.org/project/nlpaug/>

<sup>4</sup> Note that while both sets of our experiments in Section 4.1 and 4.2 show high quality reports, directly comparing their results is not meaningful as their reports



(a) Performance vs. # of epochs (smaller datasets). (b) Performance vs. # of epochs (larger datasets). (c) Performance vs. size of datasets.

Fig. 11: Evaluation with CVE-based provenance graphs.

### 4.3 User-based Study

To evaluate the quality and usefulness of our generated reports in helping human analysts, we conduct a user study<sup>5</sup> based on standard practices [10], where participants have to evaluate the factual correctness and fluency of the reports generated by VinciDecoder. Our participants include eight cybersecurity researchers working in a major telecommunication organization and five graduate researchers working in cybersecurity labs of our university. Table 5 shows the percentage of participants in each group, their reported level of expertise, and the average score for all statements.

Table 5: Average quantified agreement levels for each group (scores will be explained later). PG means provenance analysis. (A), (L), and (N) signs represent advanced, little and no knowledge, respectively, as reported by the participants.

	Industry					Academia
Background (Cloud-PG)	A-A	A-L	A-N	L-L	L-N	A-L
Participants (%)	15	23	8	8	8	38
Scores (out of 5)	3.83	4.28	3.83	3.83	3	4.13

At the beginning of the study, we show an attack scenario (our motivating example in Section 1) to the participants. Next, we provide the participants with the provenance graph, the report generated by VinciDecoder, and the manually written report. Our study asks participants to evaluate their investigation with and without VinciDecoder, and accordingly express their level of agreement with the provided statements (shown in Table 6) by choosing one of the following options: *Strongly agree*, *Agree*, *Neutral*, *Disagree* and *Strongly disagree*. We then quantify the results by assigning an integer between one and five to each option, where five means *Strongly agree* and one means *Strongly disagree*.

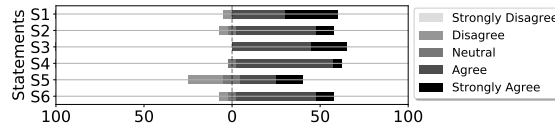


Fig. 12: Participants' agreement with statements in Table 6.

Fig. 12 shows the distribution of participants' agreement with each statement. For most participants, understanding the attack steps is much easier using our

are of incomparable lengths (e.g., cloud management-level provenance graph-based reports are typically longer which has a negative effect on the performance).

<sup>5</sup> This study has been identified as quality assurance by Research Ethics/Office of Research of our university, which means it requires no ethics approval.

Table 6: Survey statements and scores. The agreement level of participants are converted to scores between one and five (score five represents *Strongly agree*).

	Statement	Score
S1	Understanding the attack steps using the generated text is easier than using the path.	4.3
S2	The generated text is consistent with the explained attack scenario.	3.92
S3	The generated text is consistent with the path regarding the relationships of operations.	4.31
S4	The generated text captures all the information of the suspicious path.	4
S5	The generated text is sufficiently fluent compared with the manually written report.	3.46
S6	The generated text is consistent with the manually written report regarding attack steps.	3.92

generated report than the provenance graph (S1). According to most participants, our generated report contains no information contradicting the described attack scenario and the provenance graph (S2 and S3). Additionally, the results (S4) affirm that all the information captured by the provenance graph is reflected in our generated report. Most users find the generated report almost as fluent as the manually created one (S5), while the slightly lower fluency is expected for automatically generated reports [28]. Finally, the attack steps described by the generated report is consistent with the report created by the human analyst (S6). We show the average quantified scores in Table 5. VinciDecoder achieves scores above three among all groups despite their low level (little or no) of expertise, which confirm the benefits of VinciDecoder to users in investigating incidents.

## 5 Discussion

In this section, we discuss future directions and limitations of VinciDecoder.

**Application to other models.** Our approach is generic enough to support various provenance models (e.g., [25,37] and [53] for the OS and Internet of things environments, respectively) after converting paths into primitive sentences capturing both nodes and edges as words in our intermediary language. Likewise, an interesting future direction is to apply VinciDecoder to other graphical security models such as attack graphs [13] paired with their corresponding textual interpretation.

**Coverage.** In this work, we leverage NMT for generating forensic reports from long suspicious paths, as it is known to perform well in translating long sentences [46]. In our future work, we will further investigate the possibility of applying other translation techniques [31] that may increase the performance of VinciDecoder. Finally, our goal is to assist analysts, instead of replacing them, by allowing them to focus on more important but light-weight tasks, e.g., validating the report to ensure its legal value.

