




COTTAGE: Supporting Threat Analysis for Security Novices with Auto-Generated Attack Defense Trees

Keita Yamamoto¹, Masaki Oya¹, Masaki Hashimoto²^a, Haruhiko Kaiya³^b and Takao Okubo¹^c

¹Graduate School of Information Security, Institute of Information Security, Yokohama, Japan

²Faculty of Engineering and Design, Kagawa University, Takamatsu, Japan

³Faculty of Informatics Department of Computer Science, Kanagawa University, Yokohama, Japan

Keywords: Information Security, Threat Modeling, Attack Trees, Attack Defense Trees, CVE, CWE.

Abstract: In software and system development, threat analysis, especially scenario analysis to analyze the steps of an attack, requires a high degree of expertise and long hours of work, which is becoming more and more difficult in short development time, such as Agile and DevOps. In addition, the increased demand for systems has created the need for people without enough security expertise to perform threat analysis. In this paper, we developed COTTAGE, a tool that automatically generates Attack Defense Trees (ADTrees) from CAPEC and CWE knowledge bases and supports the creation of ADTrees that can be used for tree generation by those conducting the analysis. Our evaluation with six security novices demonstrated that COTTAGE enabled participants to perform threat analysis comparable to expert analysis within 30 minutes, whereas experts typically required approximately two days. The case study in a DevOps environment further confirmed COTTAGE's effectiveness in supporting iterative security analysis through automatically generated reference trees.

1 INTRODUCTION


With cyber attacks increasing by 38% globally in 2023 alone (Check Point Research, 2023), security threats are becoming more sophisticated and frequent as digital environments expand. The concept of Security by Design—considering security during product planning and design stages to reduce modification costs and ensure safety—has become an essential paradigm in modern system development (Ross et al., 2016; Lipner, 2004; OWASP Foundation, 2023; Kumar and Goyal, 2020). To effectively implement Security by Design, detailed threat analysis must be conducted early in the development lifecycle to identify potential security vulnerabilities before they become difficult to address.


However, the rapid evolution of software development methodologies has created a fundamental contradiction. While Agile and DevOps approaches accelerate development cycles to meet business demands, the time available for comprehensive security analysis is compressed. This contradiction is particu-


larly challenging amid an expanding security skills gap, with an estimated shortage of approximately 3.4 million cybersecurity professionals worldwide (ISC², 2022). As a result, non-security specialists who lack sufficient expertise to identify complex attack vectors are increasingly responsible for conducting threat analysis.

In this context, threat analysis of information systems has become an essential methodology for realizing Security by Design, serving as a comprehensive security assessment process that includes identifying potential attacks and considering countermeasures. While there are various threat analysis methods, Attack Defense Trees (ADT) are particularly effective as they can hierarchically structure and visualize attack scenarios and defense strategies. However, conducting detailed threat analysis, including ADT, requires specialized knowledge of attack methods and vulnerabilities as well as significant time investment, making its implementation increasingly difficult in modern development environments where security experts are scarce and development cycles are becoming shorter.

This difficulty in implementing threat analysis appears as a common limitation in existing threat modeling approaches. Manual threat analysis requires specialized knowledge and significant time invest-

^a <https://orcid.org/0000-0001-5596-282X>

^b <https://orcid.org/0000-0001-9816-8001>

^c <https://orcid.org/0000-0002-4490-1420>

ment, making it incompatible with shortened development schedules (Myrbakken and Colomo-Palacios, 2017). Automated tools often lack contextual awareness, generating generic results that require expert interpretation. Furthermore, while knowledge bases such as CAPEC and CWE contain valuable attack pattern information, security novices still find it difficult to leverage them effectively.

To address these challenges, we propose COTTAGE (Collaborative Tool of Threat Analysis Gazes at Enemies). COTTAGE is an Attack Defense Trees (ADTrees) (Kordy et al., 2013) generation and management tool specifically designed to bridge the expertise gap in threat analysis. COTTAGE leverages existing security knowledge bases to auto-generate reusable tree patterns, allowing non-specialists to efficiently analyze and customize system-specific threats.

The main contributions of this research are as follows:

- We propose and implement COTTAGE, a tool that automatically generates referenceable ADTrees from knowledge bases and supports users in creating them, demonstrating superior efficiency and analysis accuracy compared to existing tools through empirical evaluation.
- We design a threat analysis approach specifically tailored for DevOps environments that require accelerated security assessments without compromising completeness.
- Through user surveys and case studies, we verify that security novices can use COTTAGE to perform effective threat analysis and generate results comparable to those created by security experts in significantly less time.

The structure of this paper is as follows. Section 2 provides an overview of related research on threat analysis efficiency, knowledge base utilization, threat analysis challenges in DevOps, and existing threat analysis tools. Section 3 explains the design philosophy and implementation details of the proposed COTTAGE tool, including the reference tree generation methods from CAPEC and CWE. Section 4 presents an evaluation by experiment participants and compares their results with those of security experts. Section 5 verifies the effectiveness of the tool through a case study in a DevOps environment. Finally, Section 6 discusses the findings, significance, and limitations of our research.

