

SYNAT: Enhancing Security Knowledge Bases via Automatic Synthesizing Attack Tree from Crowd Discussions

ZIYOU JIANG*, State Key Laboratory of Intelligent Game, China, Science and Technology on Integrated Information System Laboratory, Institute of Software Chinese Academy of Sciences, China, and University of Chinese Academy of Sciences, China

LIN SHI*, School of Software, Beihang University, China

GUOWEI YANG, School of Information Technology and Electrical Engineering, The University of Queensland, Australia

XUYAN MA, State Key Laboratory of Intelligent Game, China, Science and Technology on Integrated Information System Laboratory, Institute of Software Chinese Academy of Sciences, China, and University of Chinese Academy of Sciences, China

FENGLONG LI, Huawei Cloud Computing Technologies CO., LTD., China

QING WANG[†], State Key Laboratory of Intelligent Game, China, Science and Technology on Integrated Information System Laboratory, Institute of Software Chinese Academy of Sciences, China, and University of Chinese Academy of Sciences, China

Cyber attacks have become a serious threat to the security of software systems. Many organizations have built their security knowledge bases to safeguard against attacks and vulnerabilities. However, due to the time lag in the official release of security information, these security knowledge bases may not be well maintained, and using them to protect software systems against emergent security risks can be challenging. On the other hand, the security posts on online knowledge-sharing platforms contain many crowd security discussions and the knowledge in those posts can be used to enhance the security knowledge bases. This paper proposes SYNAT, an automatic approach to synthesize attack trees from crowd security posts. Given a security post, SYNAT first utilize the Large Language Model (LLM) and prompt learning to restrict the scope of sentences that may contain attack information; then it utilizes a transition-based event and relation extraction model to extract the events and relations simultaneously from the scope; finally, it applies heuristic rules to synthesize the attack trees with the extracted

*Both authors contribute equally.

[†]Corresponding author.

Authors' addresses: **Ziyou Jiang**, State Key Laboratory of Intelligent Game, Beijing, China and Science and Technology on Integrated Information System Laboratory, Institute of Software Chinese Academy of Sciences, Beijing, China and University of Chinese Academy of Sciences, Beijing, China, ziyou2019@iscas.ac.cn; **Lin Shi**, School of Software, Beihang University, Beijing, China, shilin@buaa.edu.cn; **Guowei Yang**, School of Information Technology and Electrical Engineering, The University of Queensland, Brisbane, Queensland, Australia, guowei.yang@uq.edu.au; **Xuyan Ma**, State Key Laboratory of Intelligent Game, Beijing, China and Science and Technology on Integrated Information System Laboratory, Institute of Software Chinese Academy of Sciences, Beijing, China and University of Chinese Academy of Sciences, Beijing, China, maxuyan2021@iscas.ac.cn; **Fenglong Li**, Huawei Cloud Computing Technologies CO., LTD., Beijing, China, lifenglong4@huawei.com; **Qing Wang**, State Key Laboratory of Intelligent Game, Beijing, China and Science and Technology on Integrated Information System Laboratory, Institute of Software Chinese Academy of Sciences, Beijing, China and University of Chinese Academy of Sciences, Beijing, China, wq@iscas.ac.cn.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2018 Association for Computing Machinery.

1049-331X/2026/2-ART \$15.00

<https://doi.org/XXXXXX.XXXXXXX>

events and relations. An experimental evaluation is conducted on 5,070 Stack Overflow security posts, and the results show that SYNAT outperforms all baselines in both event and relation extraction, and achieves the highest tree similarity in attack tree synthesis. Furthermore, SYNAT has been applied to enhance HUAWEI's security knowledge base as well as public security knowledge bases CVE and CAPEC, which demonstrates SYNAT's practicality.

ACM Reference Format:

Ziyou Jiang, Lin Shi, Guowei Yang, Xuyan Ma, Fenglong Li, and Qing Wang. 2026. SYNAT: Enhancing Security Knowledge Bases via Automatic Synthesizing Attack Tree from Crowd Discussions. *ACM Trans. Softw. Eng. Methodol.* 1, 1 (February 2026), 28 pages. <https://doi.org/XXXXXX.XXXXXXX>

1 INTRODUCTION

Cyber attacks have become a serious threat to the security of software systems [70], and some attacks from malicious users may result in millions of dollars of losses in today's businesses [47]. Many research institutions and organizations have built their security knowledge base to help safeguard against attacks and vulnerabilities. For example, **Common Vulnerabilities and Exposure (CVE)** [7] is one of the widely-used public security knowledge databases that identifies each vulnerability with a unique name and standard description. Security practitioners can retrieve the relevant vulnerability from this database to patch the vulnerable projects and enhance their security. Another example is **Huawei Cloud Computing Technologies**, which is a Fortune Global 500 company in 2023, has set up a security knowledge base that consists of 241 attack trees and carries important information such as attack description, attack case, and possible mitigation in terms of STRIDE [73] threat model. The security knowledge bases provide developers and maintainers with code design specifications and threat modeling analysis of their products.

However, due to the time lag in the official release of security information, the knowledge base has not been maintained regularly, and the time lag may be utilized by attackers to deploy exploits, such as zero-day attacks. For CVE, Table 1 shows the disclosure time of the vulnerabilities and the created time of posts. We can see that some of the posts were created over 400 days earlier than the disclosure, and the time lags may be easily utilized by attackers. For HUAWEI's database, 77% of the attack trees have not been updated within the last four years. Some latest well-known attacks, such as *format-string vulnerability* have not been included in the knowledge base. Therefore, it is challenging for the company to protect its software systems against emergent security risks.

Table 1. The vulnerabilities disclosed in CVE that are first proposed in Stack Overflow's security posts.

| CVE-ID | Time of Disclosure | Corresponding Security Post | Time of Post Created |
|---------------|--------------------|---|------------------------|
| CVE-2011-1271 | 05-10-2011 | #2135509/bug-only-occurring-when-compile-optimization-enabled | 01-25-2010 (-470 days) |
| CVE-2012-5633 | 03-12-2013 | #7933293/why-does-apache-cxf-ws-security-implementation-ignore-get-requests | 10-28-2011 (-501 days) |
| CVE-2013-3350 | 07-10-2013 | #17351214/cf10-websocket-p2p-can-invoke-any-public-functions-in-any-cfc | 06-27-2013 (-13 days) |
| CVE-2015-3833 | 10-01-2015 | #24625936/getrunningtasks-doesnt-work-in-android | 07-08-2014 (-450 days) |
| CVE-2015-3198 | 07-21-2017 | #30028346/with-trailing-slash-in-url-jsp-show-source-code | 05-04-2015 (-809 days) |

Nowadays, developers tend to leverage online knowledge-sharing platforms, such as Stack Overflow, GitHub, etc., to discuss their security concerns with other developers about possible attacks, and ask for suggestions to develop secure software. These security discussions can be obtained from the security posts in Stack Overflow, and the GitHub Issues for different projects. Pan et al. [48] indicate that developers conduct the security discussions and they may pertain to the attacks that have not ever been officially reported. Identifying such emerging attacks on time will help prevent losses. In addition, due to the diversity and practical nature of Stack Overflow, the extracted information will enhance security knowledge bases for various IT organizations. Fig. 1 (a) shows an example, where developers discuss attacking and securing JWT in Stack Overflow. In particular, the questioner is concerned that hackers may steal the tokens and access their sessions. In the accepted answer, two possible attack methods are enumerated: access the computer or intercept the network traffic. If any of these attack methods succeeded, hackers could get the JWT and finally access the session data. According to this discussion, we can

potentially synthesize an attack tree, where the attack goal “***steal session***” could be achieved via two attack methods, as shown in Fig. 1 (b). The synthesized attack trees are invaluable for the software security community.

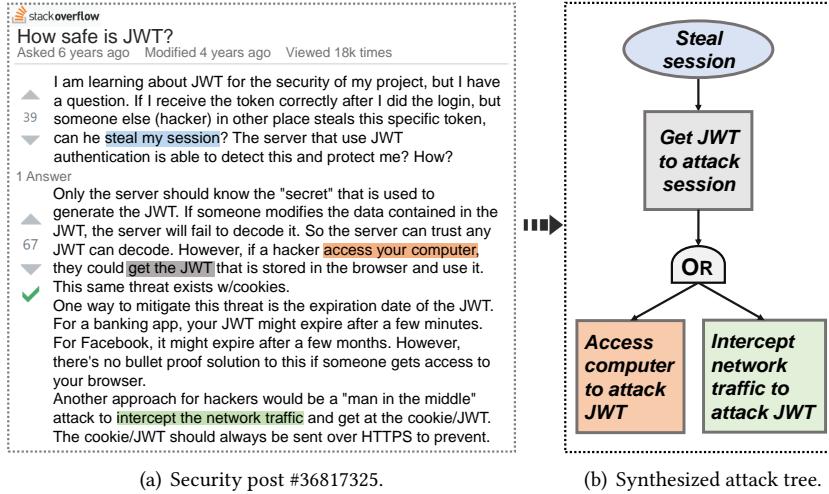


Fig. 1. The motivation example of attack tree synthesizing in security post #35817325.

However, directly synthesizing these attack trees that are buried in the textual security post is non-trivial, since the complex attack trees are highly abstract and reasonably inferred from multiple attack events that are described in the textual security discussions. To bridge the gap, we intuitively accommodate the general event and relation extraction to the security posts, then match the extracted elements to attack trees. In this paper, we propose SYNAT, an automatic approach to synthesize attack trees from security posts of Stack Overflow with transition-based event and relation extraction. Specifically, SYNAT first restricts the scope of sentences that may contain the attack information with Large Language Model (LLM) and with the auto-generated prompt, which has the State-of-the-Art (SOTA) performances on multiple natural language process tasks [51]. Then, it utilizes a transition-based joint event and relation extraction model to extract the attack events and their relations simultaneously from the scoped sentences. Finally, it applies a set of heuristic rules to synthesize the attack trees with the extracted events and relations.

To evaluate the performance of SYNAT in attack tree synthesizing, we conduct experiments on both Stack Overflow and GitHub, with 5,070 security posts, and 2,350 GitHub issue reports (IRs) labeled from the GHArchive [17], which is a large-scale IR-based security dataset collected from 2015. Then, we compare SYNAT with multiple representative baselines on attack tree synthesizing, event extraction, and relation extraction. The results show that, SYNAT achieves the highest tree similarity in attack tree synthesizing, with 10.24% Average Hamming Distance (AHD) and 7.93% Tree-edit Distance Similarity (TEDS). SYNAT also outperforms all baselines in both event and relation extraction, with 80.93% and 87.81% F1 scores, respectively. Furthermore, SYNAT has been practically applied to enhance the public security knowledge bases, i.e., CVE [7] and CAPEC [8], as well as HUAWEI’s security knowledge base. The major contributions of this paper are summarized as follows:

- **Technique:** SYNAT, an automated approach to synthesize attack trees based on LLM and joint event and relation extraction. To the best of our knowledge, this is the first work on automatically synthesizing attack trees from the crowd security posts of a knowledge-sharing platform.
- **Evaluation:** An experimental evaluation of the performance of SYNAT against state-of-the-art baselines, which shows that SYNAT outperforms all baselines.

- **Application:** Practical applications in enhancing HUAWEI’s security knowledge base and public security knowledge base, which demonstrate SYNAT’s practicality.
- **Data:** A public release of the dataset with 1,354 attack trees, and source code [61] to facilitate the replication of our study and its application in extensive contexts.

In the rest of the paper, Section 2 illustrates the background and motivation. Section 3 presents the details of our approach. Section 4 sets up the experiments. Section 5 describes the experimental results and analysis. Section 6 describes the application study. Section 7 presents the discussion and threats to validity. Section 8 discusses the related work, and Section 9 concludes this paper.

2 PRELIMINARIES

2.1 Attack Tree

Our study aims to synthesize attack trees from security posts. An attack tree is a tree-structured conceptual diagram, which demonstrates attack behaviors of invading a system in a formal way [43]. Since attack trees can enumerate the possible attacks to exploit a system [58], they have been widely used in security research and practice, such as threat modeling [70], to boost the target systems’ security [19, 21, 25, 60]. Typically, an attack tree is composed of one **Attack Goal**, the **Attack Methods** that can achieve the attack goal, and the **Relations** between attack methods (i.e. **AND** and **OR**), as shown in Fig. 2.

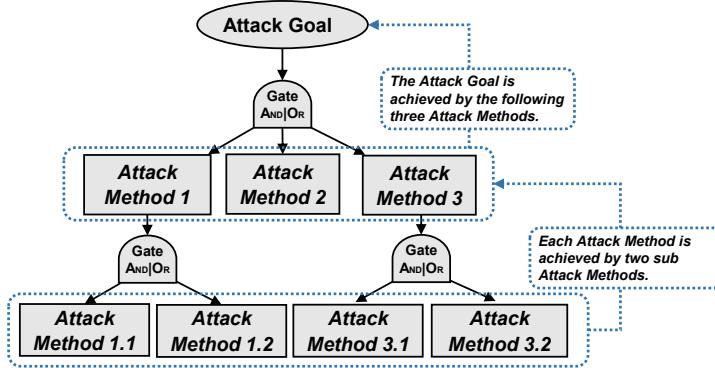


Fig. 2. The meta-framework of attack trees.

2.2 Motivation

Directly extracting attack trees from security posts (Security Post → Attack Tree) is extremely challenging since the complex attack trees are highly abstract and reasonably inferred from multiple attack events that are described in the textual security posts. Therefore, we reduce the abstract level by reformulating the attack tree extraction task into a two-step extraction, i.e., we first extract attack events and their relations from security posts and then synthesize the attack trees from these attack events (Security Post → Attack Events → Attack Tree).