## 6 Related Work

Provenance-based security solutions have been extensively explored [25,40,41,47,53]. King et al. [25] propose data provenance to investigate security incidents in operating systems. ProvDetector [54] is a provenance solution to detect anomalous programs using embedded sentences representing paths. Poirot [36] identifies attack-related subgraphs, and SteinerLog [12] detects attack campaigns across multiple hosts using alert correlation. Some of recent solutions (e.g., [37,58])

focus on increasing the interpretability of provenance graphs. ATLAS [9] adopts sequence learning to model the signature of attacks. There exist efforts adapting provenance analysis to domains other than operating systems such as the Internet of Things (IoT) (e.g., [53]) and SDN environments (e.g., [51, 52]). Wu et al. [56] propose an approach explaining the absence of events. The authors in [32] and [11] propose a provenance-based investigation and access control scheme for clouds, respectively. The authors in [39], propose a solution to enhance the access control mechanism in OpenStack. Chen et. al [16] propose CLARION to capture precise provenance graphs across namespaces of different containers. Unlike our work, none of those solutions generates a human-readable description of the provenance graph, and our approach can be applied to most of those solutions to automatically translate their results into natural language reports.

Several solutions [27, 30, 42] have been proposed to generate human-readable descriptions based on non-linguistic information. The authors in [42] propose a solution to generate textual summaries about basketball games based on tables of information using NMT. [30] is a neural text generation solution to generate the first sentence of a Wikipedia entry based on a provided *infobox*. Finally, [27] proposes a solution that generates abstracts for scientific papers (with the BLEU score of around 0.14) based on paired titles and knowledge graphs (with 4.43 edges, on average). None of those solutions are designed for generating forensic reports based on typically larger and more complex provenance graphs that are natural to the security context or cloud scale. ProvTalk [49] proposes a rule-based approach for generating textual summaries of provenance graphs, which is generalized and complemented with a learning-based approach in VinciDecoder.

## 7 Conclusion

In this paper, we presented VinciDecoder, the first solution for automatically translating provenance analysis results into human-readable forensic reports using both rule-based and learning-based techniques. To this end, we first explored the characteristics of the provenance graph to represent it in an intermediary language, which can then be translated into a natural language. We showed the feasibility of our approach by implementing VinciDecoder based on an OpenStack cloud, and demonstrated the high quality of generated reports for real-world incident scenarios using both numerical (up to 0.56 and 0.68 BLEU scores for cloud management-level and CVE-based provenance graphs, respectively) and user-based evaluations. As future work, we will integrate VinciDecoder with other (e.g., OS-level) provenance analysis tools. We will also explore other translation techniques and hyperparameters (i.e., the size of embedding vectors and batch size), which may further improve the effectiveness of our approach.

**Acknowledgment.** We thank the anonymous reviewers for their valuable comments. This work was supported by the Natural Sciences and Engineering Research Council of Canada and Ericsson Canada under the Industrial Research Chair in SDN/NFV Security and the Canada Foundation for Innovation under JELF Project 38599.



## Appendix

Algorithm 2 shows our rule-based mechanism generating reports based on the cloud management-level provenance graphs (e.g., the provenance graph in Fig. 1). To generate fluent sentences, we specify rules for indicating different subjects (line 2-5). We add resources extracted from the names of operations (e.g., a *VM* in *CreateVM*) through the template *a \$resource\_type named \$main\_resource\_name* (line 7-9). We specify various rules (line 11-20) for describing other affected resources connected to an operation node. We also specify rules to record other information such as the elapsed time between operations (line 21-26). Through such rules specifically designed for each type of operations, resources, and users, VinciDecoder generates reports when there is an insufficient amount of training data for generating high quality reports.

---

### Algorithm 2 Rule-based Report Generation

---

**Input:** path  $\leftarrow$  Suspicious path identified by the provenance analysis tool  
 Middle\_Sentence\_Subjects = ["Next, this user", "Later, he/she", "He/She also", "This user then", "Once done, he/she "]