2 RELATED RESEARCH

With the rise of cybersecurity threats, the importance of threat analysis in system development has increased. However, traditional threat analysis methods require high expertise and substantial time investments, making them challenging to implement in rapid development cycles such as Agile and DevOps. To bridge this gap, various approaches for more efficient threat analysis and knowledge reuse have been researched. This chapter provides an overview of related research that forms the foundation of our work.

2.1 Streamlining Threat Analysis

Researchers have proposed several methods to make threat analysis more efficient. Ivanova et al. (Ivanova et al., 2015) proposed a system model that can be formally transformed into attack trees. Additionally, Audinot et al. (Audinot et al., 2018) suggested calculating the costs to attackers during the attack tree generation process, allowing analysts to focus only on high-probability scenarios.

Several studies have explored the pattern-based reuse of attack tree subtrees. Moore et al. (Moore et al., 2001) proposed finding and abstracting reusable attack patterns from completed attack trees. However, these approaches all require manual processes, and keeping pattern components updated with the latest threat trends creates a significant burden for practitioners.

2.2 Knowledge Base Utilization in Threat Analysis

Techniques for leveraging attack pattern databases such as CAPEC (MITRE Corporation, 2023a) in threat modeling have also been researched. Li et al. (Li et al., 2016) developed a model connecting attack patterns with attacker goals and created inference rules to automatically select attack patterns in context, reducing the burden on security analysts selecting from CAPEC. However, the model still requires manual updates whenever CAPEC is revised, creating an ongoing maintenance challenge.

Regainia et al. (Regainia and Salva, 2017) proposed linking CAPEC, Common Weakness Enumeration (CWE) (MITRE Corporation, 2023b), and security patterns to help software designers select appropriate security patterns more efficiently. This proposal also lacks full automation and requires partial manual implementation.

2.3 Security in DevOps

Kumar et al. proposed an integrated security model (ADOC) that integrates development, security, and operations through security management automation. While their research has similar aims to ours, the proposed ADOC remains a conceptual workflow model that requires more concrete modeling for practical implementation (Kumar and Goyal, 2020). Mohan et al. proposed automation of the deployment process in DevSecOps (Mohan et al., 2018). Casola et al. proposed a methodology for integrating security into DevOps using a Security-by-Design approach (Casola et al., 2020), but there is little mention of the relationship between operations and development necessary for solving our research problem, or of making threat analysis more efficient. Gennari-Cartuan and Goya proposed a security framework applying DevSecOps (Carturan and Goya, 2019), while Lee et al. researched implementing DevSecOps across different groups (Lee, 2018). These studies do not address the conflict between rapid development and security requirements, or the collaboration between operations and development. Diaz et al. proposed a security monitoring infrastructure for continuous feedback from operations to development (Díaz et al., 2019). Their research can be utilized in the section of our research that involves detecting threats and anomalies from log traces (application logs). Rahman et al. addressed the problem of insufficient security in Agile, but their approach attempts to solve it by integrating security experience and skills of personnel (Rahman and Williams, 2016).

2.4 Our Previous Research and the Position of This Study

As direct predecessors to this research, we have worked on several aspects of security threat analysis. Okubo et al. (Okubo and Kaiya, 2022) addressed the contradictory requirements between rapid development in DevOps and comprehensive security analysis by proposing an ADTrees-based analysis method. This method used anomalies detected in the operational phase as starting points and leveraged knowledge bases like CAPEC and CWE to efficiently identify related vulnerabilities and attack methods. While case studies demonstrated the effectiveness of this approach, the need for practical tool implementation and empirical evaluation was also identified.

To facilitate the use of attack pattern knowledge, Oya et al. (Masaki Oya and Okubo, 2025) developed "FRAT," a framework for automatically classifying CAPEC attack patterns into Attack Trees, and

"RATTATA," a tool that supports the creation of Attack Trees by reusing these trees and previously created trees. While these tools contributed to more efficient threat analysis, evaluation revealed several challenges:

1. The information derived from automatically classified trees was limited, requiring more comprehensive information provision
2. The search functionality only supported English attack pattern names, making it difficult to use for non-English speakers
3. There was insufficient support for continuous threat analysis in iterative development environments like DevOps

Beyond our prior research, Kordy et al. (Kordy et al.,) developed and published a tool for generating ADTrees, but updates ceased in 2015, making it unable to incorporate recent threat information.

This research aims to comprehensively address these challenges through COTTAGE (Collaborative Tool of Threat Analysis Gazes at Enemies), an Attack Defense Trees creation tool. COTTAGE builds upon our previous work, enabling even those with limited security expertise to perform high-quality threat analysis in short timeframes.

3 PROPOSED METHOD

COTTAGE enhances the automatic generation of ADTrees by utilizing not only the CAPEC attack pattern knowledge base from our previous research but also the CWE vulnerability knowledge base. CAPEC and CWE contain cross-references—documenting vulnerabilities exploited by each attack pattern and attack patterns that can arise from each vulnerability—enabling more comprehensive threat analysis when integrated.