2.2.1 Definition of Attack Event. Event Extraction is an important task in Natural Language Process [33]. An event refers to a specific occurrence of facts that happen in a certain time and a certain place, which can usually be described as a change of state [10]. Extracted Events contain one or more **triggers** (i.e., the main words or phrase that most clearly expresses an event occurrence) and **arguments** (i.e., roles involved in an event, describing the main components of the event). For example, the triggers for banking events could be the phrase ‘Transfer

Money', and the types of argument could be 'Beneficiary' and 'Recipient'. Event extraction requires identifying the event, classifying triggers, identifying arguments, and judging the argument type. In this paper, we define the attack event inspired by the previous work [43]. The **Attack Event** is composed of one **Trigger**, and two types of arguments (i.e., **Target** and **Instrument**).

- **Trigger:** Main verbs indicating security attacks.
- **Target:** Objects aimed by attackers to achieve attacks.
- **Instrument:** Tools utilized by attackers to achieve attacks.

2.2.2 Definition of Relation. The relation extraction is a task to predict the relation between a pair of events. Ning et al. [45] extract the temporal relations between events, such as BEFORE and AFTER. Glavas et al. [14] analyze the structural relations between events, which are mainly classified as PARENT-CHILD and CHILD-PARENT. In this study, we adapt the identification of attack tree edges to the relation extraction task. We define three types of relations that are useful for constructing attack trees as follows:

- **AND:** Two events have an AND relation if they both have to be satisfied to achieve another event, corresponding to the AND relation in the attack tree.
- **OR:** Two events have an OR relation if each of them can achieve an event independently, corresponding to the OR relation in the attack tree.
- **ACHIEVEDBY:** An event is ACHIEVEDBY of another event, if the former event can be achieved by satisfying the latter event itself or together with other events. This relation corresponds to the parent-child edge in the attack tree, which means the attack event can be the parent node of other events if it is achieved by these events.

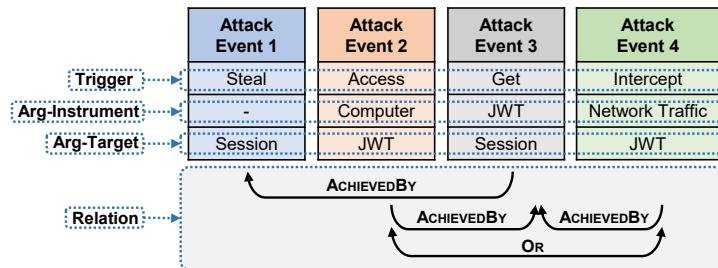


Fig. 3. The extracted attack events and relations of motivation example in Fig. 1.

Fig. 3 shows an example of attack events and relations from the security post illustrated in Fig. 1(a). Four attack events and their relations can be identified from the post. Based on the events and relations, we can synthesize the attack tree in Fig. 1(b).

2.3 Transition-based Extraction Framework

Recent research evidenced that the transition-based information extraction framework is efficient for extracting information that can be described as a change of states. For example, the transition-based extraction model has achieved state-of-the-art performances on the entity and relation extraction [66], dependency parsing [67], and semantic role labeling [6]. The basic idea of the transition-based framework is transforming the task of predicting a graph from a textual document, into predicting an action sequence of a state machine that produces the graph [4, 75]. In particular, a transition-based framework has two key components: (1) transition **states** and (2) a set of transition **actions**, where the state is used to record incomplete prediction results, and the action is used to control the transition between states.

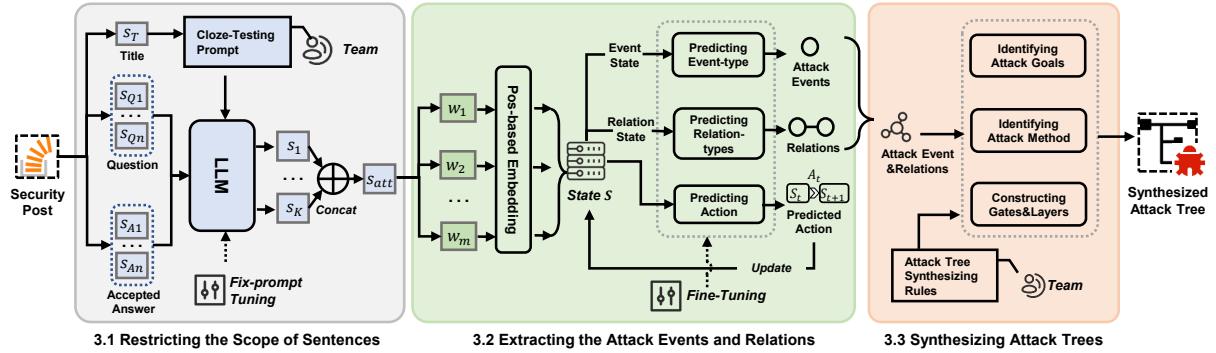


Fig. 4. The architecture of SYNAT.

The transition-based extraction framework has two major advantages compared with the other event and relation extraction models, as well as the LLMs that have been proposed recently: (1) Since the transition-based event and relation extraction only requires traversing the document once to obtain all the events and relations, it has a time complexity of $O(n)$, which is faster than all the other models. (2) The transition-based model has high accuracy when specifying the types of events and relations, and more details are shown in Section 5.

In this study, since the attack events can also be described as a change of states, we employ the transition-based extraction framework to facilitate the attack event and relation extraction by newly designing a transition system, and jointly performing event extraction and relation extraction as a single task.

3 APPROACH

The overview of SYNAT is illustrated in Fig. 4, which consists of four major steps. First, we restrict the scope of sentences from the security discussions with LLM, which may contain the attack events and relations. Second, we introduce a transition-based event and relation extraction model to extract attack events and relations simultaneously from the restricted scope. Third, we synthesize the attack tree with manually designed rules based on the extracted events and relations.

3.1 Restricting the Scope of Sentences

A security post \mathcal{P} consists of a *Title* S_T , *Question* [$S_{Q1}, S_{Q2}, \dots, S_{Qn}$] and *Accepted Answer* [$S_{A1}, S_{A2}, \dots, S_{An}$], where the *Question* and the *Accepted Answer* are all composed of a sequence of sentences. Among these sentences in *Question* and *Answer*, only a relatively small number of sentences are relevant to the attack events and their relations. To reduce the impact of non-relevant sentences, we restrict the sentences that are more likely to contain the attack events and their relations.

To achieve that, we utilize the LLM to restrict the scope of sentences and use Prompt Learning for tuning the LLM [57]. The paradigm LLM+Prompt has achieved state-of-the-art performances on various natural language process (NLP) tasks [35]. Compared with the previous Pre-train+Fine-tuning paradigm, prompt learning can fully utilizes the resources of models with its Prompt Template [16], and bridge gaps between inputs and LLMs.

3.1.1 LLM-based Scope Restriction. To restrict the scope of sentences with the prompt learning, we design a template based on the **Cloze-testing**, which is a widely-used prompt template in text extraction task [51]. We compare three widely-used templates, i.e., **Cloze Template**, **Prefix Template**, and **Template w/o Title**, and choose the best-performed one. More details will be introduced in Section 7.2). Given the security post \mathcal{P} , the template for restricting the scope is shown as follows:

Cloze-Template Prompt for LLM-based Scope Restriction

Cloze-Testing: From the following security post {Question, Accepted_Answer} (Title is {Title} to summarize the main topic of security post), [Y] are restricted sentences that may contain the attack trees. (Omit the definition of attack trees.)

Task: Please predict the token [Y] with K restricted sentences from the security post's question and answers to make the cloze-testing complete. The format of the output is the bullet list with Sentence 1 to Sentence K.

where the {Question, Accepted_Answer} indicates the inputted sentences in *Question* and *Accepted Answer*. Since the {Title} of the security post summarizes the main topic of the security post, we incorporate it in the template to restrict the scope more accurately. The token [Y] is the output of restricted sentences that may contain the attack trees, and we ask the LLM to predict the sentences in this token to make the close testing complete. We choose the generative GPT-3 [5] as the LLM, which outperforms other LLMs on the text generation tasks. We utilize the GPT-3 to output the restricted K sentences with the [Y] token as $[S_1, \dots, S_K]$, then concatenate them into one single sentence as the sentence-scope S_{att} .

3.1.2 Fix-prompt Tuning. To train the LLM, we apply the Fix-prompt Tuning [36], which is a typical training method for the manually designed prompt templates:

$$\mathcal{L}_{sco} = Cr(y_{sco}, S_{att}) \quad (1)$$

where function Cr indicates the CRINGE loss [3], which is specifically used for training generative LLMs; y_{sco} is the ground-truth label for the sentences that contain the attack tree. We train the LLM with the loss \mathcal{L}_{sco} until it achieves the training convergence.

3.1.3 Comparison of LLMs. To select the best LLM for restricting the scopes of sentences, we compare the performances with four representative LLMs on restricting the sentence scopes, i.e., **Albert** [26], **T5** [54], and **GPT-3** [5], and **ChatGPT** [46]. Compared with other LLMs, these models achieve the highest performances on different NLP tasks. We utilize the same fix-prompt tuning method to train the Albert, T5, and GPT-3. Since ChatGPT has not opened the fine-tuning interface, we choose in-context learning (ICL) to optimize it. According to the previous works [42, 59], the ICL is the effective few-shot strategy that can enable the ChatGPT on various NLP tasks, and we find that 5-shot ICL can achieve the highest performance on sentence restricting. To measure the performances, we utilize three metrics to analyze the performances of restricting the sentences, i.e., **Precision (Pre.)**, **Recall (Rec.)**, and **F1-measure (F1)**:

$$Pre = \frac{N(Restrict \cap Attack)}{N(Restrict)}, Rec = \frac{N(Restrict \cap Attack)}{N(Attack)}, F1 = \frac{2 \times Pre \times Rec}{Pre + Rec} \quad (2)$$

where $N(\cdot)$ calculates the number of sentences; *Restrict* indicates the sentences restricted by LLMs, and *Attack* indicates the ground-truth sentences that contain the attack methods and relations.

Table 2. The performances of various LLMs on restricting sentences that cover the attack trees. (%).

| Method | Model Version | Parameters | Training Strategy | Metrics | | |
|---------|------------------|------------|-------------------|--------------|--------------|--------------|
| | | | | Pre. | Rec. | F1 |
| Albert | Albert-large | 18M | Fix-prompt Tuning | 80.93 | 84.65 | 82.75 |
| T5 | T5-base | 220M | Fix-prompt Tuning | 90.10 | 91.74 | 90.91 |
| GPT-3 | Text-davinci-003 | 175B | Fix-prompt Tuning | 97.24 | 99.36 | 98.29 |
| ChatGPT | GPT-3.5-turbo | Unknown | ICL (5-shot) | 76.35 | 88.54 | 82.00 |

Table 2 shows the results of different LLMs. We can see that, the fine-tuned GPT-3 outperforms other models in restricting the scope of sentences with the 97.24% Precision and 99.36% Recall. Therefore, we believe that using GPT-3 with the fix-prompt tuning can accurately restrict the scope of sentences that may contain the attacks.

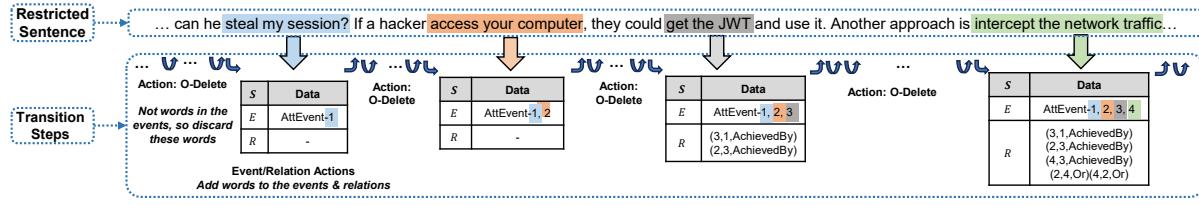


Fig. 5. The example of how the transition-based model performs to extract the events and relations from the restricted scope of sentences in motivation example (Fig. 1).

3.2 Extracting the Attack Events and Relations

In this section, we propose a transition-based model to jointly extract attack events and their relations. We first define the state, which is a key concept in the transition-based model. We also define 12 actions that transit the states and extract the events and relations simultaneously while updating the intermediate results. Finally, we describe the four key steps involved in the transition-based model to extract events and relations.

3.2.1 The Logic Flow of the Transition-based Event & Relation Extraction. In Section 2.3, we have discussed the benefits of the transition-based framework that is utilized in the information extraction tasks, such as joint entity-relation extraction. However, the structures of the attack events and their relations are more complex than the normal entities, and previous transition-based frameworks cannot be directly applied to our tasks on synthesizing the attack trees. To extract the attack events and relations, we propose a new transition-based model based on their structures.

The transition-based model needs to traverse all the words in the restricted sentence scope **once** and preserve all the essential information while traversing the sentence. When the model traverses to a certain word w_i , we utilize the **State S** to save the current situation based on the historical words $(w_1, \dots, w_i) \rightarrow S_i$, which illustrates which attack events and relations are extracted, and some intermediate information that can be used to predict the next events and relations. After storing the current state, the model will traverse to the next word w_{i+1} , and determine whether the state will be **updated** based on the newly-traversed word with the **Action A** . The action is determined by the current state and the newly-traversed words $(S_i, w_{i+1}) \rightarrow A_i$. As illustrated in Fig. 5, we utilize a specific example to illustrate how the transition-based framework performs to extract the events and relations in the restricted scope of sentences, where restricted sentences come from the motivation example in Fig. 1. If the transition-based model encounters words that do not belong to the events and relations, it will delete them with the action O-DELETE. Otherwise, the model will preserve these words and restore them to the current states, where we separately store the events and relations to the event set (E) and relation set (R). From this example, we can intuitively find out that the transition-based model only requires traversing all words once to extract the attack events and relationships, which has a small time and space complexity ($O(n)$).

In practice, it is quite a challenge to utilize the transition-based model to extract the attack events and relations, mainly due to the following reasons. (1) First, the current definition of the state S only incorporates the event and relation sets, but we also need to design a specific data structure to distinguish the **Trigger**, **Arg-Instrument**, and **Arg-Target** in the attack events, as well as storing the relations between attack events. (2) Second, the processed words in each state need to be considered to predict the next action, where some words deleted or inserted into the previous two sets should be involved in the action prediction. To address this, we include the **stack** in the state to hold the processed and unprocessed words. (3) Third, the transition-based model requires the appropriate function to map current states to the required actions, and we introduce the StackLSTM to embed the current state, and predict the action with a linear multi-label classifier.

