EvalSVA: Multi-Agent Evaluators for Next-Gen Software Vulnerability Assessment

Anonymous ACL submission

Abstract

Software Vulnerability (SV) assessment is a crucial process of determining different aspects of SVs (e.g., attack vectors and scope) for developers to effectively prioritize efforts in vulnerability mitigation. It presents a challenging and laborious process due to the complexity of SVs and the scarcity of labeled data. To mitigate the above challenges, we introduce EvalSVA, a multi-agent evaluators team to autonomously deliberate and evaluate various aspects of SV assessment. Specifically, we propose a multi-agent-based framework to simulate vulnerability assessment strategies in real-world scenarios, which employs multiple Large Language Models (LLMs) into an integrated group to enhance the effectiveness of SV assessment in the limited data. We also design diverse communication strategies to autonomously discuss and assess different aspects of SV. Furthermore, we construct a multi-lingual SV assessment dataset based on the new standard of CVSS, comprising 699, 888, and 1,310 vulnerability-related commits in C++, Python, and Java, respectively. Our experimental results demonstrate that EvalSVA averagely outperforms the 44.12% accuracy and 43.29% F1 for SV assessment compared with the previous methods. It shows that EvalSVA offers a human-like process and generates both reason and answer for SV assessment. EvalSVA can also aid human experts in SV assessment, which provides more explanation and details for SV assessment.

1 Introduction

Software Vulnerabilities (SVs) are mostly caused by insecure code that can be exploited to attack software systems (Dissanayake et al., 2022; Khan and Parkinson, 2018), and further cause security issues such as systems susceptible to cyber-attacks, and data leakage problems (Le et al., 2023). Over the past decade, the number of SVs has been increasing rapidly (Smyth, 2017), rising from 5,697 in 2013 to 29,065 in 2023 (Statista, 2024). Therefore, SV assessment is a crucial yet challenging problem in security.

The expert-based Common Vulnerability Scoring System (CVSS) (CVS, 2024a) is a widely adopted framework for assessing SVs, which provides metrics to quantify the exploitability, impact, and severity metrics of SVs (CVS, 2024c; Foreman., 2019). Such procedures are labor-intensive and suffer from inefficiencies due to the complexity of vulnerabilities (Bilge and Dumitras, 2012; Feutrill et al., 2018). Traditional automated approaches for SV assessment, primarily reliant on user-submitted reports, are hampered by substantial delays—over 82% of reports are filed more than 30 days post initial detection (Thung et al., 2012). Recent studies aim to automate assess SV via commits (Le et al., 2021; Zhou et al., 2021), greatly reducing reliance on manual expert evaluations and accelerating the assessment process.

However, the existing methods still pose several major challenges that need to be addressed: *Firstly*, the existing methods depend on extensive labeled data, which is difficult to evolve in practice. Specifically, the CVSS framework updates rapidly, evolving from CVSS v2 to v3, and subsequently to v3.1 (CVS, 2024d,b,c). It is time-consuming for experts to furnish high-quality assessments in new standards. For instance, the National Vulnerability Database (NVD) (NIST, 2024) and the Common Vulnerabilities and Exposures (CVE) (CVE, 2024) lists maintained by Mend (WhiteSource, 2023) only contains 699 complete vulnerability entries for C++ from 2013 to 2023. Consequently, the labeled data presents difficulties in industry and limits practical value in real-world scenarios, potentially leading to unreliable performance. *Second*, the previous commit-level SV assessment studies have not started to use the new standards (CVS, 2024c), which incorporate additional metrics (e.g., *Scope* and *User Interaction*) to enhance the complexity of vulnerability and become the current standard in industry. *Additionally*, most of the existing techniques solely predict SV scores of CVSS. They provide no idea about how the vulnerability assessment is derived from the input, making the results difficult to interpret and verify.

To mitigate the above challenges, we propose a multi-agent **EVAL**uators team to autonomously deliberate and evaluate various aspects for **S**oftware **V**ulnerability **A**ssessment, called **EvalSVA**. Specifically, we propose a multi-agent-based framework to simulate vulnerability assessment strategies in real-world scenarios, which employs multiple Large Language Models (LLMs) into an integrated group to enhance the effectiveness of SV assessment in limited data. We also design diverse communication strategies to autonomously discuss, which conduct comprehensive processes and assess different aspects of SV. Moreover, to verify our multi-agent framework in the real-world scenario, we construct the first multi-lingual vulnerability assessment dataset based on the new standard of CVSS, comprising 699, 888, and 1,310 vulnerability-related commits in C++, Python, and Java, respectively. Our case study also shows that EvalSVA offers a human-like process and generates both reason and answer for SV assessment.

We summarize our major contributions as:

- We are the first to propose the multi-agent evaluators with autonomously deliberating for next-gen software vulnerability assessment. Our experimental results demonstrate that EvalSVA averagely outperforms the 44.12% accuracy and 43.29% F1 compared with the single agent.
- We construct the first multi-lingual vulnerability assessment dataset based on the new standard of CVSS, comprising 699, 888, and 1,310 vulnerability-related commits in C++, Python, and Java, respectively.
- We explore the performance of different communication strategies. The results show that EvalSVA can aid human experts in many aspects of SV assessment.