As shown in Figure 1, we have implemented three types of referenceable tree structures in COTTAGE:

1. CAPEC Reference Trees - placing attack pattern names at the top with prerequisites and mitigations below
2. CWE Reference Trees - placing vulnerability names at the top with mitigations below
3. CAPEC-CWE Integrated Trees - combining attack patterns with related vulnerability information

This approach is particularly valuable in DevOps environments where, as demonstrated in our previous research (Okubo and Kaiya, 2022), operations teams

need to investigate detected attacks using CAPEC to discover potential vulnerabilities or examine identified vulnerabilities to understand possible attack patterns. Additionally, multilingual support makes security knowledge accessible to practitioners worldwide.

The following sections detail the automatic generation process for each type of reference tree in COTTAGE.

3.1 CAPEC Reference Tree Creation Process

The procedure for creating referenceable Attack Defense Trees from CAPEC follows these steps:

1. Download the latest attack pattern information from CAPEC in XML format.
2. Extract "Name," "Description," "Prerequisites," and "Mitigations" text from each attack pattern.
3. Split the "Prerequisites" text so that each condition appears in a separate statement.
4. Translate the text.
5. Create nodes using the translated "Name," "Prerequisites," and "Mitigations."
6. Set "Name" as the parent node and connect all "Prerequisites" and "Mitigations" as child nodes with AND connections.
7. Include "Description" as supplementary information.

Unlike Oya's research (Masaki Oya and Okubo, 2025), our approach does not classify "Prerequisites" into "attack subjects" (enemy actions) and "attack targets" (our system state).

Below is an example of converting "CWE-1007: Insufficient Visual Distinction of Homoglyphs Presented to User" into a tree component. The results after translation in step 4 are as follows:

- Name: Insufficient Visual Distinction of Homoglyphs Presented to User
- Description: This product displays information or identifiers to a user, but the display mechanism does not make it easy for the user to distinguish between visually similar or identical glyphs (homoglyphs), which may cause the user to misinterpret a glyph and perform an unintended, insecure action.
- Mitigation 1:
Implementation
Use a browser that displays Punycode for IDNs in the URL and status bars, or which color codes various scripts in URLs.

Due to the dangers of homoglyph attacks, several browsers now protect against these attacks by using Punycode.

- Mitigation 2:
Implementation
Use an email client with strict filters that prevents messages with mixed character sets from entering the user's inbox.
Some email clients like Google's Gmail prevent the use of non-Latin characters in email addresses or links within emails.

Finally, as shown in Figure 1, the vulnerability name "Name" is placed at the top, with "Potential_Mitigations" arranged underneath to create the CAPEC reference tree.

3.2 CWE Reference Tree Creation Process

The procedure for creating referenceable Attack Defense Trees from CWE follows these steps:

1. Download the latest vulnerability information from the CWE site in XML format.
2. Extract "Name," "Description," and "Potential_Mitigations" text from each vulnerability.
3. Summarize "Potential_Mitigations" text if it is too lengthy.
4. Translate the text.
5. Create nodes using the translated "Name" and "Potential_Mitigations."
6. Set "Name" as the parent node and connect all "Potential_Mitigations" as child nodes with AND connections.
7. Include "Description" as supplementary information.

Below is an example of converting "CWE-1007: Insufficient Visual Distinction of Homoglyphs Presented to User" into a tree component.

Step 2 extraction results in the following texts. The vulnerability name "Name" is taken from the Name attribute of the Weakness tag, "Description" is taken directly from the text, and "Potential_Mitigations" consists of individual "Mitigation" entries, each containing a "Phase" indicating when in development the mitigation can be applied and a "Description" of the specific countermeasure. For readability, we decompose "Potential_Mitigations" by individual "Mitigation" entries.

- Name: Insufficient Visual Distinction of Homoglyphs Presented to User

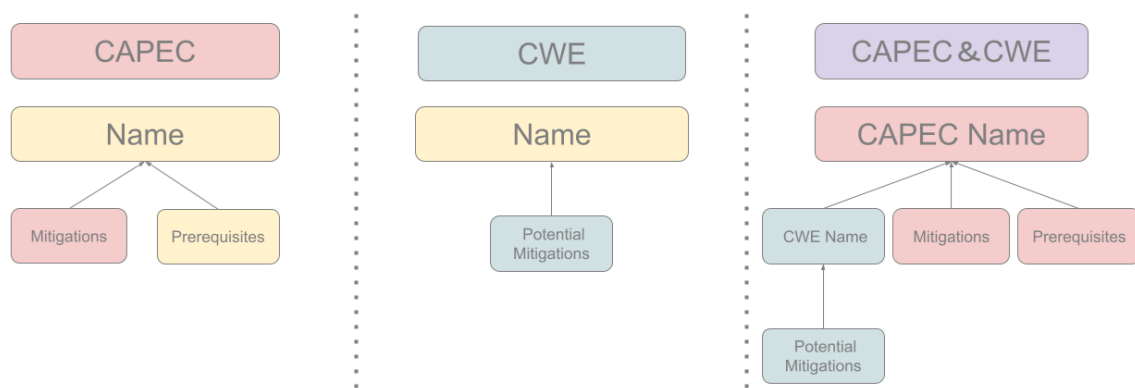


Figure 1: Conceptual Architecture of Referenceable Tree Structures in COTTAGE.