Table 3. The formula expression and description of 12 state actions.

| Category | Id | Action | Formula Expression* | Description |
|------------------|--------------------|-------------|---|---|
| Event Actions | \mathcal{A}_1 | GEN-TRG | $\{[j e]\} \Rightarrow \{e, [j^t \beta], E \cup \{j^t\}\}$ | The word j is predicted as trigger in e , so it is marked as j^t and moved to β , and j^t is inserted into set E . |
| | \mathcal{A}_2 | GEN-ARG | $\{[j e]\} \Rightarrow \{e, [j^a \beta], E \cup \{j^a\}\}$ | The word j is predicted as argument, so it is marked as j^a and moved to β , and j^a is inserted into set E . |
| | \mathcal{A}_3 | DEP-SHIFT | $\{[\xi i^t], [j^a \beta]\} \Rightarrow \{[\xi i^t], [\tau j^a], \beta, E \cup \{i^t \leftrightarrow j^a\}\}$ | Construct the one-to-one dependency with the top argument j^a in β and the top trigger i^t in ξ , insert it into set E , and push j^a back to the τ . |
| | \mathcal{A}_4 | TRG-PASS | $\{[\xi i^t], [j^a \beta]\} \Rightarrow \{\xi, [i^t \xi'], E \cup \{i^t \leftrightarrow j^a\}\}$ | Construct the one-to-many dependency with the top argument j^a in β and the top trigger i^t in ξ' , and insert it into event state E . |
| | \mathcal{A}_5 | ARG-PASS | $\{[\tau i^a], [j^t \beta]\} \Rightarrow \{\tau, [i^a \tau'], E \cup \{i^a \leftrightarrow j^t\}\}$ | Construct the one-to-many dependency D_{ij} with the top trigger j^t in β and the top argument i^a in τ' , and insert it into event state E . |
| Relation Actions | \mathcal{A}_6 | RIGHT-SHIFT | $\{[\xi i^t], [j^t \beta]\} \Rightarrow \{[\xi i^t, j^t], \beta, R \cup \{j^t \rightarrow i^t\}\}$ | Construct the one-to-one right-to-left relation from the top trigger i^t in ξ to the top trigger j^t in β , insert it to the set R , and push the j to the ξ . |
| | \mathcal{A}_7 | RIGHT-PASS | $\{[\xi i^t], [j^t \beta]\} \Rightarrow \{\xi, [i^t \xi'], R \cup \{j^t \rightarrow i^t\}\}$ | Construct the one-to-many right-to-left relation. Shift the i^t from ξ to ξ' , construct a relation from i^t to the top trigger j^t in β , insert it to the set R . |
| | \mathcal{A}_8 | LEFT-SHIFT | $\{[\xi i^t], [j^t \beta]\} \Rightarrow \{[\xi i^t, j^t], \beta, R \cup \{j^t \leftarrow i^t\}\}$ | Construct the one-to-one left-to-right relation from the top trigger j^t in β to the top trigger i^t in ξ , insert it to the set R , and push the j^t to the ξ . |
| | \mathcal{A}_9 | LEFT-PASS | $\{[\xi i^t], [j^t \beta]\} \Rightarrow \{\xi, [i^t \xi'], R \cup \{j^t \leftarrow i^t\}\}$ | Construct the one-to-many left-to-right relation. Shift the i^t from ξ to ξ' , construct a relation from the top trigger j^t in β to i^t , insert it to the set R . |
| Word Actions | \mathcal{A}_{10} | GEN-SHIFT | $\{[j \beta]\} \Rightarrow \{[j e]\}$ | Process a word j . Shift a word j from stack β to e for further processing. |
| | \mathcal{A}_{11} | NO-SHIFT | $\{[i^t \xi'], [j^a \tau'], [k^t, l^a e]\} \Rightarrow \{[\xi i^t, k^t], [\tau j^a, l^a]\}$ | Store the processed word. Push the triggers and arguments from ξ' , τ' , e to the ξ and τ . |
| | \mathcal{A}_{12} | O-DELETE | $\{[j \beta]\} \Rightarrow \{\beta\}$ | Delete a word j from state β . |

* Note: The symbol $[X^Y|Z]$ indicates that the word X has the type Y (trigger t or argument a), and it is pushed into the top of Stack Z .

3.2.2 The Definition of State. To accommodate the transition-based joint event and relation extraction, we first define a transition state $\mathcal{S} = \{\xi, \tau, \xi', \tau', e, \beta, E, R\}$ to represent the key information regarding attack events and relations, where the eight elements of \mathcal{S} are described as follows:

- ξ and τ are stacks to store the words of processed triggers and arguments, respectively.
- ξ' and τ' are stacks to store the words that were popped out of ξ and τ but will be pushed back later.
- e is the stack holding the words of partial trigger and action.
- β is a buffer holding the unprocessed words in the candidate sentence.
- E and R are two sets, which respectively store the predicted events (triggers, arguments, and their dependencies), and relations between events predicted during the extraction process.

Based on the event set E , we can formulate an event E_i with a trigger i^t , and arguments j^a , stored in the set that depend on the trigger. That is, $E_i = (i^t, j^a, arg_type)$, where arg_type is the type of argument, i.e., Target and Instrument. The relation between two events E_i and E_j is formulated as $R_{ij} = (E_i, E_j, rel_type)$, where rel_type is the type of relation, i.e., AND, OR and ACHIEVEDBY.

3.2.3 The Definition of Action. We define 12 actions $\{\mathcal{A}_1, \mathcal{A}_2, \dots, \mathcal{A}_{12}\}$ to transit the states, and reason attack events and their relations during the extraction process. As shown in Table 3, these actions are in three categories:

- **Event Actions** ($\mathcal{A}_1 \sim \mathcal{A}_5$), which are utilized to recognize triggers (GEN-TRG, TRG-PASS) and arguments (GEN-ARG, ARG-PASS) of attack events, and the dependencies between them (DEP-SHIFT);

Algorithm 1: Process of transition-based attack events and relations extraction.

Input: The words in restricted sentence S_{atk} : $\{w_1, w_2, \dots, w_m\}$; transition state \mathcal{S} ; and transition actions $\{\mathcal{A}_1, \mathcal{A}_2, \dots, \mathcal{A}_{12}\}$.

Output: The terminal state \mathcal{S}_T with the extracted events and relations in $\{E, R\} \in \mathcal{S}_T$.

// **Step 1:** Initializing the state to \mathcal{S}_0 , with the position-based word embeddings.

1 $\{\vec{w}_1, \vec{w}_2, \dots, \vec{w}_m\} = PosEmbedding\{w_1, w_2, \dots, w_m\}, [\{\vec{w}_1, \vec{w}_2, \dots, \vec{w}_m\}]|\beta]$, where $\mathcal{S} = \mathcal{S}_0$;

// **Step 2:** Predicting the actions and updating the states, which are the iterations of state-action transition.

2 **while** State Stacks $\{\xi', \tau', e, \beta\} \in \mathcal{S}$ are empty, where $\mathcal{S} = \mathcal{S}_T$ **do**

3 $\tilde{\mathcal{S}} = StackLSTM(\mathcal{S})$;

4 $\mathcal{A}_i = \arg \max_{\mathcal{A}_i} p(\mathcal{A}_i | \mathcal{S}) = \arg \max_{\mathcal{A}_i} \sigma_{\mathcal{A}}(\tilde{\mathcal{S}})$, where $\sigma_{\mathcal{A}}$ is the multi-label classifier;

5 $\mathcal{S} \xrightarrow{\mathcal{A}_i} \mathcal{S}_{next}$, where the details of transition steps are shown in Table 3;

6 **end**

7 **for** $\{i^t \leftrightarrow j^a\} \in E$ and $\{i^t \leftrightarrow j^t\} \in R$ **do**

8 // **Step 3:** Predicting the type of the events and relations, with the sets in $\{E, R\} \in \mathcal{S}_T$.

9 $E_i.arg_type = \arg \max_{arg_type} p(arg_type | i^t, j^a) = \arg \max_{arg_type} \sigma_E(\vec{j}^a, \vec{i}^t)$, where σ_E is the multi-label classifier, and $arg_type \in \{\text{Target, Instrument}\}$;

10 $\vec{E}_i = \vec{i}^t \oplus \{\vec{j}_1, \vec{j}_2, \dots, \vec{j}_n\}$, where all the arguments j depend on i^t in E_i : $i^t \leftrightarrow \{j_1, j_2, \dots, j_n\}$;

11 $R_{ij}.rel_type = \arg \max_{rel_type} p(rel_type | E_i, E_j) = \arg \max_{rel_type} \sigma_R(\vec{E}_i, \vec{E}_j)$, where σ_R is the multi-label classifier, and $rel_type \in \{\text{AND, OR, ACHIEVEDBy}\}$;

12 // **Step 4:** Identifying the implicit relations, with **Transitive Rule** and **Asymmetric Rule**.

13 **Transitive:** $(R_{ij}.rel_type = \text{ACHIEVEDBy}, R_{jk}.rel_type \neq \emptyset) \Rightarrow (R_{ik}.rel_type = \text{ACHIEVEDBy})$;

14 **Asymmetric:** $(R_{ij}.rel_type = \text{ACHIEVEDBy}) \Rightarrow (R_{ji}.rel_type \neq \text{ACHIEVEDBy})$;

15 **end**

16 **return** $\{E, R\}$;

- **Relation Actions** ($\mathcal{A}_6 \sim \mathcal{A}_9$), which are used to recognize relations between event pairs (RIGHT/LEFT-SHIFT, RIGHT/LEFT-PASS);
- **Word Actions** ($\mathcal{A}_{10} \sim \mathcal{A}_{12}$), which are used to process (GEN-SHIFT) and delete (O-DELETE) an unprocessed word, or store the processed words (NO-SHIFT).

3.2.4 *Event and Relation Extraction.* Our transition-based event and relation extraction is an iterative process. In each iteration, the model predicts an action based on the current state and then uses the predicted action to update the state, which will be used to predict the action in the next iteration. Such process continues until the model terminates (i.e., stacks ξ', τ', e, β become empty) or a bound on the number of iterations is reached. There are four key steps involved in the process, as is shown in Algorithm 1:

Step 1: Initializing states (Line 1 in Algorithm 1). Given the restricted sentence s_{atk} (Section 7.2), we embed each inside word in a combined way to better capture the contextual information, and initialize the state \mathcal{S} with the word embedding. We first embed each word with the position-based word embedding (*PosEmbedding*) [62], which is a widely-used method that can embed the words with its position information, and improve the accuracy of joint extraction [76]. Then, we initialize the unprocessed words stack $\beta \in \mathcal{S}_0$ with the embedded words, and keep the other states empty.

Step 2: Predicting Actions and Updating States (Line 2~6). After initializing the state \mathcal{S}_0 , we input it into the transition iterations. In each iteration, SYNAT predicts an action based on the current state. We first represent

the state to \vec{S} with the Stack LSTM [11], which can embed the state stacks with the push&pop order of words. Then, we introduce a multi-label classifier with the fully connected layer $\sigma_{\mathcal{A}}$, which can predict the probability of each of the 12 actions we have defined. The predicted action \mathcal{A}_i with the highest probability will update the transition state S , which will further be used to predict the action in the next iteration.

Step 3: Predicting Events and Relations (Line 8~10). After the iteration reaches the terminal, we first predict the argument type with the event set $E \in \mathcal{S}_T$. To decide whether the argument type in event E_i 's dependency between trigger i^t and argument j^a (i.e., $i^t \leftrightarrow j^a$) is **Instrument** or **Target**, we feed the word embeddings of them into another multi-label classifier σ_E and choose the argument type with the highest probability $E_i.\text{arg_type}$. After all the argument types of E_i have been decided, we concatenate the embeddings of trigger word \vec{i}^t and all the dependent arguments $\{\vec{j}_1, \vec{j}_2, \dots, \vec{j}_n\}$, and build the event embedding \vec{E}_i .

Second, we predict the relation types with the relation set $R \in \mathcal{S}_T$. To decide whether the relation type between event E_i and E_j (i.e., R_{ij}) is **AND**, **OR** or **ACHIEVEDBY**, we introduce the third multi-label classifier σ_R , and feed the embeddings of events \vec{E}_i and \vec{E}_j into the classifier. The relation type that has the highest probability is chosen as the type for relation $R_{ij}.\text{rel_type}$.

Step 4: Identify the Implicit Relations (Line 11~12). After predicting the types of all the events and relations, according to the attack tree structure, defined by Vidhyashree et al. [43], we propose two rules to identify relations that are not explicitly predicted, i.e., **Transitive Rule** and **Asymmetric Rule**.

- **Transitive Rule:** The ACHIEVEDBy relation has transitivity, i.e., if the attack event E_i can be achieved by E_j , and E_j has any relation with E_k , then E_i can be achieved by the event E_k .
- **Asymmetric Rule:** The attack tree should not contain the circles, i.e., if attack event E_i can be achieved by E_j , then E_j cannot be achieved by E_i .

After these four steps, we can obtain the final events and relations in the two sets E and R .

3.2.5 Fine-tuning. We use gradient descent to fine-tune the three multi-label classifiers in the transition-based event and relation extraction model. After extracting the events and relations, we utilize the combined loss to jointly optimize the classifiers as follows:

$$\mathcal{L} = \lambda \cdot \underbrace{\text{SSVM}(y_E, y_R, p(\text{arg_type}|E), p(\text{rel_type}|R))}_{\mathcal{L}_{\text{Extract}}} + (1 - \lambda) \cdot \underbrace{H(y_{\mathcal{A}}, p(\mathcal{A}|S))}_{\mathcal{L}_{\mathcal{A}}} \quad (3)$$

where \mathcal{L} is the combined loss, and λ is the loss-balancing parameter. where y_E , y_R , and $y_{\mathcal{A}}$ are ground-truth argument-type in set E , relation-type in set R , and the iterations' actions.