2 Methodology

In this section, we elaborate on the overview of EvalSVA by first introducing SV assessment task formulation and then explaining our evaluators.

2.1 Software Vulnerability Assessment Formulation

Common vulnerability scoring system. The CVSS has emerged as the definitive framework for evaluating the severity of SVs. In this paper, we first employ CVSS v3.1 for the commit-level SV assessment. In this paper, we focus on the prediction of *Base Metrics* due to their broader applicability. These metrics encapsulate the intrinsic attributes of a vulnerability that remain constant over time and across different user environments;

SV assessment task formulation. As shown in Figure 1(a), a vulnerability-related commit can be denoted by the input $\mathcal{X}$ (the template of input $\mathcal{X}$ are shown in Appendix) and SV tasks can be performed in all metrics simultaneously. The goal of EvalSVA is to learn a mapping $\mathcal{F} : \mathcal{X} \mapsto \mathcal{Y}$ from input $\mathcal{X}$ to the output signals $\mathcal{Y}$. Specifically, the output signals for SV assessment tasks can be broadly classified into three aspects: Exploitability, Scope, and Impact. As shown in Figure 1(b), the output signals consists of the security of *Attack Vector (AV)* y_{AV}, *Attack Complexity (AC)* y_{AC}, *Privileges Required (PR)* y_{PR} and *User Interaction (UI)* y_{UI} for exploitability aspect, the security of *Scope Change (S)* y_{S} for scope aspect, and the security of *Confidentiality (C)* y_{C}, *Integrity (I)* y_{I} and *Availability (A)* y_{A} for impact aspect. We then briefly introduce each task of the SV assessment in the CVSS v3.1 as follows:

(1) Exploitability: The exploitability reflects the properties of the vulnerability that lead to a successful attack. In this paper, we use the four metrics to represent the exploitability, including *AV*, *AC*, *PR*, and *UI*. Specifically, the *AV* metric reflects the attack path by which vulnerability exploitation is possible. *AC* metric describes the difficulty of conditions beyond the attacker's control to exploit the vulnerability. *PR* metric assesses the level of authority or access rights that an attacker must acquire to successfully exploit the vulnerability. *UI* metric distinguishes between vulnerabilities that can be exploited solely by attackers and those requiring involvement from a separate user process. For example, Figure 1(a)'s original code (shaded in red) contains a Cross-Site Scripting (XSS) (CWE, 2024b) vulnerability which generally requires "Low" privileges, such as a standard user account and "Required" user interaction with a potentially victim triggering the malicious script. This vulnerability can be exploited through the "Network" attack

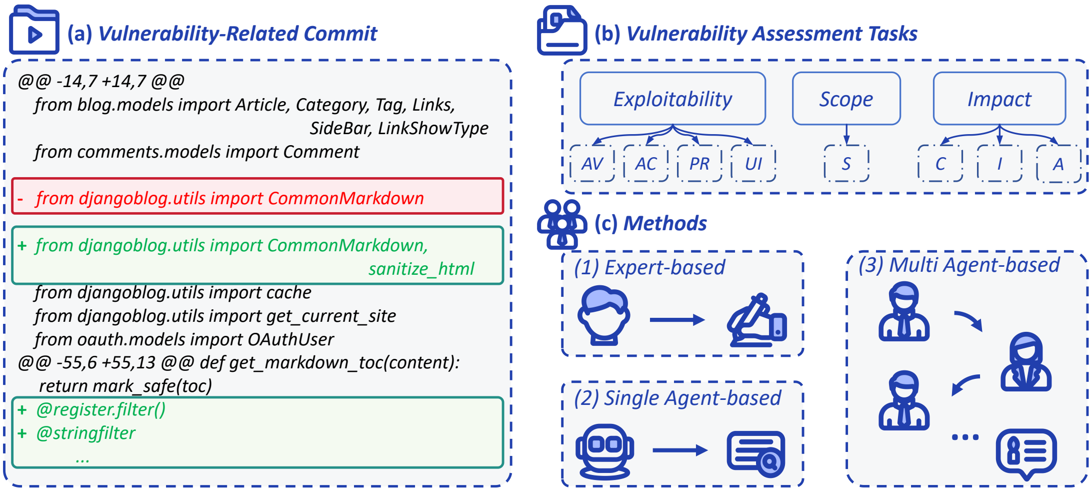


Figure 1: Figure (a) presents the vulnerability commit of CVE-2023-2954 (Detail, 2024a). The code shaded in red and green denote the vulnerability code and corresponding fixed code from commit, respectively. Figure (b) presents the three aspects and eight tasks of SV assessment. Figure (c) presents the three types of SV assessment method.