- **Description:** The product displays information or identifiers to a user, but the display mechanism does not make it easy for the user to distinguish between visually similar or identical glyphs (homoglyphs), which may cause the user to misinterpret a glyph and perform an unintended, insecure action.
- **Mitigation_No.1:**
Implementation
Use a browser that displays Punycode for IDNs in the URL and status bars, or which color code various scripts in URLs.
Due to the prominence of homoglyph attacks, several browsers now help safeguard against this attack via the use of Punycode. For example, Mozilla Firefox and Google Chrome will display IDNs as Punycode if top-level domains do not restrict which characters can be used in domain names or if labels mix scripts for different languages.
- **Mitigation_No.2:**
Implementation
Use an email client that has strict filters and prevents messages that mix character sets to end up in a user's inbox.
Certain email clients such as Google's GMail prevent the use of non-Latin characters in email addresses or in links contained within emails. This helps prevent homoglyph attacks by flagging these emails and redirecting them to a user's spam folder.

In step 3, we summarize "Mitigation" texts that are excessively long, as CWE entries often contain detailed explanations that could affect the readability of the entire tree. We use a document summarization library to shorten texts while preserving their meaning. Our guideline is to summarize texts with more than 250 characters (excluding spaces) to under 250 characters. Below is an example of summarizing Mit-

igation_No.1:

- **Mitigation_No.1 original text** (character count excluding spaces):
Implementation(14 characters)
Use a browser that displays Punycode for IDNs in the URL and status bars, or which color code various scripts in URLs.(97 characters)
Due to the prominence of homoglyph attacks, several browsers now help safeguard against this attack via the use of Punycode. For example, Mozilla Firefox and Google Chrome will display IDNs as Punycode if top-level domains do not restrict which characters can be used in domain names or if labels mix scripts for different languages.(280 characters)
In this case, "Due to the prom. ..." is summarized.
- **After summarization (character count):**
"Implementation" and "Use a browser. ..." remain unchanged
Due to the prominence of homoglyph attacks, several browsers now help safeguard against this attack via the use of Punycode.(105 characters)

Step 4 involves translating each text into the target language.

Finally, steps 5 and 6 arrange the attack pattern name "Name," prerequisites "Prerequisites," and mitigations "Mitigations" as shown in Figure 1, with "Name" at the apex and "Prerequisites" and "Mitigations" placed in parallel.

Unlike Oya's research (Masaki Oya and Okubo, 2025), we do not classify "Prerequisites" into "attack subjects" (enemy actions) and "attack targets" (our system state). We decided to include attack targets in the tree, considering that information about implemented rules and mechanisms and what has been implemented is important for threat analysis.

3.3 Tree Integration

The CAPEC-CWE integrated trees are created based on information extracted in Sections 3.2 and 3.1. The specific procedure is as follows:

1. Extract related attack pattern IDs ("Related Attack Patterns") from the CWE XML file.
2. Aggregate by the extracted attack pattern IDs.
3. Connect CWE reference trees associated with the attack pattern IDs to the trees created in Section 3.1 using AND connections.

4 EVALUATION

This chapter discusses the evaluation of the proposed COTTAGE tool. We first explain the evaluation design and methodology, followed by quantitative and qualitative evaluation results.

4.1 Evaluation Design and Methodology

To evaluate the effectiveness of COTTAGE, we conducted an experiment with six students from the Institute of Information Security. Participants were asked to perform threat analysis tasks under the following three conditions:

1. Using only the tree creation functionality of COTTAGE with a general search engine (Search+UI)
2. Using FRAT&RATTATA
3. Using COTTAGE

To compare analysis results, we specified three threat types as top nodes for the threat analysis trees: "impersonation," "denial of service," and "personal information leakage." The tools, threats to be analyzed, and the order of use were varied for each participant to minimize bias.

For the target system, we selected a requirements definition document for a microchip registration web application procured by a Japanese government agency (Ministry of the Environment of Japan, 2023). This document was chosen because it included use cases and was relatively easy to understand in terms of system specifications.

Each participant was given 30 minutes to perform threat analysis, and we evaluated both the number of nodes in the created ADTrees and the qualitative aspects of the analysis. We also compared the participants' results with ADTrees created by a security expert without using COTTAGE to verify the differences between novices and experts.

4.2 Quantitative Evaluation Results

Table 1 shows the aggregated number of nodes created with each tool. Participants A and C created a large number of nodes when using COTTAGE, as they efficiently utilized the citation feature, resulting in a high total number of trees.

Table 1: Comparison of total number of nodes.

ID	Search+UI	FRAT&RATTATA	COTTAGE
A	27	38	162
B	33	18	38
C	24	8	130
D	10	22	41
E	20	23	29
F	24	14	16

For statistical analysis of the collected data, we first performed the Shapiro-Wilk test, which confirmed that the data followed a normal distribution. However, the Bartlett test did not confirm homogeneity of variance. Although analysis of variance did not show statistically significant differences, with a p -value of approximately 0.056, we performed Tukey's multiple comparison test to compare the tools. The results showed that COTTAGE had a p -value of approximately 0.08 compared to FRAT&RATTATA and about 0.1 compared to Search+UI. Despite the small sample size, these low values suggest that COTTAGE is superior as a threat analysis creation tool.