The combined loss \mathcal{L} consists of two parts, and the first part is the loss for event and relation extraction $\mathcal{L}_{\text{Extract}}$. Given the probabilities of the event's argument-type prediction (**Line 8 in Algorithm 1**) and the relation-type prediction (**Line 10**), we calculate the loss based on the SSVM loss, which is the widely-used loss for joint event and relation extraction [15].

The second part is the loss for action prediction $\mathcal{L}_{\mathcal{A}}$, where the function $H(y, p_x)$ calculates the cross-entropy loss between the probabilities of action prediction (**Line 4**), and ground-truth action labels. We iteratively optimize the SYNAT until the combined loss achieves convergence.

3.3 Synthesizing Attack Trees

In this step, we aim to synthesize the attack trees with three rules based on the extracted attack events and their relations. Given the extracted events and relations, we compare the differences between attack trees and these extracted results, and more details are shown in Section 2.2.

Then, we ask the three security practitioners to help us design three synthesizing rules to synthesize the attack trees according to their structures [43]. The same three security practitioners have over five years of

experience in software security, and they will also participate in the ground-truth labeling of our dataset in the following section. We introduce the **Open Card Sorting** [56] to devise the rules that synthesize the attack trees, which is a flexible classification method that allows us to create information categories freely, thus helping designers develop more appropriate types. Specifically, the rule designers follow three criteria when analyzing the differences between the extracted event & relations and the ground-truth attack trees in the training dataset.

- **Criterion 1 (Node Creation):** We ask the rule designers to independently define the rules that reflect the content of nodes in the synthesized attack trees based on the extracted events.
- **Criterion 2 (Edge Creation):** We ask the rule designers to independently define the rules that reflect the layer between the attack tree's nodes with the extracted relation, and find the attack goals in the tree.
- **Criterion 3 (Node/Edge Deletion):** We ask the rule designers to discuss with each other and remove some rules that cannot properly describe the nodes and edges in the attack trees, then we uniform and simplify the expressions of these rules.

We conduct over 10 iterations of open card sorting with the rule designers and analyze over 500 pairs of extracted events & relations and attack trees, and we also ask HUAWEI's security practitioners to utilize the attack trees in their company's security knowledge base when designing the rules. Finally, we devise the following three rules to synthesize the attack trees, which can map the events to the attack goals and methods, as well as map the relations to the tree layers and logic gates.

- **Rule 1 (Identifying Attack Goal):** The attack goal is the final target of all the attack methods. Therefore, the attack event that the attacker can achieve indicates the attack tree's goal. The content pattern for the attack goal node is that "*attackers achieve (i.e., the event's [trigger] that represents how to achieve the target) a certain target (i.e., the event's [target] argument)*", and we use the attack event to replace the content. For example, the root attack event is "Attack Event 1" in Fig. 3, which has a trigger "**Steal**" and a target argument "**Session**". Thus, the content for the attack goal in the synthesized attack tree is "**Steal session**", as shown in Fig. 1(b).
- **Rule 2 (Identifying Attack Methods):** For the other events that are not identified as attack goals, we identify them as multiple attack methods that can be used to achieve the attack goal. The content pattern for each attack method node is that "*attackers use (i.e., the event's [trigger] that represents how to use the instrument) the instrument (the event's [instrument] argument) to attack a certain target (the event's [target] argument)*", and we use the corresponding attack events to replace them respectively.
- **Rule 3 (Constructing Gates & Layers):** The ACHIEVEDBy relation indicates the layers of attack trees, and the AND and OR relations indicate the type of logic gates between all the attack methods in the same layers. Therefore, we construct the attack trees with the relations. For example, the logic gate between "Attack Event 2" and "Attack Event 4" in Fig. 3 is **OR Gate**. The attack methods identified from these two methods are the sub-node of the "Attack Event 3" since they both have the ACHIEVEDBy relation with it.

Finally, SYNAT synthesizes attack trees from the security discussions with the previous three rules, and we store them as crowd security knowledge for future reuse.

4 EXPERIMENTAL DESIGN

To evaluate the performance of SYNAT, we investigate the following three research questions:

- **RQ1: What is the performance of SYNAT on synthesizing attack trees from the security discussions?**
- **RQ2: What is the performance of SYNAT on extracting events from the security discussions**
- **RQ3: What is the performance of SYNAT on extracting relations from the security discussions?**
- **RQ4: What is the performance of SYNAT on enhancing the security knowledge bases?**

Table 4. The statistics of our dataset.

| Category | Original | | | | Augmented | | | |
|----------|----------|-------|--------|-----------|-----------|--------|--------|-----------|
| | #post | #tree | #event | #relation | #post | #tree | #event | #relation |
| Train | 3,042 | 868 | 1,685 | 4,625 | 16,175 | 13,425 | 30,172 | 89,629 |
| Valid | 1,014 | 189 | 497 | 1,255 | 1,014 | 189 | 497 | 1,255 |
| Test | 1,014 | 297 | 828 | 1,876 | 1,014 | 297 | 828 | 1,876 |
| Total | 5,070 | 1,354 | 3,010 | 7,756 | 18,203 | 13,911 | 31,497 | 92,760 |

4.1 Security Dataset Preparation

To evaluate the performance of SYNAT, we prepare the security dataset from Stack Overflow’s security posts with the following four steps, i.e., security post collection with security-related tags, data preprocessing, ground-truth labeling, and data augmentation.

Step 1: Security Post Collection. Following the previous study [28], we collect security posts from Stack Overflow via Stack Exchange [12]. Specifically, we retrieve security posts from the beginning until Jan 1st, 2023, and prepare the dataset in the following steps: (1) We retrieve all the posts in Stack Overflow that have security-related tags (i.e., `<security>`, `<websecurity>`, and `<firebase>`, etc.). Among the posts in Stack Overflow, we find that posts with specific tags may contain over 90% of security-related knowledge, such as the attack and mitigations for the different threats; (2) We remove the posts that receive negative scores voted by Stack Overflow users, as well as non-English posts; and (3) We select the posts that have an accepted answer, and this answer receives the highest score in the post to reduce the bias in subjective decisions.

Step 2: Data Preprocessing. We preprocess the collected security posts as follows: (1) First, we remove stopwords, correct any typos, and lemmatize the posts with Spacy [13]. (2) Then, we replace the code blocks (wrapped by HTML tags `<code>`, `</code>`), with token `[CODE]`. (3) Finally, we remove other HTML tags, such as `<a>`, ``, etc., and retrain the plain text inside these HTML tags.

Step 3: Ground-truth Labeling. For each security post, we label the triggers and arguments based on BIO tagging [76], and label the events and relations with Han’s tagging [15]. To guarantee the correctness of the labeling result, we build a 13-people team with three professors, two Ph.D. students, five Master’s students, and three security practitioners of HUAWEI. All of them have rich experience in secure software development, and six out of 13 (46%) participants are external annotators to reduce the bias of labeling. Each security post is labeled by two different team members. When different opinions occur on the labels, we discuss them with all team members until a decision has been reached. Only a few attack trees have conflicting opinions, and the average Cohen’s Kappa [50] is 0.87¹, meaning the biases of ground-truth labeling are minor.

Step 4: Data Augmentation. For training and testing the SYNAT, we split the dataset into training, validating, and testing datasets with the proportion 60%, 20%, and 20%. Since the number of trained posts with attack trees is much lower than posts without attack trees, as shown in **Original** column of Table 4, we employ two strategies to augment the training dataset. The first strategy is to use bootstrap sampling [55], which randomly copies the posts with events and relations. The second strategy is EDA [69] augmentation, which is a typical data augmentation technique. While keeping the events and relations not affected by data augmentation, EDA randomly inserts, deletes, and swaps several words in the source security post with a certain probability. The results of our data augmentation are shown in the **Augmented** column.

Table 4 shows the size of the dataset, where the column “**Original**” shows the number of posts, trees, events, and relations in the original dataset, and the column “**Augmented**” shows the size of the expanded dataset. In total, we have collected 5,070 security posts initially and expanded the dataset to 18,203 security posts.

¹The magnitude guidelines define that kappa value above 0.81, which means almost perfect agreement among all the team members.

4.2 Baselines

Since our work is the first to synthesize attack trees from security posts, there are currently no directly comparable approaches. Therefore, we use the SOTA approaches of the most similar NLP tasks, i.e., *attack tree synthesizing* task (RQ1), *attack event extraction* task (RQ2), and *relation extraction* task (RQ3). To make these approaches fit our task, we fine-tune them with the same settings as our approach, so that they can output the attack trees, attack events, and attack-event relations. Note that, to fairly compare with baselines, we retrain and test them under the same dataset and parameter settings as SYNAT.

Baseline for RQ1. In RQ1, we compare the SYNAT with the two types of baselines. The first type is the joint event and relation extraction model, i.e., the **Structured-Joint** [15], which is the SOTA model to extract both event and relation in parallel, with shared representation learning and structured prediction. To accommodate Structured-Joint to the attack tree synthesizing, we extend it to **Structured-Joint+** by (1) retraining with the same labeled event and relation data as our SYNAT; (2) applying our heuristic rules to help Structured-Joint synthesize attack trees. The second type is the aforementioned LLM, which is effective in various NLP tasks. We choose three SOTA generative LLMs to synthesize the attack trees, i.e., **Albert** [26], **T5** [54], and **GPT-3** [5], by utilizing the follow-up prompt template:

Cloze-Template Prompt for LLM Baselines

Cloze-Testing: From the following security post {Question, Accepted_Answer}, {Title is {Title}} to summarize the main topic of security post [Y] is the synthesized attack tree. (Omit the definition of attack trees)

Task: Please predict the token [Y] with the generated attack trees from the security post's question and answers to make the cloze-testing complete. The format of the attack tree is a dictionary. We use the following example to show the format.

Output Example: {Nodes: [{G: steal session}, {M₁: get JWT to attack session}, {M₂: access computer to attack JWT}, {M₃, intercept network traffic to attack JWT}], Edges: [(G, M₁, ACHIEVEDBY), (M₁, M₂, ACHIEVEDBY), (M₂, M₃, OR)]}

where the *Question*, *Accepted_Answer*, and *{Title}* are the input sentences and title of the security post, and *[Y]* is the text for representing the output attack trees, which contains a set of tuples [58]. The output of the attack tree is organized as a dictionary *{Nodes: Node_List, Edges: Edge_List}*. In the *Node_List*, *G* is the attack goal, and *M_i* indicates the *i_{th}* attack method. In the *Edge_List*, we use the triplets to represent the edges, where the first two elements indicate the nodes concatenated by the edge, and the third element is the relation. We also use the output example of Figure 1 to guide the LLMs to output the attack trees.

Baselines for RQ2. We first compare the SYNAT with the previous joint event and relation extraction model, i.e., **Structured-Joint**, and the best-performed LLM in RQ1, i.e., **GPT-3**. Then we compare SYNAT with two representative baselines on event extraction, which obtain the SOTA performances on our dataset. **ED3C** [63] introduces the context selection to reduce the noise and improve the representation of events in a large-scale document; and **MLBiNet** [37] propagates event-based semantic information across sentences and extract multiple events from the complex documents.

Baselines for RQ3. We first compare the SYNAT with the joint event and relation extraction model, and best-performed LLM in RQ1, i.e., **Structured-Joint** and **GPT-3**. Then we compare SYNAT with two representative baselines on event extraction, which obtain SOTA performances on our dataset. **KnowledgeILP** [44] is a data-driven method which incorporates ILP and commonsense knowledge; and **JC Learning** [65] is a joint-constrained learning model for multi-faceted event-event relation extraction. In addition, we also compare the **Time Cost** (i.e., Training/Fine-tuning Hours) of SYNAT and baselines. We calculate the hours of training the SYNAT and fine-tuning the baselines on our dataset.

4.3 Evaluation Metrics

To evaluate the performance of attack tree synthesizing, we employ two widely used metrics of tree similarity [34]: Average Hamming Distance and Tree-editing Distance Similarity, reflecting to what extent the synthesized attack trees are similar to the ground truth. (1) **Average Hamming Distance (AHD)** [23] aims to measure the difference between attack goals and methods, which calculates the average proportion of deviation words between nodes at the same position on synthesized and ground-truth attack tree, indicating the node similarity between two trees. (2) **Tree-editing Distance Similarity (TEDS)** [31] aims to measure the relation extraction based on the attack tree, which calculates the average proportion of edges and logic gates to be edited from synthesized to the ground-truth attack tree, indicating the structural similarity between two trees.

$$AHD = \frac{\sum \text{Hamming}(\text{Node_Predict}, \text{Node_Truth})}{\max N(\text{Node_Predict}, \text{Node_Truth})}, TEDS = \frac{N(\text{Edge_Predict} \rightarrow \text{Edge_Truth})}{\max N(\text{Edge_Predict}, \text{Edge_Truth})} \quad (4)$$

where function $N(\cdot)$ calculates the number of nodes, edges, and editing steps from predicted to ground-truth trees. A lower AHD or TEDS value means a higher level of tree similarity.

To evaluate the performance of event and relation extraction, we use three commonly used metrics: (1) **Precision (Pre.)**, which calculates the ratio of correct positive predictions to the total positive predictions; (2) **Recall (Rec.)**, which calculates the ratio of correct positive predictions to the ground-truth positive labels; and (3) **F1-measure (F1)**, which calculates the harmonic mean of the precision and recall.

$$Pre = \frac{N(ER_Predict \cap ER_Truth)}{N(ER_Predict)}, Rec = \frac{N(ER_Predict \cap ER_Truth)}{N(ER_Truth)}, F1 = \frac{2 \times Pre \times Rec}{Pre + Rec} \quad (5)$$

where function $N(\cdot)$ calculates the number of events and relations. The higher of these three metrics indicates the higher performances of event and relation extraction.

4.4 Experiment Setting

For SYNAT training on all experiments, we utilize the grid search [27] technique to tune the hyper-parameters until the model achieves the highest performances. Among these hyper-parameters, we set the *batch_size* = 8 for training the SYNAT, and λ = 0.5 for loss balancing. SYNAT and baselines run on a Windows 10 PC, with NVIDIA GeForce RTX 2060 GPU and 32GB RAM.