**Output:** Description

1. **foreach** node  $\in$  path **do**
2.   **if** isFirstNode(node) or isNotEqualPreviousUser(node) **then** %User of the first operation
3.     Subj\_main  $\leftarrow$  Admin\_NonAdmin\_Specifier(userID, adminID)
4.   **else** %Other users with prior words (e.g., "Once done, he/she")
5.     Subj\_main  $\leftarrow$  random\_choice(Middle\_Sentence\_Subjects)
6.   **if** isAnOperation(node) **then**
7.     Verb, MainObject  $\leftarrow$  OperationType.split(operation)
8.     MainObject  $\leftarrow$  MainObject.setDeterminer("a")
9.     MainObject  $\leftarrow$  addAfter("with the ID " + MainObject["id"])
10.    OtherAffectedResources  $\leftarrow$  EndOfOutgoingEdges(node)
11.    **foreach** SecondaryObject  $\in$  OtherAffectedResources: %Choosing prior words
12.     **if** verb != "delete" **then**
13.       SecondaryObject.addBefore("on").addAfter(resource["id"])
14.     **else**
15.       SecondObject.addBefore("from a").addAfter(resource["id"])
16.     **if** previousUser(resource) != operation["user"] **then** %Update by a different user
17.       SecondaryObject.addAfter(", previously affected by a different user,")
18.     **if** isNotFirstNode(node) **then** %Range of elapsed time between operations
19.       ElapsedTime  $\leftarrow$  TimeRangeDescriptor(ThisOperation - PreviousOperationTime)
20.     **if** isAlertNode(node) **then**
21.       MainObject.addAfter(", which is associated to the alert.")
22.     sentence.setSubj(Subj\_main).setVerb(Verb).setObj(MainObject) %Form sentence
23.     sentence.addAfter(SecondaryObject).addAfter(ElapsedApprox)
24.     **if** ThisOperation = PreviousOperation **then** %Emphasize the repetition
25.       sentence.addComponent("again")
26.     PathDescription.append(sentence)
27. **return** Description

---

## References

1. Cisco AVOS, accessed July 28, 2022. <https://github.com/CiscoSystems/avos>
2. CVE-2014-0056, accessed July 28, 2022, <https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-0056/>

3. CVE-2015-5240, accessed July 28, 2022, <https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-5240>
4. CVE-2016-7498, accessed July 28, 2022, <https://nvd.nist.gov/vuln/detail/CVE-2016-7498>
5. CVE-2020-17376, accessed July 28, 2022, <https://bugs.launchpad.net/nova/+bug/1890501>
6. CVE details, accessed June 14, 2022, <https://www.cvedetails.com/vulnerability-list/>
7. Neo4j Graph Platform, accessed July 28, 2022. <https://neo4j.com/>
8. OpenStack, accessed July 28, 2022. <https://www.openstack.org/>
9. Alsaheel, A., Nan, Y., Ma, S., Yu, L., Walkup, G., Celik, Z.B., Zhang, X., Xu, D.: ATLAS: A Sequence-based Learning Approach for Attack Investigation. In: *USENIX Security*. pp. 3005–3022 (2021)
10. Assila, A., Ezzedine, H., et al.: Standardized Usability Questionnaires: Features and Quality Focus. *eJCIST* (2016)
11. Bates, A., Mood, B., Valafar, M., Butler, K.R.B.: Towards Secure Provenance-based Access Control in Cloud Environments. In: *CODASPY*. pp. 277–284 (2013)
12. Bhattarai, B., Huang, H.: SteinerLog: Prize Collecting the Audit Logs for Threat Hunting on Enterprise Network. In: *ASIA CCS*. pp. 97–108 (2022)
13. Binyamini, H., Bitton, R., Inokuchi, M., Yagyu, T., Elovici, Y., Shabtai, A.: A Framework for Modeling Cyber Attack Techniques from Security Vulnerability Descriptions. In: *KDD*. p. 2574–2583 (2021)
14. Bleikertz, S., Vogel, C., Groß, T., Mödersheim, S.: Proactive Security Analysis of Changes in Virtualized Infrastructures. In: *ACSAC*. pp. 51–60. ACM (2015)
15. Chen, S.F., Goodman, J.: An empirical study of smoothing techniques for language modeling. *Computer Speech & Language* **13**(4), 359–394 (1999)
16. Chen, X., Irshad, H., Chen, Y., Gehani, A., Yegneswaran, V.: CLARION: Sound and Clear Provenance Tracking for Microservice Deployments. In: *USENIX Security*. pp. 3989–4006 (2021)
17. Chiche, A., Yitagesu, B.: Part of speech tagging: a systematic review of deep learning and machine learning approaches. *Journal of Big Data* **9**(1), 1–25 (2022)
18. Cho, K., van Merriënboer, B., Bahdanau, D., Bengio, Y.: On the Properties of Neural Machine Translation: Encoder–Decoder Approaches. In: *SSST*. pp. 103–111. ACL (2014)
19. Fadaee, M., Bisazza, A., Monz, C.: Data Augmentation for Low-Resource Neural Machine Translation. In: *ACL*. pp. 567–573 (2017)
20. Gao, P., Shao, F., Liu, X., Xiao, X., Qin, Z., Xu, F., Mittal, P., Kulkarni, S.R., Song, D.: Enabling Efficient Cyber Threat Hunting with Cyber Threat Intelligence. In: *ICDE*. pp. 193–204. IEEE (2021)
21. Hassan, W.U., Aguse, L., Aguse, N., Bates, A., Moyer, T.: Towards Scalable Cluster Auditing Through Grammatical Inference over Provenance Graphs. In: *NDSS* (2018)
22. He, D., Lu, H., Xia, Y., Qin, T., Wang, L., Liu, T.Y.: Decoding with Value Networks for Neural Machine Translation. *Advances in Neural Information Processing Systems* **30** (2017)
23. Hochreiter, S., Schmidhuber, J.: Long Short-Term Memory. *Neural computation* **9**(8), 1735–1780 (1997)
24. Johnson, C., Badger, L., Waltermire, D., Snyder, J., Skorupka, C., et al.: Guide to cyber threat information sharing. NIST special publication **800**(150) (2016)
25. King, S.T., Chen, P.M.: Backtracking Intrusions. In: *SOSP*. pp. 223–236 (2003)