Table 2 compares the number of trees created by participants for each threat type with those created by a security expert who did not use COTTAGE. For convenience, IDs in the table are defined as 1 for the analysis result with the highest number of trees and 2 for the one with fewer trees. The security expert's (Pro) trees omitted recurring parts, but apart from participants who heavily utilized citation, there was no significant difference in numbers. It is noteworthy that while the security expert spent approximately two days creating the trees, participants completed their analysis within a 30-minute time limit. This suggests that COTTAGE significantly contributes to the rapid identification of threats.

Table 2: Comparison of node counts by threat type.

ID	Impersonation	Info. Theft	DoS
1	162	130	29
2	41	38	16
Pro	56	49	13

4.3 Qualitative Evaluation Results

Qualitative comparison of the analysis content for each threat type revealed interesting patterns.

For "impersonation," participants' trees identified voice phishing as a threat, which was overlooked in the expert's analysis. This indicates that knowledge base-derived reference trees can help prevent oversight of specific threat scenarios.

For "information theft," there were many commonalities in technical threats such as SQL injection between participant and expert analyses. However, internal threats such as embedding malicious code during development were not derived in the participants' analyses. This may be attributed to the content of the knowledge base or the participants' security awareness.

For "denial of service," mitigation methods such as implementing load balancing mechanisms were largely consistent. However, threats such as session hijacking and service denial through malware infection, which were derived in the expert's analysis, were not identified in the participants' analyses using COTTAGE. The causes of these differences require further investigation.

In post-experiment surveys and interviews, participants provided positive feedback regarding COTTAGE's usability and efficiency. Several mentioned that the knowledge base citation feature helped promote learning and understanding for beginners. On the other hand, some challenges were also identified, such as difficulty in selecting appropriate results when search results were too numerous.

5 PRACTICAL APPLICATIONS AND EFFECTIVENESS VALIDATION

This chapter describes a case study that verifies the effectiveness of COTTAGE in a DevOps environment as a practical application example, and discusses its potential for use in professional settings.

5.1 Applying COTTAGE in DevOps Environments

To verify whether COTTAGE functions effectively in DevOps environments, we reproduced and evaluated a case study previously conducted by Okubo et al. (Okubo and Kaiya, 2022).

The verification items for this case study were the following two points:

1. Can reference trees support ADTrees analysis?
2. Is COTTAGE effective in DevOps environments, and are there any challenges?

In the research by Okubo et al. (Okubo and Kaiya, 2022), a document sharing system for sharing drafts of entrance examination questions in a university undergraduate program was used as a case study, and analysis using misuse cases and ADTrees was proposed. In our research, we implemented five of the seven situations from that case study:

- Dev1 (Development): A simple implementation where the developer assumes the system will only be used on a secure campus network. Anyone can upload and download manuscripts without personal identification or authentication, although operation logs are recorded.
- Ops1 (Operation): Log analysis reveals that, contrary to assumptions, the system is being accessed from off-campus networks and operated during nighttime and outside business hours. Issues arising from this deviation from assumptions are discovered. Therefore, threat analysis is conducted, and new requirements for system access are identified for the developers. Threats such as impersonation and information leakage are identified.
- Dev2 (Development): Development is carried out based on Ops1 requirements. Countermeasures are implemented against the identified threats, including login functionality with personal verification and authentication based on two roles (professors and staff), logout functionality, and communication path encryption.
- Ops2 (Operation): Logs indicate that professor accounts are being used by multiple users, each accessing from different locations (access points). Also, when a user logs in once and then does not perform any operations for hours or days, the session enters a paused state, but after this paused state continues for a certain period, the same user begins operating again.
- Dev3 (Development): Similarly, development is carried out to meet the requirements from Ops2. The developer introduces session timeout to mitigate the threat of session hijacking using replay techniques. A function for updating user passwords from the system is added.

Figure 2 shows the ADTrees created by Okubo et al. (Okubo and Kaiya, 2022) reproduced using COTTAGE.