For RQ1-3, we aim to first compare the SYNAT with the traditional baselines on attack tree synthesizing. Then, we need to compare it with the LLMs and analyze the advantages of SYNAT compared to directly using the LLMs to synthesize the attack trees, extract attack events, and extract their relations. We choose the same *batch_size* of SYNAT to all the baselines, and other parameters are determined by greedy strategy, which can achieve the best performance after tuning these baselines and our approach.

For RQ4, we aim to analyze the ability of synthesized attack trees from security posts and Github issues, to enhance the typical public security bases, i.e., CAPEC [8] and CVE [7], and the private security bases, i.e., HUAWEI's attack tree database.

5 RESULTS

5.1 Performance on attack tree synthesizing

Table 5 illustrates the comparison between SYNAT and the four baselines on attack tree synthesizing, and the best performance of each column is highlighted in **bold face**.

Comparison with Baseline Attack Tree Synthesizers. SYNAT achieves the lowest both AHD (15.03%) and TEDS (13.24%) scores on the performance of synthesizing attack trees, reducing all the baselines by 8.20% (AHD) and 7.13% (TEDS). These results indicate that the attack trees generated by SYNAT are much more similar to the ground truth than the baselines.

Table 5. The comparison between SYNAT and baselines on synthesizing the attack trees (%).

| Method | Model Version | Parameters | Time Cost | Metrics | |
|-------------------|-------------------|------------------|-----------|--------------------------------------|-------------------------------------|
| | | | | AHD | TEDS |
| Structured-Joint+ | Structured-Joint+ | <10M | 15h | 30.75 | 25.15 |
| Albert | Albert-large | 18M | 22.7h | 33.16 | 23.74 |
| T5 | T5-base | 220M | 25h | 26.17 | 20.67 |
| GPT-3 | text-davinci-003 | 175B | 31.5h | 18.44 | 15.06 |
| SYNAT | SYNAT | <10M (w/o GPT-3) | 18h | 10.24 (\downarrow 8.20) | 7.93 (\downarrow 7.13) |

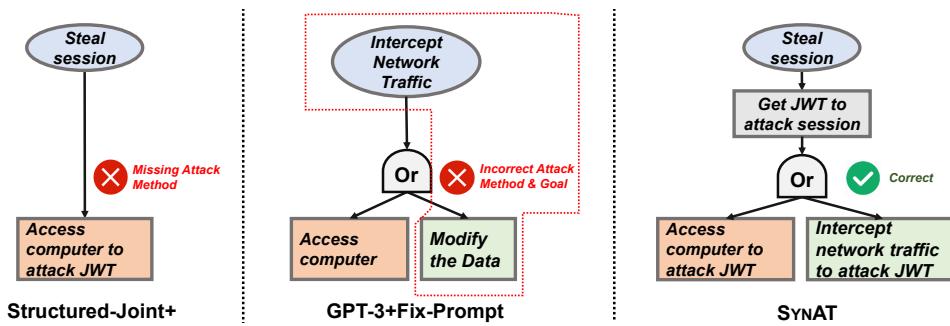


Fig. 6. The case study of SYNAT on Fig. 1's example.

Training/Fine-tuning Hours. The training hours of SYNAT are 18h, which is only longer than the fine-tuning hours of Structured-Joint+. Taking both the comparison results and time efficiency into consideration, we believe that SYNAT has advantages over baselines on synthesizing the attack trees.

Case Study. To qualitatively illustrate the performance of SYNAT, we show the comparison with the example in Fig. 6. We can see that, SYNAT can accurately synthesize the nodes and edges of the attack tree. On the contrary, Structured-Joint+ misses the attack method “intercept network traffic to attack JWT”; GPT-3 incorrectly predicts the attack goal to “intercept network traffic” and predicts an incorrect attack method “modify the data”. Therefore, our method outperforms other baselines in attack tree synthesizing of this case.

Benefits. We believe the advantages of SYNAT mainly come from the accurate prediction of events and relations from the lengthy post. Compared with the baselines, SYNAT can obtain the events in different sentences and accurately analyze the relation between them, so we can accurately transform them to attack trees with some simple tree synthesizing rules.

Answering RQ1: SYNAT outperforms the SOTA baseline in synthesizing the attack trees. It reaches the lowest AHD and TEDS at 10.24% and 7.93%, indicating that the attack trees generated by SYNAT are much more similar to the ground truth. The qualitative analysis further illustrates the effectiveness of SYNAT on the attack tree synthesizing.

5.2 Performance on Event Extraction

Table 6 illustrates the comparison results of SYNAT between six baselines on trigger, instrument, and target detection. The best performance of each column is highlighted in **bold face**.

Comparison with Baselines. SYNAT outperforms all the baselines on average, improving the best baseline in Precision (+9.03%), Recall (+7.03%), and F1 (+8.50%). For each trigger and argument detection, SYNAT also reaches

Table 6. The baseline comparison on event extraction (%).

| Method | Time Cost | Trigger | | | Instrument | | | Target | | | Average | | |
|-------------------|-----------|--------------|--------------|--------------|--------------|--------------|--------------|--------------|--------------|--------------|--------------|--------------|--------------|
| | | Pre. | Rec. | F1 |
| ED3C | 7.7h | 64.17 | 65.35 | 64.75 | 65.46 | 60.45 | 62.86 | 58.76 | 66.84 | 62.54 | 62.80 | 64.21 | 63.38 |
| MLBiNet | 17h | 71.45 | 67.12 | 69.22 | 70.34 | 69.45 | 69.89 | 67.32 | 75.05 | 70.98 | 69.70 | 70.54 | 70.03 |
| Structured-Joint+ | 14.5h | 70.35 | 74.24 | 72.24 | 64.27 | 69.08 | 66.59 | 63.24 | 78.03 | 75.86 | 65.95 | 73.78 | 71.56 |
| GPT-3 | 27h | 72.44 | 73.15 | 72.79 | 70.15 | 71.26 | 70.70 | 70.75 | 77.16 | 73.82 | 71.11 | 73.86 | 72.44 |
| SYNAT | 16h | 83.36 | 81.25 | 82.29 | 81.93 | 79.25 | 80.57 | 77.83 | 82.16 | 79.94 | 81.04 | 80.89 | 80.93 |
| | | (↑10.92) | (↑7.01) | (↑9.50) | (↑11.78) | (↑7.99) | (↑9.87) | (↑7.08) | (↑4.13) | (↑4.08) | (↑9.93) | (↑7.03) | (↑8.50) |

the best performance on Precision, Recall, and F1, with all F1 over 75%. Results indicate that our model is more appropriate to extract events in security posts.

Training/Fine-tuning Hours. The training hours of SYNAT are 16h, which is higher than the simplest event extraction baseline, i.e., ED3C, and the Structured-Joint. Combining both the comparison results and time efficiency, we believe that SYNAT has the advantage of extracting the attack events.

Benefits. We believe the advantages come from two aspects: (1) SYNAT can extract events from attack sentences instead of the complete security post, which can reduce the impact of noise sentences without attack events. (2) SYNAT can combine the information of relations in event extraction, and optimize the event extraction with multiple losses. In this way, SYNAT can adjust the incorrectly extracted events according to relations, which improves its performance on attack event extraction.

Answering RQ2: SYNAT outperforms the four baselines in extracting attack events across the trigger and argument detection, and the average Precision, Recall, and F1 are 81.04%, 80.89%, and 80.93%, outperforming the best baseline by 9.93% (Pre.), 7.03% (Rec.), and 8.50% (F1).

5.3 Performance on Relation Extraction

Table 7 illustrates the comparison results of SYNAT between baselines on ACHIEVEDBY, AND, and OR relation extraction. The best performance of each column is also highlighted in **bold face**.

Table 7. The baseline comparison on relation extraction (%).

| Method | Time Cost | ACHIEVEDBY | | | AND | | | OR | | | Average | | |
|-------------------|-----------|--------------|--------------|--------------|--------------|--------------|--------------|--------------|--------------|--------------|--------------|--------------|--------------|
| | | Pre. | Rec. | F1 |
| KnowledgeILP | 8.5h | 62.06 | 60.45 | 61.24 | 70.33 | 63.44 | 66.71 | 65.87 | 72.13 | 68.86 | 66.09 | 65.34 | 65.60 |
| JC Learning | 16h | 68.44 | 63.17 | 65.70 | 71.45 | 62.60 | 66.73 | 71.42 | 70.59 | 71.00 | 70.44 | 65.45 | 67.81 |
| Structured-Joint+ | 14.5h | 75.94 | 62.10 | 68.33 | 73.32 | 68.75 | 70.96 | 74.66 | 81.72 | 78.03 | 74.67 | 70.86 | 72.44 |
| GPT-3 | 27h | 78.22 | 70.15 | 73.97 | 69.06 | 70.15 | 69.60 | 76.24 | 82.16 | 79.09 | 74.51 | 74.15 | 74.22 |
| SYNAT | 16h | 88.04 | 89.17 | 88.60 | 90.19 | 93.08 | 91.61 | 80.04 | 86.67 | 83.22 | 86.09 | 89.64 | 87.81 |
| | | (↑9.82) | (↑19.02) | (↑14.64) | (↑16.87) | (↑22.93) | (↑20.65) | (↑3.80) | (↑4.51) | (↑4.13) | (↑11.42) | (↑15.49) | (↑13.59) |

Comparison with Baselines. SYNAT outperforms all the baselines on average, improving the best baseline in Precision (+11.42%), Recall (+15.49%), and F1 (+13.59%). For each relation, SYNAT also achieves the highest Precision, Recall, and F1, with all F1 over 80%. These results indicate that SYNAT is more appropriate for extracting the relations in security posts.

Training/Fine-tuning Hours. The training hours of SYNAT are 16h, which is higher than the simplest relation extraction baseline, i.e., KnowledgeILP, and the Structured-Joint+. Combining both the comparison results and time efficiency, we believe that SYNAT has the advantage of extracting the relations between attack events.

Benefits. Apart from the benefits in RQ2, we believe the advantages of relation extraction also come from the constraint loss. For example, if the relation R_{ij} is predicted to be ACHIEVEDBy, then SYNAT will adjust R_{ji} to AND and OR instead of ACHIEVEDBy. Therefore, constraint loss reduces the probability of rings in the attack trees, thus improving the accuracy of relation extraction.

Answering RQ3: SYNAT outperforms the four baselines in extracting AND, OR and ACHIEVEDBy relations, and the average Precision, Recall, F1 are 86.09%, 89.64%, 87.81%, outperforming the best baseline by 11.42% (Pre.), 15.49% (Rec.), 13.59% (F1).

5.4 Performance on Enhancing Security Knowledge Base

In this section, we conduct the practical application of enhancing the public security knowledge base, as well as enhancing and private knowledge base (HUAWEI's attack tree database) with its experienced practitioners.

After training the SYNAT, we first collect 192 security posts with the same StackExchange tool in the previous experiments, dated after Feb 1st, 2023 (*note that these posts are never used in the model training or testing*). Then, we synthesized 121 new attack trees with the trained SYNAT, and each of them is never modified after they are synthesized. Third, we invite three security practitioners in HUAWEI to help identify whether these new attack trees can be used to enhance the security knowledge bases. All the practitioners have more than 10 years of experience in maintaining software security of open-source projects, and they mainly care about the attacks and defenses in Web Applications and MongoDB Database. We contact the security practitioners via internal emails by sending the attack trees to their email addresses and receiving feedback. Besides, we have no communication with these practitioners, so the influences of subjective evaluation can be alleviated.

Enhancing Public Security Knowledge base. Based on the 121 synthesized attack trees we synthesized with SYNAT, we ask the security practitioner to analyze their creation times, i.e., whether the synthesized attack trees are proposed earlier in Stack Overflow than that in the public security knowledge base, i.e., CVE and CAPEC. Although these knowledge bases are comprehensive, they require experienced practitioners to manually analyze the attacks, which may have lags in official information releases. Moreover, if the attacks have not occurred in these public knowledge bases, we ask the practitioners to analyze whether these attack trees are correct and can be treated as new attacks on the community.

We classify the attacks with IDs in **Common Weakness Enumeration (CWE)** [40], which is a widely-used categorization mechanism in these knowledge bases that differentiates the types of vulnerabilities with their causes, behaviors, and consequences. As a result, the attacks are classified into 10 unique CWE-IDs, from CWE-787 to CWE-125, and most of these CWE-IDs are the Top-20 CWE in 2023.

Table 8 shows the results of enhancing the public security knowledge bases CVE and CAPEC with the attack trees synthesized by SYNAT. The column #**SYNAT** shows the number of attack trees in each CWE-ID, column #**Early** indicates the number and ratio of attacks that are proposed in Stack Overflow earlier than the disclosure time, and column #**New** are the number and ratio of attacks that are potentially new to the knowledge base. We can see that 40.50% of the attack trees are created earlier than that in the CVE and 53.72% are earlier than CAPEC. Also, 16.53% of the attack trees may contain new attacks that have not been officially reported in CVE, and 19.83% of them are new to CAPEC. To sum up, SYNAT can help with enhancing the public security knowledge base.