26. Klein, G., Kim, Y., Deng, Y., Senellart, J., Rush, A.: OpenNMT: Open-Source Toolkit for Neural Machine Translation. In: Proceedings of ACL, System Demonstrations. pp. 67–72. ACL (2017)
27. Koncel-Kedziorski, R., Bekal, D., Luan, Y., Lapata, M., Hajishirzi, H.: Text Generation from Knowledge Graphs with Graph Transformers. In: NAACL (2019)
28. Läubli, S., Sennrich, R., Volk, M.: Has Machine Translation Achieved Human Parity? A Case for Document-level Evaluation. In: EMNLP. pp. 4791–4796. ACL (2018)
29. Lavie, A.: Evaluating the Output of Machine Translation Systems. AMTA Tutorial **86** (2010)
30. Lebrecht, R., Grangier, D., Auli, M.: Neural Text Generation from Structured Data with Application to the Biography Domain. In: EMNLP. pp. 1203–1213. ACL (2016)
31. Lopez, A.: Statistical Machine Translation. ACM Computing Surveys (CSUR) **40**(3), 1–49 (2008)
32. Lu, R., Lin, X., Liang, X., Shen, X.S.: Secure Provenance: The Essential of Bread and Butter of Data Forensics in Cloud Computing. In: ASIA CCS. pp. 282–292 (2010)
33. L’Heureux, A., Grolinger, K., Elyamany, H.F., Capretz, M.A.M.: Machine learning with big data: Challenges and approaches. IEEE Access **5**, 7776–7797 (2017). <https://doi.org/10.1109/ACCESS.2017.2696365>
34. Madi, T., Zhang, M., Jarraya, Y., Alimohammadifar, A., Pourzandi, M., Wang, L., Debbabi, M.: QuantiC: Distance Metrics for Evaluating Multi-Tenancy Threats in Public Cloud. In: CloudCom. pp. 163–170. IEEE (2018)
35. Miao, H., Deshpande, A.: Understanding Data Science Lifecycle Provenance via Graph Segmentation and Summarization. In: ICDE. pp. 1710–1713. IEEE (2019)
36. Milajerdi, S.M., Eshete, B., Gjomemo, R., Venkatakrisnan, V.: Poirot: Aligning Attack Behavior with Kernel Audit Records for Cyber Threat Hunting. In: CCS. pp. 1795–1812 (2019)
37. Milajerdi, S.M., Gjomemo, R., Eshete, B., Sekar, R., Venkatakrisnan, V.N.: HOLMES: Real-Time APT Detection through Correlation of Suspicious Information Flows. In: IEEE S&P. pp. 1137–1152 (2019)
38. Mitchell, T.M.: Machine Learning. McGraw-hill New York (1997)
39. Nguyen, D., Park, J., Sandhu, R.: Adopting Provenance-based Access Control in OpenStack Cloud IaaS. In: NSS. pp. 15–27. Springer (2014)
40. Pasquier, T., Han, X., Goldstein, M., Moyer, T., Eysers, D., Seltzer, M., Bacon, J.: Practical Whole-System Provenance Capture. In: SoCC. pp. 405–418 (2017)
41. Pasquier, T., Han, X., Moyer, T., Bates, A., Hermant, O., Eysers, D., Bacon, J., Seltzer, M.: Runtime Analysis of Whole-System Provenance. In: CCS. pp. 1601–1616. ACM (2018)
42. Puduppully, R., Dong, L., Lapata, M.: Data-to-Text Generation with Content Selection and Planning. In: AAAI. vol. 33, pp. 6908–6915 (2019)
43. Santana, M.A.B., Ricca, F., Cuteri, B.: Reducing the Impact of out of Vocabulary Words in the Translation of Natural Language Questions into SPARQL Queries. arXiv preprint arXiv:2111.03000 (2021)
44. Satvat, K., Gjomemo, R., Venkatakrisnan, V.: EXTRACTOR: Extracting Attack Behavior from Threat Reports. In: EuroS&P. pp. 598–615. IEEE (2021)
45. Sharma, S., El Asri, L., Schulz, H., Zumer, J.: Relevance of Unsupervised Metrics in Task-Oriented Dialogue for Evaluating Natural Language Generation. CoRR **abs/1706.09799** (2017)