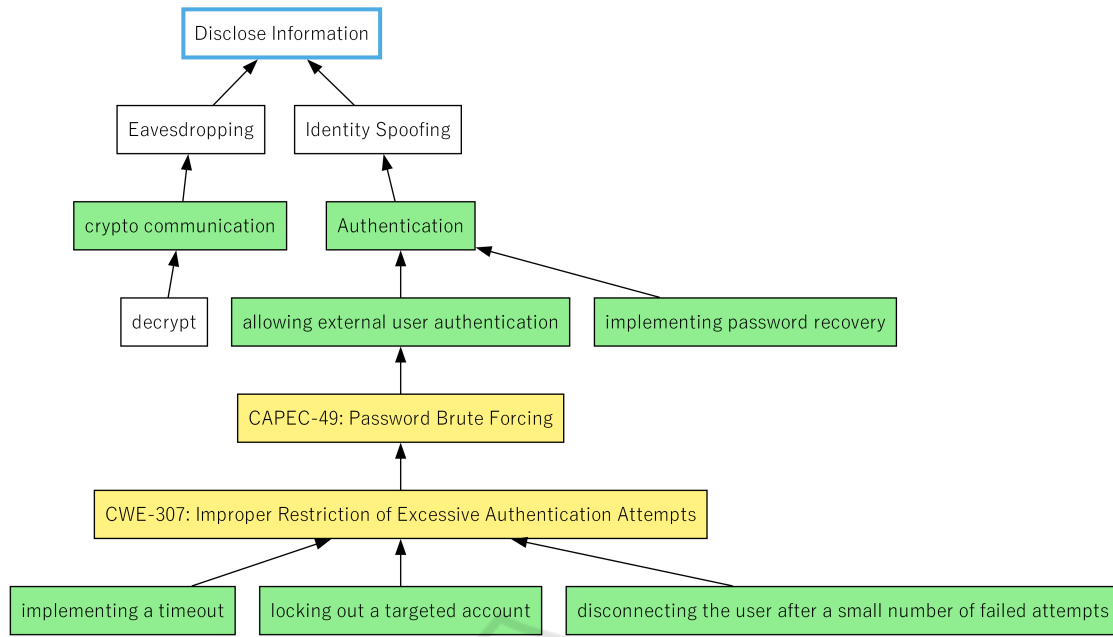


Figure 2: ADTrees created by Okubo et al. in their case study.

5.2 Practical Effectiveness

The trees created in this case study did not exactly match those in (Okubo and Kaiya, 2022), but they were able to derive effective mitigation strategies such as multi-factor authentication, password recovery implementation, and encrypted communication. Therefore, we can evaluate that reference trees can effectively support the creation of ADTrees.

In the research by Okubo et al. (Okubo and Kaiya, 2022), they expected to reduce the effort of threat analysis by constructing ADTrees from logs generated during operation and taking the difference from the previous trees. This expectation was confirmed in the interaction between Ops2 and Dev3, where if the operations team could identify vulnerabilities that occurred in Ops2 and add them to the tree, the development team could compare with previously created trees to identify new features and potential attacks, and consider effective mitigation strategies.

A noteworthy point is that in this case study, even when creating trees by searching from COTTAGE's reference trees using given terms and avoiding unfamiliar terms in the results, a certain level of security could be ensured. This indicates that the tool contributes to the efficiency of tree creation.

From these evaluation results, we conclude that COTTAGE is particularly effective in DevOps environments. In the rapid and iterative development cycles characteristic of DevOps, COTTAGE provides practical value in the following ways:

1. It enables efficient identification of related attack patterns based on vulnerabilities and anomalies discovered in the operation phase
2. It allows for efficient consideration of new threats and countermeasures based on ADTrees created in previous development cycles
3. It enables appropriate threat analysis using knowledge bases even for team members with insufficient security knowledge

It was also suggested that COTTAGE could be an effective tool in development methodologies other than DevOps, particularly in development teams lacking security experts or in situations requiring threat analysis in a short period.

6 DISCUSSION

This chapter discusses the insights and significance obtained through the development and evaluation of COTTAGE, its limitations and challenges, and future research directions.

6.1 Key Findings and Significance

This research proposed and implemented the Attack Defense Trees creation tool COTTAGE with the aim of overcoming the weaknesses of FRAT and RAT-TATA, accommodating a wider range of development

methodologies, and creating a more user-friendly tool for beginners. We achieved the following results:

- Support for DevOps
- Expansion of knowledge bases for reference trees
- Improvement of reference tree searchability
- Development and release of an ADTrees creation tool (COTTAGE)

Regarding DevOps support, we added CWE to the searchable knowledge bases, allowing operations teams to identify vulnerabilities and developers to search for anticipated attack patterns using CAPEC reference trees. This reduced the time required to create ADTrees and enabled even beginners to perform threat analysis in DevOps environments.

For knowledge base expansion, we newly added the CWE vulnerability knowledge base. In the automatic classification framework (FRAT) created by Oya, 58.1% of reference trees contained only attack pattern names. Our improvements allow users to obtain additional information from reference trees for all but 56 attack patterns (approximately 10% of the total 549 patterns) that lack both "Prerequisites" and "Mitigations."

Regarding search improvement, we enhanced the tool's searchability by enabling searches from multilingual pattern names and descriptions, rather than only from English attack pattern names.

Additionally, to help beginners easily perform ADTrees threat analysis, COTTAGE uses reference trees created from knowledge bases without classifying mitigations and prerequisites using Bayesian classifiers as in FRAT. As a result, the speed of automatic reference tree generation was significantly improved from approximately 1 hour with FRAT to about 30 seconds with COTTAGE in the same environment.

These achievements have demonstrated that COTTAGE is a valuable tool that enables security novices to efficiently perform high-quality threat analysis, particularly in rapid development environments such as DevOps.

6.2 Limitations and Challenges

Through the evaluation and verification of COTTAGE, several limitations and challenges were also identified.

In post-survey interviews, some participants mentioned that the search results for attack patterns were too numerous. This may have highlighted the difficulty in determining what poses a threat to their system and what should be applied, now that more patterns can be cited from knowledge bases.