Enhancing Private Security Knowledge Base. Apart from the CVE and CAPEC, we also ask the security practitioners to manually inspect the synthesized 121 attack trees, and finally select trees to replenish their attack knowledge database. The items in HUAWEI's knowledge base are different attack trees, which include 241 attack trees and carry important information such as attack description, attack case, and possible mitigation in terms of STRIDE [73] threat model. The database provides developers with code design specifications and threat modeling analysis of their products. However, the database has not been maintained and updated for a long time,

Table 8. The results of enhancing the public security knowledge base, i.e., CVE and CAPEC.

| Category (CWE) | #SYNAT | CVE | | CAPEC | |
|----------------|--------|----------------|--------------|----------------|--------------|
| | | #Early (Ratio) | #New (Ratio) | #Early (Ratio) | #New (Ratio) |
| CWE-787 | 22 | 8 (36.36%) | 3 (13.64%) | 16 (72.73%) | 3 (13.64%) |
| CWE-79 | 17 | 6 (35.29%) | 4 (23.53%) | 12 (70.59%) | 5 (29.41%) |
| CWE-78 | 15 | 7 (46.67%) | 2 (13.33%) | 7 (46.67%) | 4 (26.67%) |
| CWE-352 | 16 | 6 (37.50%) | 3 (18.75%) | 7 (43.75%) | 3 (18.75%) |
| CWE-190 | 14 | 7 (50.00%) | 3 (21.43%) | 7 (50.00%) | 3 (21.43%) |
| CWE-287 | 11 | 3 (27.27%) | 1 (9.09%) | 4 (36.36%) | 2 (18.18%) |
| CWE-121 | 8 | 5 (62.50%) | 1 (12.50%) | 4 (50.00%) | 1 (12.50%) |
| CWE-138 | 6 | 2 (33.33%) | 1 (16.67%) | 3 (50.00%) | 1 (16.67%) |
| CWE-183 | 7 | 2 (28.57%) | 2 (28.57%) | 2 (28.57%) | 2 (28.57%) |
| CWE-125 | 5 | 3 (60.00%) | 0 (0.00%) | 3 (60.00%) | 0 (0.00%) |
| All | 121 | 49 (40.50%) | 20 (16.53%) | 65 (53.72%) | 24 (19.83%) |

Table 9. The result of enhancing HUAWEI’s security knowledge base.

| Category | #HTWTrees | #SYNAT | #Accept (Ratio) | Increment Ratio |
|------------------------|-----------|--------|-----------------|-----------------|
| Spoofing | 65 | 12 | 9 (75.00%) | 13.85% |
| Tampering | 31 | 44 | 38 (86.36%) | 122.58% |
| Repudiation | 16 | 2 | 2 (100.00%) | 12.50% |
| Information Disclosure | 54 | 41 | 34 (82.93%) | 62.96% |
| Denial of Service | 22 | 9 | 9 (100.00%) | 40.91% |
| Elevation of Privilege | 53 | 13 | 10 (76.92%) | 18.87% |
| ALL | 241 | 121 | 102 (84.30%) | 42.32% |

statistically, 77% of the attack trees have not been updated within four years. Therefore, they urgently need to use external knowledge to update their security knowledge base.

Table 9 shows the results of enhancing HUAWEI’s private security knowledge base, where column #HTWTrees indicates the original number of attack trees in their knowledge bases, #Accept indicates the number of accepted trees, and Increment Ratio shows the increment of the knowledge base. We can see that, 102 out of 121 synthesized attack trees are accepted with a ratio of 84.30%, and the overall increment ratio for the entire database is 42.32%. To sum up, SYNAT helps with enhancing the private attack tree database.

Answering RQ4: SYNAT can be utilized to enhance the public and private security knowledge bases. For the public knowledge bases CVE and CAPEC, 40.50% and 53.72% of them are earlier in Stack Overflow, and 16.53% and 19.83% are potentially new attacks. For the HUAWEI’s knowledge base, 84.30% of the synthesized attack trees are accepted, and the entire database is incremented with 42.32%.

6 HUMAN EVALUATION FOR ATTACK TREE SYNTHESIZING

In this section, we conduct the human evaluation of the quality of synthesized attack trees with its experienced practitioners. First, we obtain the security posts that relate to the 121 new attack trees synthesized by SYNAT (Section 5.4). Then, we apply two representative baselines in the evaluation, i.e., Structured-Joint+ and GPT-3, on these security posts to synthesize the attack trees, where these DL and LLM models obtain the highest performances except for SYNAT. After these two steps, we can separately obtain 121 attack trees for each baseline.

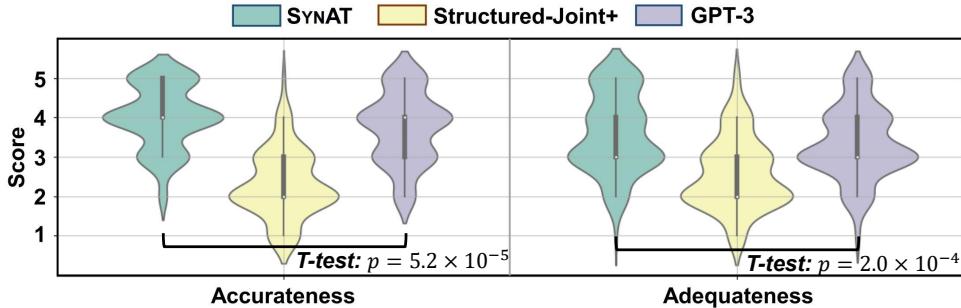


Fig. 7. The results of human evaluation on attack tree synthesizing.

Third, we ask the practitioners of HUAWEI to manually inspect the attack trees synthesized by SYNAT and the two baselines, then rate these attack trees with the following two criteria:

- **Accurateness:** Whether the attack tree is accurate to describe a real attack in software security.
- **Adequateness:** Whether the attack tree is adequate to describe the details of attacks.

where we ask the practitioners to rate 1-5 under the above criteria. For each attack tree, a score of 5 means that the practitioner is satisfied with this attack tree, and a score of 1 means that the attack tree is not satisfactory. Finally, the security practitioners send the feedback through internal email, and we gather each practitioner's feedback on these two criteria and present the distribution of scores in the violin plot.

Fig. 7 shows the score distribution on the two criteria for SYNAT, Structured-Joint+, and GPT-3. We can see that, the maximum distribution score of SYNAT is all over 3 on the three criteria, which is higher than the Structured-Joint+ and GPT-3. The average scores of SYNAT are 3.8, which is also higher than the Structured-Joint+ (2.4) and GPT-3 (3.5). These results indicate that practitioners are generally satisfied with the attack trees extracted by SYNAT. Moreover, we also utilize the significance testing to analyze the significance of SYNAT's improvement to the GPT-3. We choose the T-test with bilateral and independent samples to analyze the differences between SYNAT and GPT-3, which is widely used in significance testing. The result shows that the p -values on accurateness and adequateness are 5.2×10^{-5} and 2.0×10^{-4} , which means the scores of SYNAT are significantly higher than the SOTA LLM baseline, i.e., GPT-3².

Furthermore, we interview these security practitioners regarding the perceived usefulness of SYNAT, and they indicate that SYNAT is quite helpful. The practitioners demonstrate that some attacks, such as **format-string vulnerability** have not been included in the database, which is a common threat encountered in the coding process. SYNAT finds the attack trees with these missing attacks from the security posts, which can be used to replenish the security knowledge. Here is one of the comments:

One of the attack methods from the Format-String attack tree provided by SYNAT is ‘execute harmful code to attack buffer string in printf(buffer)’. When reviewing this, I just realized that my previous code is also vulnerable due to unsafely using the printf function. To make our colleges more cautious, I added this information to our security knowledge base.

7 DISCUSSION

7.1 Usage Scenarios for Designing Appropriate Security Patches

The attack tree can model the software risks that the program may encounter. Compared with the textual description, the attack trees contain detailed attack methods and goals that indicate how the attackers harm the software system. For developers unfamiliar with the current vulnerability they have encountered, the synthesized

²The value $p < 0.01$ indicates the significant differences between two sets of scores.

Table 10. The results of applying SYNAT on designing the security patches for the vulnerable OSS projects.

| Types | Approach | Training Strategy | Fix@1 | Fix@5 | Fix@10 |
|-----------|---------------------------|---------------------|-----------------------|-----------------------|------------------------|
| Manual | Human | - | 20/37 (54.05%) | 23/37 (62.16%) | 24/37 (64.86%) |
| | +HWTrees | - | 22/37 (59.46%) | 25/37 (67.57%) | 28/37 (75.68%) |
| | +SYNAT&HWTrees | - | 30/37 (81.08%) | 32/37 (86.49%) | 37/37 (100.00%) |
| Automatic | CodeT5 | Fine-Tuning | 15/37 (40.54%) | 20/37 (54.05%) | 20/37 (54.05%) |
| | +SYNAT | Fine-Tuning | 25/37 (67.57%) | 32/37 (86.49%) | 33/37 (89.19%) |
| | ChatGPT | In-Context Learning | 23/37 (62.16%) | 24/37 (64.86%) | 24/37 (64.86%) |
| | +SYNAT | In-Context Learning | 29/37 (78.38%) | 30/37 (81.08%) | 31/37 (83.78%) |

attack trees provide some cases with similar attack goals that can guide them in fixing their vulnerabilities. Therefore, the cost of learning a new vulnerability has decreased with the help of synthesized attack trees from applying the SYNAT.

In addition to the previous practical applications, SYNAT has another practical usage scenario, i.e., designing the patches for vulnerable code in the Open Source Software (OSS). Some OSS projects may contain vulnerable code, and if attackers utilize this unfixed vulnerable code, they may deploy exploits to harm the systems [48]. However, it is difficult for developers unfamiliar with software security to understand how the attackers may achieve the attacks and design appropriate patches to fix these vulnerabilities. The synthesized attack trees provide historical information that describes how the attackers achieve the attack goals. This can guide the developers to analyze how the attackers may potentially exploit the vulnerabilities and design appropriate patches based on the attack methods [58]. However, there is currently no large-scale public security knowledge base with cases of attack trees existing in the software community. Although HUAWEI has built an attack tree database, it is private and has not been updated and maintained for a long time, as is illustrated in Section 5.4. In comparison, the SYNAT is an automated approach that is fully trained on the security posts in the Stack Overflow and can be applied to synthesize attack trees from multiple online knowledge-sharing platforms. Previous experiments show that the SYNAT can build a large-scale database that is utilized to enhance public/private security databases.

To evaluate the contribution of SYNAT, we first gather the vulnerability-related issue reports from the GHArchive [17], which is a data source that achieves the latest vulnerability-related issue reports in the software community. These vulnerability-related issue reports indicate the vulnerabilities in the corresponding OSS and the developers describe the details of how they observe these vulnerabilities in their projects, but some vulnerable code is not fixed after the issue report is released. We have manually obtained 37 unfixed vulnerabilities from the GHArchive dataset after Jan 1st, 2024 based on the vulnerability-related issue reports, where some unfixed vulnerabilities may correspond to the attack trees synthesized by SYNAT.

Then, we experiment with two tasks, i.e., manual and automatic security patch designing. **(1) For manual patch designing**, we ask the security practitioners in the HUAWEI to discuss with each other and manually design the patches for the 37 unfixed vulnerabilities, then they utilize the HUAWEI's attack trees (241 trees) as well as SYNAT's synthesized attack trees (121 trees) in Table 9 to help them design the patches; **(2) For automatic patch designing**, since HUAWEI's attack tree database is private, we cannot directly use it in the automatic security patch generation, so the analysis of practical usage on automatic security patching is only conducted on the SYNAT's synthesized 121 attack trees. We introduce two LLMs for code generation, i.e., CodeT5 [68] and ChatGPT [46]. These two LLMs are end-to-end text generation models that can analyze vulnerable code and recommend the appropriate security patches. Compared with other LLMs, fine-tuning the CodeT5 and utilizing in-context learning in ChatGPT can obtain the SOTA performances on fixing the vulnerable code. In both manual and automatic patch designing, we obtain Top-K patches that are most relevant to the vulnerability and calculate

the fixing rate $Fix@K$. The metric is calculated with the following equation.

$$Fix@K = \frac{\#(Trig_Vul@K \cap Fix_Vul@K)}{\#Total_Vul@K} \quad (6)$$

where “#” is the symbol of the number calculation of evaluation samples. The number $\#Total_Vul@K = 37$, and $Fix@K = 1$ if both the vulnerability triggering and fixing are satisfied in Top- K , i.e., $\#(Trig_Vul@K \cap Fix_Vul@K) = 1$, and we make sure that the vulnerable code is triggered and the usefulness of the patch by using the vulnerability detection tools [1, 2] and manual analysis.

Table 10 shows the results of generating the patches for fixing the vulnerabilities with the help of attack trees, where **+HWTrees** indicates that using the HUAWEI’s attack trees to guide the patch designing, and **+SYNAT** means using the attack trees generated by SYNAT. We can see that utilizing the attack trees can significantly improve the fixing rate $Fix@K$ of both manual and automatic patch designing. For manual patch designing, introducing both HUAWEI&SYNAT’s attack trees improves the fixing rate with +35.14% (Fix@10), and the security practitioners accurately fix all 37 vulnerabilities after they refer to the attack trees. For automatic patch designing, introducing the attack trees generated by SYNAT can also improve the Fix@10 with +25.14 (CodeT5) and +18.92 (ChatGPT). In summary, introducing the SYNAT can effectively guide the ability to design appropriate patches for fixing the vulnerabilities.

7.2 Effectiveness of Restricting the Scope of Sentences

To analyze the effectiveness of restricting the sentence scope, we compare the SYNAT with different types of prompt templates, and the number of selected sentences K , on event and relation extraction.

7.2.1 Effect of Prompt Template. The design of the prompt template in LLM affects the performance of SYNAT. First, we compare the original **Cloze Template** with the Q&A format **Prefix Template** [32], which indicates the effect of template types. Then, we compare the template with the **Template without (w/o) Title**, which illustrates the effect of *{Title}* in the template. The content of the two variants are shown as follows:

| Prompt Variants |
|---|
| Prefix-Template: From the following security post {Question, Accepted_Answer} (Title is {Title} to summarize the main topic of the security post), what are the restricted sentences that may contain the attack trees? [Y] (Definition of attack trees.) Task: Please predict the token [Y] with K restricted sentences from the security post’s question and answers to answer this question. The format of the output is the bullet list with Sentence 1 to Sentence K . |
| Template w/o Title: From the following security post {Question, Accepted_Answer}, [Y] are restricted sentences that may contain the attack trees. (Definition of attack trees.) Task: Please predict the token [Y] with K restricted sentences from the security post’s question and answers to make the cloze-testing complete. The output format is the list with Sentence 1...K . |

Fig. 8 (a) illustrates the comparison of these three templates. We can see that, the cloze template outperforms the prefix template on event extraction (+3.2%) and relation extraction (+5.7%), and it also outperforms the template w/o title on event extraction (+2.7%) and relation extraction (4.5%). In summary, the cloze template contributes to the SYNAT than other templates.