46. Sutskever, I., Vinyals, O., Le, Q.V.: Sequence to Sequence Learning with Neural Networks. *Advances in neural information processing systems* **27** (2014)
47. Tabiban, A., Jarraya, Y., Zhang, M., Pourzandi, M., Wang, L., Debbabi, M.: Catching Falling Dominoes: Cloud Management-Level Provenance Analysis with Application to OpenStack. In: *CNS*. pp. 1–9. IEEE (2020)
48. Tabiban, A., Majumdar, S., Wang, L., Debbabi, M.: PERMON: An Openstack Middleware for Runtime Security Policy Enforcement in Clouds. In: *CNS*. pp. 1–7. IEEE (2018)
49. Tabiban, A., Zhao, H., Jarraya, Y., Pourzandi, M., Zhang, M., Wang, L.: ProvTalk: Towards Interpretable Multi-level Provenance Analysis in Networking Functions Virtualization (NFV). In: *NDSS* (2022)
50. Thirunavukkarasu, S.L., Zhang, M., Oqaily, A., Chawla, G.S., Wang, L., Pourzandi, M., Debbabi, M.: Modeling NFV Deployment to Identify the Cross-level Inconsistency Vulnerabilities. In: *CloudCom*. pp. 167–174. IEEE (2019)
51. Ujcich, B.E., Jero, S., Edmundson, A., Wang, Q., Skowrya, R., Landry, J., Bates, A., Sanders, W.H., Nita-Rotaru, C., Okhravi, H.: Cross-App Poisoning in Software-Defined Networking. In: *CCS*. pp. 648–663 (2018)
52. Wang, H., Yang, G., Chinprutthiwong, P., Xu, L., Zhang, Y., Gu, G.: Towards Fine-grained Network Security Forensics and Diagnosis in the SDN Era. In: *CCS*. pp. 3–16. ACM (2018)
53. Wang, Q., Hassan, W.U., Bates, A., Gunter, C.: Fear and Logging in the Internet of Things. In: *NDSS* (2018)
54. Wang, Q., Hassan, W.U., Li, D., Jee, K., Yu, X., Zou, K., Rhee, J., Chen, Z., Cheng, W., Gunter, C., et al.: You Are What You Do: Hunting Stealthy Malware via Data Provenance Analysis. In: *NDSS* (2020)
55. Wang, Y., Madi, T., Majumdar, S., Jarraya, Y., Alimohammadifar, A., Pourzandi, M., Wang, L., Debbabi, M.: TenantGuard: Scalable Runtime Verification of Cloud-Wide VM-Level Network Isolation. In: *NDSS* (2017)
56. Wu, Y., Zhao, M., Haeberlen, A., Zhou, W., Loo, B.T.: Diagnosing Missing Events in Distributed Systems with Negative Provenance. In: *ACM SIGCOMM*. pp. 383–394 (2014)
57. Yusif, S., Hafeez-Baig, A.: A Conceptual Model for Cybersecurity Governance. *Journal of Applied Security Research* **16**(4), 490–513 (2021)
58. Zeng, J., Chua, Z.L., Chen, Y., Ji, K., Liang, Z., Mao, J.: WATSON: Abstracting Behaviors from Audit Logs via Aggregation of Contextual Semantics. In: *NDSS* (2021)