There was also feedback about wanting to standardize the abstraction level of reference trees. We believe this could be addressed with minor modifications to COTTAGE by displaying search results categorized by abstraction level, allowing users to use high-abstraction reference trees in basic design and low-abstraction reference trees in detailed design stages, without modifying the knowledge bases.

A technical challenge is that COTTAGE retrieves text from knowledge base XML files, but encounters issues extracting text from tabs with standalone meanings, as shown by the red line in Figure 3.

In the current implementation, this problem is resolved by visually identifying the two attack patterns where this issue occurs and removing the failing parts before generating the attack patterns. However, this approach cannot accommodate a larger number of registered attack patterns. This is partly an issue with the library being used, but there is a need to modify the system to retrieve text from CSV format files instead for more accurate text extraction. However, CSV format files lack information needed for text appearance, such as ul tags for bullet points, so implementation decisions must be made between prioritizing information accuracy or appearance.

7 CONCLUSION

7.1 Summary of Contributions

In this research, we developed "COTTAGE," a tool that enables even security novices to perform effective threat analysis in modern software environments where rapid development cycles are required. COTTAGE integrates two major security knowledge bases, CAPEC and CWE, to support the creation of Attack Defense Trees (ADTrees).

The main contributions of this research can be summarized in the following three points:

- We proposed and implemented COTTAGE, which automatically generates referenceable ADTrees from knowledge bases, and demonstrated its superiority in efficiency and analysis accuracy compared to existing tools through empirical evaluation.
- We designed a threat analysis approach specifically tailored for DevOps environments, and demonstrated its effectiveness through case studies.
- Through user surveys, we verified that security novices can perform high-quality threat analysis in a short time.

```

<Attack_Pattern ID="42" Name="MIME Conversion" Abstraction="Detailed" Status="Draft">
  <Description>An attacker exploits a weakness in the MIME conversion routine to cause a buffer overflow and gain control over the m
  <Prerequisites>
    <Prerequisite>The target system uses a mail server.</Prerequisite>
    <Prerequisite>Mail server vendor has not released a patch for the MIME conversion routine, the patch itself has a security hole
  </Prerequisites>
  <Mitigations>
    <Mitigation>Stay up to date with third party vendor patches</Mitigation>
    <Mitigation>
      <html:p>Disable the 7 to 8 bit conversion. This can be done by removing the F=9 flag from all Mailer specifications in the
      <html:p>For example, a sendmail.cf file with these changes applied should look similar to (depending on your system and con
      <html:div style="margin-left:10px;" class="informative">Mlocal, P=/usr/libexec/mail.local, F=lsDFMAwS:/|@qrmn, S=10/30, R=2
      <html:br/>Mprog, P=/bin/sh, F=lsDFMoqeu, S=10/30, R=20/40,<html:div style="margin-left:10px;">D=$:/,<html:br/>T=X-Uni
      </html:div>
      <html:p>can be achieved for this the "Mlocal" and "Mprog" Mailers by modifying the ".mc" file to include the following line
      <html:div style="margin-left:10px;" class="informative">define(`LOCAL_MAILER_FLAGS',<html:div style="margin-left:10px;">if
      </html:div>
      <html:br>define(`LOCAL_SHELL_FLAGS',
      <html:div style="margin-left:10px;">ifdef(`LOCAL_SHELL_FLAGS',
      <html:div style="margin-left:10px;">`translit(LOCAL_SHELL_FLAGS, `9')',
      <html:br/>`eu'))

```

Figure 3: Example of failure in extracting text from attack pattern information (part of attack pattern ID: 42).

From the results of our evaluation experiments and case studies, it became clear that COTTAGE provides practical value, especially in environments lacking security experts and in rapid development cycles. By addressing the challenges detailed in Chapter 6 and pursuing the future research directions outlined below, COTTAGE is expected to evolve further and contribute to secure system development.

7.2 Future Research Directions

Based on the challenges identified in this research and the results of practical applications, we suggest the following future research directions.

First, functionality extensions for optimizing search results could be implemented. Features such as displaying search results categorized by abstraction level and filtering based on system context would help users efficiently select appropriate reference trees.

Second, implementing management functions for tree reference history from previous development cycles could further improve analysis consistency and efficiency. This is particularly important in DevOps environments, where development and operation cycles rotate rapidly, making efficient reference to and utilization of past analysis results essential.

Third, integration with Large Language Models (LLMs) shows promise for improving the accuracy of automatic tree generation and enhancement. Using LLMs could enable more appropriate attack pattern recommendations from natural language threat descriptions and contextual mitigation proposals.

Furthermore, automation of continuous updates and expansion of knowledge bases is an important research challenge. Since CAPEC and CWE are updated periodically, a mechanism to automatically reflect these updates in COTTAGE's reference trees is

necessary.

Through these future research directions, COTTAGE can evolve into a more user-friendly and effective threat analysis tool, further enhancing its value as a practical solution to the real-world challenge of security expert shortages in the industry.

ACKNOWLEDGEMENT

We acknowledge the use of Claude AI assistant (Anthropic) for English translation of this paper. This work was supported by JSPS KAKENHI Grant Number 21K11895.