7.2.2 Effect of Selected Sentences K . We analyze the effect of selected sentence K on the two tasks F1 in $K \in \{1, 2, \dots, 10\}$. Fig. 8 (b) illustrates the comparison results. We can see that, when $K \in \{0, \dots, 5\}$, the F1 performances of both event and relation extraction are largely increased, where event extraction increases from 0% to 80.9%, and relation extraction increases from 0% to 87.8%. When $K \in \{5, \dots, 10\}$, the F1 performances of event and relation extraction decrease slightly by around 5%. In summary, the sentence-scope limiter contributes to the SYNAT, and SYNAT achieves the optimal value when $K = 5$.

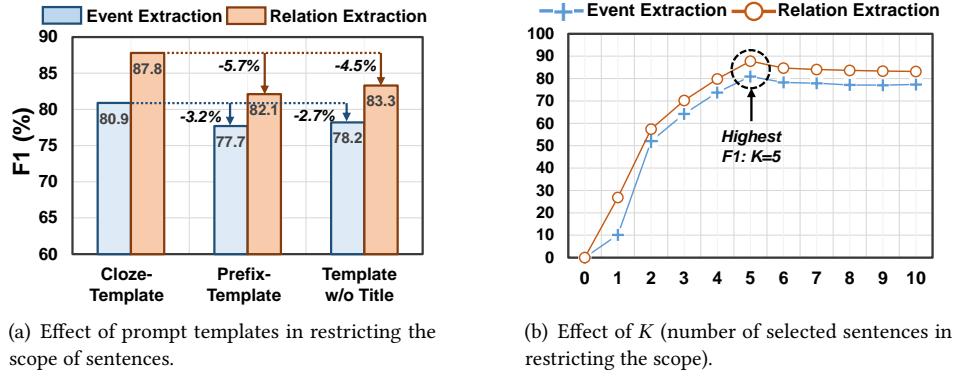


Fig. 8. The effect of restricting the scope of sentences.

7.3 Analysis of Unaccepted Attack Trees

In the human evaluation, we find that 15.7% (19/121) attack trees are not accepted by the analysts. We manually inspect the unaccepted attack trees and find two reasons for their unacceptability: the **Duplicate** and **Low-readability** attack trees.

- **Duplicate Attack Tree (10.7%)**: There are 10.7% (13/121) synthesized attack trees that describe the attacks that have already existed in HUAWEI’s security knowledge base. For example, the attack tree synthesized from post #49168789 describes the server attacking from the XHR request, and we receive the feedback as “*This attack tree accurately describes how the XHR request happens. However, it has already been included in our database: T-Tampering/ServerAttack, so we did not adopt it*”. From the feedback, we can see that, although the synthesized attack tree is fine, it has already been included in the company’s database. To address this issue, we plan to add a module to analyze the similarity between the extracted attack tree and the attack trees in the database, thus excluding the duplications.
- **Low-readability Attack Tree (5.0%)**: There are 5.0% (6/121) synthesized attack trees that are considered hard to understand by security analysts. Since the developer’s discussions on Stack Overflow are typically informal, they are inevitable to contain some *jargon* or *vague expressions*. If contained in our extracted results, it will lower the readability of the attack tree. To address this issue, we plan to add a coreference resolution module [38] to analyze the meaning of jargon. We also plan to add a perplexing measurement to reduce the vague attack methods.

7.4 Threats to Validity

In this section, we aim to discuss the internal, external, and constructive threats of SYNAT and how to mitigate these threats in our approach.

Internal Threats. The internal threat comes from the experiments. Due to the lack of labeled data, the performance rule-based attack tree construction is difficult to evaluate. To alleviate this, we evaluate the practical application of SYNAT, where 84% of attack trees synthesized by rules are accepted. The result indicates that rule-based construction can help build better attack trees. There is also a threat coming from our application study. We cannot guarantee that the evaluations of security analysts on attack trees are all fair. To mitigate this threat, we conduct the training course, thoroughly discussing SYNAT and the extracted attack trees with all the security practitioners, which reduces the occurrence of subjective evaluation to some extent.

External Threats. The external threat comes from the quality of the dataset from the security posts, as well as the data augmentation strategy we use. In this study, we augment the dataset from 5,070 posts to 18,203 posts,

which may lead to the augmented samples having semantic differences from the original ones. We manage this threat by manually inspecting 200 augmented samples. It turns out that 91% of the inspected samples are correct, and only 9% of samples have slight differences.

Constructive Threats. The constructive threat lies in the metrics used in our study. We use the commonly used metrics (Precision, Recall, and F1) to evaluate the performance, and the metrics (AHD and TEDS) to evaluate the performance of the synthesized attack trees. We also manually label the attack events and relations as the ground truth when calculating the performance metrics. This threat can be largely relieved as all the instances are reviewed with a concluding discussion session to resolve any disagreement in labeling.

8 RELATED WORKS

In this section, we discuss the related works of SYNAT. The prior studies relate to manually constructing attack trees and analyzing security online discussions.

Manually Constructing Attack Trees. Many works have recently been proposed to construct attack trees to analyze system risks and defend against attacks. Most of them utilize template or rule-based syntax analysis to construct attack trees based on structural data sources. Traditionally, the attack trees are manually built based on the dependencies, such as Goal-inducing Attack Chains (GACs) and attack graphs [9, 18, 20, 22, 49, 64]. Recently, the attack trees have been constructed based on the abstract description of workflow and identification of hazards. Xu et al. [71] constructed attack trees by transforming fault trees. Lemaire et al. [30] utilized templates to construct attack trees for evaluating the security of cyber-physical systems. Mentel et al. [39] constructed an attack tree from a set of generated implications for the given attack goal by backward chaining. Pinchinat et al. [52] proposed a model-free method by taking an abstract model for some expert knowledge as input. Kern et al. [24] proposed a preliminary system architecture model consisting of hardware network architecture, logical functional architecture with data dependencies, and threat agents to construct attack trees. Different from the previous work, we focus on automatically constructing attack trees from security posts of Stack Overflow based on event and relation extraction, which can extract valuable security information from knowledge-sharing platforms and facilitate secure software development.

Analyzing Security Online Discussion. Recently, more and more developers leveraged open-source community platforms, e.g., GitHub, Stack Overflow, and Security StackExchange, to post their security-related concerns and discuss solutions with other developers. These large-volume security data contain rich information for understanding security discussions. Some researchers, such as Pletea et al. [53] investigated the atmosphere and mood of security discussions on GitHub, finding that developers showed more negativity in security discussions than in non-security discussions. Others studied the topic of security discussions and analyzed the area that most security practitioners care about [29, 41, 72, 74]. Among them, Le et al. [29] and Yang et al. [72] utilize the automatic approach, such as Latent Dirichlet Allocation (LDA) to analyze the discussion topics on Stack Overflow and Security StackExchange. Zahedi et al. [74] used an approach that combined topic modeling with qualitative analysis to summarize 26 security-related topics on GitHub. Our work is different from the previous works as we focus on extracting attack trees from these security conversations, and aim to increase the opportunity to enrich the security database and develop secure software by reusing this attack knowledge.

9 CONCLUSION AND FUTURE WORK

In this paper, we proposed SYNAT, an automated approach to synthesize attack trees from online discussions on Stack Overflow. SYNAT first restricts the sentence scope with the LLM, which may contain the attacks from the security post; then SYNAT utilizes a transition-based event and relation extraction model to extract the events and relations simultaneously; finally, SYNAT constructs the attack trees with a set of heuristic rules from the extracted events and relations. We conducted experiments on 5,070 security posts collected from Stack Overflow

and compared SYNAT with multiple representative baselines on event extraction and relation extraction. The experimental results show that our approach outperforms all the baselines on event and relation extraction. SYNAT was also successfully applied to enhance HUAWEI's security knowledge base and public security knowledge base.

In the future, we plan to further enhance the SYNAT with more extended datasets from other crowd security platforms, such as GitHub Issue Reports (IRs). We also plan to combine the rich-text information, such as code blocks and page screenshots, in the extraction and refine the synthesized attack tree with mitigations.

REFERENCES

- [1] 2024. Wapiti - A simple and fast discriminative sequence labelling toolkit. <https://wapiti.limsi.fr/>.
- [2] 2024. Zed Attack Proxy. <https://www.zaproxy.org/>.
- [3] Leonard Adolphs, Tianyu Gao, Jing Xu, Kurt Shuster, Sainbayar Sukhbaatar, and Jason Weston. 2022. The CRINGE Loss: Learning what language not to model. *CoRR* abs/2211.05826 (2022). <https://doi.org/10.48550/arXiv.2211.05826> arXiv:2211.05826
- [4] Ramón Fernandez Astudillo, Miguel Ballesteros, Tahira Naseem, Austin Blodgett, and Radu Florian. 2020. Transition-based Parsing with Stack-Transformers. In *Findings of the Association for Computational Linguistics: EMNLP 2020, Online Event, 16-20 November 2020 (Findings of ACL, Vol. EMNLP 2020)*, Trevor Cohn, Yulan He, and Yang Liu (Eds.). Association for Computational Linguistics, 1001–1007. <https://doi.org/10.18653/v1/2020.findings-emnlp.89>
- [5] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (Eds.). <https://proceedings.neurips.cc/paper/2020/hash/1457c0d6bfcb4967418bf8ac142f64a-Abstract.html>
- [6] Jinho D. Choi and Martha Palmer. 2011. Transition-based Semantic Role Labeling Using Predicate Argument Clustering. In *Proceedings of the ACL 2011 Workshop on Relational Models of Semantics, RELMS@ACL 2011, Portland, Oregon, USA, June 23, 2011*, Su Nam Kim, Zornitsa Kozareva, Preslav Nakov, Diarmuid Ó Séaghdha, Sebastian Padó, and Stan Szpakowicz (Eds.). Association for Computational Linguistics, 37–45. <https://aclanthology.org/W11-0906>
- [7] M. Corporation. 2022. Common vulnerabilities and exposures. <https://cve.mitre.org/>.
- [8] T. M. Corporation. 2011. Common Attack Pattern Enumeration and Classification (CAPEC). <http://capec.mitre.org/> (2011).
- [9] Jerald Dawkins and John Hale. 2004. A Systematic Approach to Multi-Stage Network Attack Analysis. In *IWIA*. IEEE Computer Society, 48–58. <https://doi.org/10.1109/IWIA.2004.1288037>
- [10] George R. Doddington, Alexis Mitchell, Mark A. Przybocki, Lance A. Ramshaw, Stephanie M. Strassel, and Ralph M. Weischedel. 2004. The Automatic Content Extraction (ACE) Program - Tasks, Data, and Evaluation. In *Proceedings of the Fourth International Conference on Language Resources and Evaluation, LREC 2004, May 26-28, 2004, Lisbon, Portugal*. European Language Resources Association. <http://www.lrec-conf.org/proceedings/lrec2004/summaries/5.htm>
- [11] Chris Dyer, Miguel Ballesteros, Wang Ling, Austin Matthews, and Noah A. Smith. 2015. Transition-Based Dependency Parsing with Stack Long Short-Term Memory. In *ACL*. The Association for Computer Linguistics, 334–343. <https://doi.org/10.3115/v1/p15-1033>
- [12] Stack Exchange. 2022. Stack Exchange Sites. <https://stackexchange.com/sites>.
- [13] Explosion. 2022. Spacy. <https://www.spacy.io>.
- [14] Goran Glavas, Jan Snajder, Marie-Francine Moens, and Parisa Kordjamshidi. 2014. HiEve: A Corpus for Extracting Event Hierarchies from News Stories. In *LREC*. European Language Resources Association (ELRA), 3678–3683.
- [15] Rujun Han, Qiang Ning, and Nanyun Peng. 2019. Joint Event and Temporal Relation Extraction with Shared Representations and Structured Prediction. In *EMNLP-IJCNLP*. Association for Computational Linguistics, 434–444. <https://doi.org/10.18653/v1/D19-1041>
- [16] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for NLP. In *International Conference on Machine Learning*. PMLR, 2790–2799.
- [17] igrigorik/gharchive.org. 2023. GHArchive. <https://www.gharchive.org>.
- [18] Marieta Georgieva Ivanova, Christian W. Probst, René Rydhof Hansen, and Florian Kammüller. 2015. Transforming Graphical System Models to Graphical Attack Models. In *GramSec (Lecture Notes in Computer Science, Vol. 9390)*. Springer, 82–96. https://doi.org/10.1007/978-3-319-29968-6_6
- [19] Ravi Jhawar, Barbara Kordy, Sjouke Mauw, Sasa Radomirovic, and Rolando Trujillo-Rasua. 2015. Attack Trees with Sequential Conjunction. In *SEC (IFIP Advances in Information and Communication Technology, Vol. 455)*. Springer, 339–353. https://doi.org/10.1007/978-3-319-18467-8_23