REFERENCES

- Audinot, M., Pinchinat, S., and Kordy, B. (2018). Guided Design of Attack Trees: A System-Based Approach. In *2018 IEEE 31st Computer Security Foundations Symposium (CSF)*, pages 61–75.
- Carturan, S. B. O. G. and Goya, D. H. (2019). A systems-of-systems security framework for requirements definition in cloud environment. In *Proceedings of the 13th European Conference on Software Architecture - Volume 2, ECSA '19*, page 235–240, New York, NY, USA. Association for Computing Machinery.
- Casola, V., De Benedictis, A., Rak, M., and Villano, U. (2020). A novel security-by-design methodology: Modeling and assessing security by slas with a quantitative approach. *Journal of Systems and Software*, 163:110537.
- Check Point Research (2023). 2023 mid-year security report. Technical report, Check Point Software Technologies Ltd.
- Díaz, J., Pérez, J. E., Lopez-Peña, M. A., Mena, G. A., and Yagüe, A. (2019). Self-service cybersecurity monitor-

- ing as enabler for devsecops. *IEEE Access*, 7:100283–100295.
- ISC² (2022). 2022 cybersecurity workforce study. Technical report, International Information System Security Certification Consortium.
- Ivanova, M. G., Probst, C. W., Hansen, R. R., and Kammüller, F. (2015). Transforming Graphical System Models to Graphical Attack Models. In *GraM-Sec@CSF*.
- Kordy, B., Kordy, P., Mauw, S., and Schweitzer, P. Adtool: Security analysis with attack-defense trees. Online. Last updated: 2015, Accessed: 2025-03-01.
- Kordy, B., Kordy, P., Mauw, S., and Schweitzer, P. (2013). Adtool: security analysis with attack–defense trees. In *Quantitative Evaluation of Systems: 10th International Conference, QEST 2013, Buenos Aires, Argentina, August 27-30, 2013. Proceedings 10*, pages 173–176. Springer.
- Kumar, R. and Goyal, R. (2020). Modeling continuous security: A conceptual model for automated devsecops using open-source software over cloud (adoc). *Computers & Security*, 97:101967.
- Lee, J. S. (2018). The devsecops and agency theory. In *2018 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, pages 243–244.
- Li, T., Paja, E., Mylopoulos, J., Horkoff, J., and Beckers, K. (2016). Security attack analysis using attack patterns. In *2016 IEEE Tenth International Conference on Research Challenges in Information Science (RCIS)*, pages 1–13.
- Lipner, S. (2004). The Trustworthy Computing Security Development Lifecycle. In *Proceedings of the 20th Annual Computer Security Applications Conference, ACSAC '04*, pages 2–13, Washington, DC, USA. IEEE Computer Society.
- Masaki Oya, Keita Yamamoto, M. H. and Okubo, T. (2025). Reusable attack tree patterns using common attack pattern enumeration and classification. In *9th International Conference on Cryptography, Security and Privacy, Okinawa, Japan*.
- Ministry of the Environment of Japan (2023). Requirements definition document for microchip registration web application. Online. Published: 2022-06-01, Accessed: 2025-03-01.
- MITRE Corporation (2023a). Common attack pattern enumeration and classification (capec). Online. Accessed: 2025-03-01.
- MITRE Corporation (2023b). Common weakness enumeration (cwe). Online. Accessed: 2025-03-01.
- Mohan, V., ben Othmane, L., and Kres, A. (2018). Bp: Security concerns and best practices for automation of software deployment processes: An industrial case study. In *2018 IEEE Cybersecurity Development (SecDev)*, pages 21–28.
- Moore, A., Ellison, R., and Linger, R. (2001). Attack Modeling for Information Security and Survivability. Technical Report CMU/SEI-2001-TN-001, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA.
- Myrbakken, H. and Colomo-Palacios, R. (2017). Devsecops: A multivocal literature review. In Mas, A., Mesquida, A., O'Connor, R. V., Rout, T., and Dorling, A., editors, *Software Process Improvement and Capability Determination*, pages 17–29, Cham. Springer International Publishing.
- Okubo, T. and Kaiya, H. (2022). Efficient secure devops using process mining and attack defense trees. *Procedia Computer Science*, 207:446–455.
- OWASP Foundation (2023). Security by design principles. OWASP Wiki. Accessed: 2025-02-20.
- Rahman, A. A. U. and Williams, L. (2016). Software security in devops: Synthesizing practitioners' perceptions and practices. In *2016 IEEE/ACM International Workshop on Continuous Software Evolution and Delivery (CSED)*, pages 70–76.
- Regainia, L. and Salva, S. (2017). A methodology of security pattern classification and of Attack-Defense Tree generation. In *3rd International Conference on Information Systems Security and Privacy {(ICISSP)} 2017*, Porto, Portugal, France.
- Ross, R., McEvelley, M., and Oren, J. C. (2016). Systems security engineering: Considerations for a multidisciplinary approach in the engineering of trustworthy secure systems. Technical Report NIST Special Publication 800-160 Vol. 1, National Institute of Standards and Technology. Includes updates as of March 21, 2018.