- [20] Ravi Jhawar, Karim Lounis, Sjouke Mauw, and Yunior Ramírez-Cruz. 2018. Semi-Automatically Augmenting Attack Trees Using an Annotated Attack Tree Library. In *STM (Lecture Notes in Computer Science, Vol. 11091)*. Springer, 85–101. https://doi.org/10.1007/978-3-030-01141-3_6
- [21] Rong Jiang, Rongxing Lu, Ye Wang, Jun Luo, Changxiang Shen, and Xuemin Shen. 2014. Energy-Theft Detection Issues for Advanced Metering Infrastructure in Smart Grid. *Tsinghua Science and Technology* 19, 2 (2014), 105–120.
- [22] Khaled Karray, Jean-Luc Danger, Sylvain Guilley, and M. Abdelaziz Elaabid. 2018. Attack Tree Construction and Its Application to the Connected Vehicle. In *Cyber-Physical Systems Security*. Springer, 175–190. https://doi.org/10.1007/978-3-319-98935-8_9
- [23] Gerzson Kéri and Ákos Kisvölcsay. 2004. On Computing the Hamming Distance. *Acta Cybern* 16, 3 (2004), 443–449.
- [24] Matthias Kern, Bo Liu, Victor Pazmino Betancourt, and Jürgen Becker. 2021. Model-Based Attack Tree Generation for Cybersecurity Risk-Assessments in Automotive. In *ISSE*. IEEE, 1–7. <https://doi.org/10.1109/ISSE51541.2021.9582462>
- [25] Barbara Kordy, Sjouke Mauw, Saša Radomirović, and Patrick Schweitzer. 2012. Attack-Defense Trees. *Journal of Logic and Computation* 24, 1 (06 2012), 55–87. <https://doi.org/10.1093/logcom/exs029>
- [26] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2020. ALBERT: A Lite BERT for Self-supervised Learning of Language Representations. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net. <https://openreview.net/forum?id=H1eA7AEtvS>
- [27] Steven M. LaValle, Michael S. Branicky, and Stephen R. Lindemann. 2004. On the Relationship between Classical Grid Search and Probabilistic Roadmaps. *Int. J. Robotics Res.* 23, 7-8 (2004), 673–692. <https://doi.org/10.1177/0278364904045481>
- [28] Triet Huynh Minh Le, Roland Croft, David Hin, and Muhammad Ali Babar. 2020. Demystifying the Mysteries of Security Vulnerability Discussions on Developer Q&A Sites. *CoRR* abs/2008.04176 (2020). arXiv:2008.04176
- [29] Triet Huynh Minh Le, Roland Croft, David Hin, and Muhammad Ali Babar. 2021. A Large-Scale Study of Security Vulnerability Support on Developer Q&A Websites. In *Evaluation and assessment in software engineering*, 109–118.
- [30] Laurens Lemaire, Jan Vossaert, Bart De Decker, and Vincent Naessens. 2017. Security Evaluation of Cyber-Physical Systems Using Automatically Generated Attack Trees. In *CRITIS (Lecture Notes in Computer Science, Vol. 10707)*. Springer, 225–228. https://doi.org/10.1007/978-3-319-99843-5_20
- [31] Fei Li, Hongzhi Wang, Jianzhong Li, and Hong Gao. 2013. A Survey on Tree Edit Distance Lower Bound Estimation Techniques for Similarity Join on XML Data. *SIGMOD Rec.* 42, 4 (2013), 29–39. <https://doi.org/10.1145/2590989.2590994>
- [32] Xiang Lisa Li and Percy Liang. 2021. Prefix-Tuning: Optimizing Continuous Prompts for Generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, ACL/IJCNLP 2021, (Volume 1: Long Papers), Virtual Event, August 1-6, 2021*, Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli (Eds.). Association for Computational Linguistics, 4582–4597. <https://doi.org/10.18653/v1/2021.acl-long.353>
- [33] Shasha Liao and Ralph Grishman. 2010. Using Document Level Cross-Event Inference to Improve Event Extraction. In *ACL*. The Association for Computer Linguistics, 789–797.
- [34] Zhiwei Lin. 2010. *Tree Similarity Measurement and its Applications*. Ph.D. Dissertation. Ulster University, UK. <https://ethos.bl.uk/OrderDetails.do?uin=uk.bl.ethos.525854>
- [35] Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2023. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *Comput. Surveys* 55, 9 (2023), 1–35.
- [36] Xiao Liu, Yanan Zheng, Zhengxiao Du, Ming Ding, Yujie Qian, Zhilin Yang, and Jie Tang. 2021. GPT Understands, Too. *CoRR* abs/2103.10385 (2021). arXiv:2103.10385 <https://arxiv.org/abs/2103.10385>
- [37] Dongfang Lou, Zhilin Liao, Shumin Deng, Ningyu Zhang, and Huajun Chen. 2021. MLBiNet: A Cross-Sentence Collective Event Detection Network. In *ACL/IJCNLP*. Association for Computational Linguistics, 4829–4839. <https://doi.org/10.18653/v1/2021.acl-long.373>
- [38] Yaojie Lu, Hongyu Lin, Jialong Tang, Xianpei Han, and Le Sun. 2022. End-to-end neural event coreference resolution. *Artif. Intell.* 303 (2022), 103632. <https://doi.org/10.1016/j.artint.2021.103632>
- [39] Heiko Mantel and Christian W. Probst. 2019. On the Meaning and Purpose of Attack Trees. In *CSF*. IEEE, 184–199. <https://doi.org/10.1109/CSF.2019.00020>
- [40] M.Corporation. 2021. Common weakness enumeration. <https://cwe.mitre.org/>.
- [41] Benjamin S Meyers, Nuthan Munaiah, Andrew Meneely, and Emily Prud'hommeaux. 2019. Pragmatic Characteristics of Security Conversations: An Exploratory Linguistic Analysis. In *CHASE*. IEEE, 79–82.
- [42] Sewon Min, Xinxi Lyu, Ari Holtzman, Mikel Artetxe, Mike Lewis, Hannaneh Hajishirzi, and Luke Zettlemoyer. 2022. Rethinking the Role of Demonstrations: What Makes In-Context Learning Work?. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, EMNLP 2022, Abu Dhabi, United Arab Emirates, December 7-11, 2022*. Association for Computational Linguistics, 11048–11064.
- [43] Vidhyashree Nagaraju, Lance Fiondella, and Thierry Wandji. 2017. A Survey of Fault and Attack Tree Modeling and Analysis for Cyber Risk Management. In *HST 2017*. 1–6. <https://doi.org/10.1109/THS.2017.7943455>
- [44] Qiang Ning, Sanjay Subramanian, and Dan Roth. 2019. An Improved Neural Baseline for Temporal Relation Extraction. In *EMNLP-IJCNLP*. Association for Computational Linguistics, 6202–6208. <https://doi.org/10.18653/v1/D19-1642>

- [45] Qiang Ning, Hao Wu, Haoruo Peng, and Dan Roth. 2018. Improving Temporal Relation Extraction with a Globally Acquired Statistical Resource. In *NAACL-HLT*. Association for Computational Linguistics, 841–851. <https://doi.org/10.18653/v1/N18-1077>
- [46] OpenAI. 2023. Chatgpt: A language model for conversational AI. <https://www.openai.com/research/chatgpt/>.
- [47] Liuxuan Pan and Allan Tomlinson. 2016. A Systematic Review of Information Security Risk Assessment. *International Journal of Safety and Security Engineering* 6 (06 2016), 270–281. <https://doi.org/10.2495/SAFE-V6-N2-270-281>
- [48] Shengyi Pan, Jiayuan Zhou, Filipe Roseiro Côgo, Xin Xia, Lingfeng Bao, Xing Hu, Shanping Li, and Ahmed E. Hassan. 2022. Automated unearthing of dangerous issue reports. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2022, Singapore, Singapore, November 14-18, 2022*. ACM, 834–846.
- [49] Stéphane Paul. 2014. Towards Automating the Construction & Maintenance of Attack Trees: a Feasibility Study. In *GramSec (EPTCS, Vol. 148)*, 31–46. <https://doi.org/10.4204/EPTCS.148.3>
- [50] Jorge E. Pérez, Jessica Díaz, Javier García Martín, and Bernardo Tabuenca. 2020. Systematic Literature Reviews in Software Engineering - Enhancement of the Study Selection Process Using Cohen's Kappa Statistic. *J. Syst. Softw.* 168 (2020), 110657. <https://doi.org/10.1016/j.jss.2020.110657>
- [51] Fabio Petroni, Tim Rocktäschel, Sebastian Riedel, Patrick Lewis, Anton Bakhtin, Yuxiang Wu, and Alexander Miller. 2019. Language Models as Knowledge Bases?. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Association for Computational Linguistics, Hong Kong, China, 2463–2473. <https://doi.org/10.18653/v1/D19-1250>
- [52] Sophie Pinchinat, François Schwarzenbauer, and Sébastien Lê Cong. 2020. Library-Based Attack Tree Synthesis. In *GramSec (Lecture Notes in Computer Science, Vol. 12419)*. Springer, 24–44. https://doi.org/10.1007/978-3-030-62230-5_2
- [53] Daniel Pletea, Bogdan Vasilescu, and Alexander Serebrenik. 2014. Security and Emotion: Sentiment Analysis of Security Discussions on Github. In *Proceedings of the 11th working conference on mining software repositories*. 348–351.
- [54] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *J. Mach. Learn. Res.* 21 (2020), 140:1–140:67. <http://jmlr.org/papers/v21/20-074.html>
- [55] Christian Robert. 2014. Machine Learning, a Probabilistic Perspective. *CHANCE* 27 (04 2014), 62–63.
- [56] Gordon Rugg and Peter McGeorge. 1997. The Sorting Techniques: a Tutorial Paper on Card Sorts, Picture Sorts and Item Sorts. *Expert Systems* 14, 2 (1997), 80–93.
- [57] Timo Schick and Hinrich Schütze. 2021. Few-shot text generation with natural language instructions. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. 390–402.
- [58] Bruce Schneier. 1999. Attack trees. *Dr. Dobb's journal* 24, 12 (1999), 21–29.
- [59] Mukul Singh, José Cambrónero, Sumit Gulwani, Vu Le, Carina Negreanu, and Gust Verbruggen. 2023. CodeFusion: A Pre-trained Diffusion Model for Code Generation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*. Association for Computational Linguistics, 11697–11708.
- [60] Jeremy Straub. 2020. Modeling Attack, Defense and Threat Trees and the Cyber Kill Chain, ATT&CK and STRIDE Frameworks as Blackboard Architecture Networks. In *SmartCloud*. IEEE, 148–153. <https://doi.org/10.1109/SmartCloud49737.2020.00035>
- [61] SynAT. 2023. Anonymized Repository/Anonymous Github. <https://anonymous.4open.science/r/SynAT2-82A1/>.
- [62] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, undefinedukasz Kaiser, and Illia Polosukhin. 2017. Attention is All You Need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS'17)*. Curran Associates Inc., Red Hook, NY, USA, 6000–6010.
- [63] Amir Pouran Ben Veyseh, Minh Van Nguyen, Nghia Trung Ngo, Bonan Min, and Thien Huu Nguyen. 2021. Modeling Document-Level Context for Event Detection via Important Context Selection. In *EMNLP*. Association for Computational Linguistics, 5403–5413. <https://doi.org/10.18653/v1/2021.emnlp-main.439>
- [64] Roberto Vigo, Flemming Nielson, and Hanne Riis Nielson. 2014. Automated Generation of Attack Trees. In *CSF*. IEEE Computer Society, 337–350. <https://doi.org/10.1109/CSF.2014.31>
- [65] Haoyu Wang, Muhan Chen, Hongming Zhang, and Dan Roth. 2020. Joint Constrained Learning for Event-Event Relation Extraction. In *EMNLP*. Association for Computational Linguistics, 696–706. <https://doi.org/10.18653/v1/2020.emnlp-main.51>
- [66] Shaolei Wang, Yue Zhang, Wanxiang Che, and Ting Liu. 2018. Joint Extraction of Entities and Relations Based on a Novel Graph Scheme. In *IJCAI*. ijcai.org, 4461–4467. <https://doi.org/10.24963/ijcai.2018/620>
- [67] Yuxuan Wang, Wanxiang Che, Jiang Guo, and Ting Liu. 2018. A Neural Transition-Based Approach for Semantic Dependency Graph Parsing. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*. Sheila A. McIlraith and Kilian Q. Weinberger (Eds.). AAAI Press, 5561–5568. <https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16549>
- [68] Yue Wang, Hung Le, Akhilesh Gotmare, Nghi D. Q. Bui, Junnan Li, and Steven C. H. Hoi. 2023. CodeT5+: Open Code Large Language Models for Code Understanding and Generation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language*

- Processing*, EMNLP 2023, Houda Bouamor, Juan Pino, and Kalika Bali (Eds.). Association for Computational Linguistics, 1069–1088.
- [69] Jason W. Wei and Kai Zou. 2019. EDA: Easy Data Augmentation Techniques for Boosting Performance on Text Classification Tasks. In *EMNLP-IJCNLP*. Association for Computational Linguistics, 6381–6387. <https://doi.org/10.18653/v1/D19-1670>
 - [70] Wenjun Xiong and Robert Lagerström. 2019. Threat Modeling - A Systematic Literature Review. *Comput. Secur.* 84 (2019), 53–69. <https://doi.org/10.1016/j.cose.2019.03.010>
 - [71] Jian Xu, Krishna K. Venkatasubramanian, and Vasiliki Sfyrla. 2016. A Methodology for Systematic Attack Trees Generation for Interoperable Medical Devices. In *SysCon*. IEEE, 1–7. <https://doi.org/10.1109/SYSCON.2016.7490632>
 - [72] Xin-Li Yang, David Lo, Xin Xia, Zhi-Yuan Wan, and Jian-Ling Sun. 2016. What Security Questions Do Developers Ask? A Large-Scale Study of Stack Overflow Posts. *Journal of Computer Science and Technology* 31, 5 (2016), 910–924.
 - [73] Zhimin Yang and Zengguang Zhang. 2007. The Study on Resolutions of STRIDE Threat Model. In *2007 First IEEE International Symposium on Information Technologies and Applications in Education*. 271–273. <https://doi.org/10.1109/ISITAE.2007.4409285>
 - [74] Mansooreh Zahedi, Muhammad Ali Babar, and Christoph Treude. 2018. An Empirical Study of Security Issues Posted in Open Source Projects. In *HICSS*. ScholarSpace / AIS Electronic Library (AISeL), 1–10.
 - [75] Yue Zhang and Stephen Clark. 2011. Syntactic Processing Using the Generalized Perceptron and Beam Search. *Comput. Linguistics* 37, 1 (2011), 105–151. https://doi.org/10.1162/coli_a_00037
 - [76] Suncong Zheng, Feng Wang, Hongyun Bao, Yuexing Hao, Peng Zhou, and Bo Xu. 2017. Joint Extraction of Entities and Relations Based on a Novel Tagging Scheme. In *ACL*. 1227–1236. <https://doi.org/10.18653/v1/P17-1113>