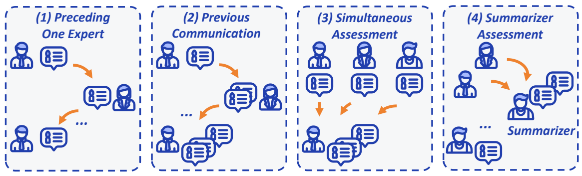


Figure 2: Communication strategy for SV assessment.

Table 1: Statistics of the dataset in C++, Java and Python.

Language	*# Types of Vul*	*# Projects*	*# Commits*	*# Files*
C++	105	169	689	1,506
Java	129	307	888	2,925
Python	159	366	1,310	2,760

path with "Low" complexity via user inputs. **(2) Scope**: It indicates whether exploiting a vulnerability impacts resources beyond its security scope (e.g., application and operating system). *S* determines whether exploiting a vulnerability within a component's scope provides the ability to access or impact the scopes of other components. **(3) Impact**: It captures the consequences of a successfully exploited vulnerability, which can cause losses in *Confidentiality*, *Integrity*, and *Availability*. *Confidentiality* refers to limiting information access and disclosure while preventing unauthorized individuals from gaining access. *Integrity* refers to the trustworthiness and accuracy of information, ensuring that data remains reliable. *Availability* presents the accessibility of information resources, such as processor cycles or disk space.

2.2 Multi Agent Evaluators

2.2.1 Multi Agents and Software Vulnerability Assessment

Various studies (Gao et al., 2023; Peng et al., 2023; Deng et al., 2024) have shown that LLM-based methods are utilized to boost interpretability and practical values behind the classical supervised-based method. Despite the capability of a single LLM to handle a wide range of tasks across multiple domains (Zheng et al., 2023; Imran et al., 2023), it continues to encounter significant challenges in SV assessment. This is primarily due to assessing the severity of vulnerabilities entails a complex and consequential process (Croft et al., 2021), which typically requires collaboration among multiple experts rather than relying solely on individual assessments. These complex situations make it difficult for an existing single LLM to perform well in SV assessment. Inspired by the recent advance in multi-agent methods has demonstrated its effectiveness (Li et al., 2023a; Liang et al., 2023; Huang et al., 2024), we design the first multi-agent-based framework for effectively SV assessment, where the agents interact and communicate within a collaborative environment, aiming to emulate the interaction and collaboration strategies in real-world scenarios (Karpinska et al., 2021). We elaborate on the two components in EvalSVA including vulnerability expert agents and communication strategy.

2.2.2 Component

We provide the details of each component's role and functionality in this section.

1. Vulnerability Expert Agents. Vulnerability expert agents for evaluators constitute a critical component in EvalSVA, where each individual LLM is regarded as an expert agent for SV assessment tasks. For each task related to SV assessment, we meticulously craft unique prompts tailored to the specific requirements of the task. Each LLM is tasked with evaluating the severity of a vulnerability-related commit and subsequently providing a detailed explanation. The responses generated by all agents are preserved within the chat history. This archive of interactions enables subsequent evaluators in future rounds of assessment from prior communications, which mirrors the real-world interactions for SV assessment. It is worth mentioning that each agent evaluates all aspects of the same commit, employing different prompts tailored to specific tasks.

2. Communication Strategy. Another pivotal challenge involves leveraging references from previous expert analyses to construct new prompts that facilitate further exploration by agents. As previously discussed, assessing the multifaceted aspects of vulnerabilities is an intricate and critical process, we are more concerned with how to refer to other expert responses and interpretations for further SV assessment. As shown in Figure 2, we explore four distinct communication strategies to emulate the processes in the real-world scenarios.

(1) Referencing the preceding one expert. Each expert agent constructs its response based on the input from the immediately preceding expert, except the initial agent. We incorporate only the prior agent's response into the current agent's conversational history. It prevents excessive past interactions from influencing present results.

(2) Referencing the previous communication. The expert agents sequentially generate their responses in a predetermined order. This procedure involves concatenating all previous responses into the chat history to construct the assistant's prompt for the next agents. This approach simulates the written communication for SV assessment in the real world, where experts access all prior information and make their judgments accordingly.

(3) Simultaneous assessment. Every expert agent cannot reference the responses of other experts from the current round but may consider the responses from all experts in the previous round. This method minimizes the dependency of an agent on the responses of other experts and mitigates the influences that could arise from sequential order.

(4) Summarizer assessment. Building on the strategy (3), each round additionally augments a summarizer, which synthesizes the responses of all experts within the current round and makes a final judgment. This approach emulates real-world scenarios where conflicting opinions on SV assessments, and introduces an expert specifically designated for decision-making purposes.

3. Adaptive Environment. In EvalSVA, each LLM is treated as an agent that interacts with the adaptive environment. The environment presents two aspects: integration of knowledge from the CVSS standard and coordination with multiple agents from the chat history. The CVSS standard, which can be either predefined or user-modified, is designed to facilitate the rapid integration of new domain knowledge and adapt to evolving standards in SV assessment. The chat history is dynamically produced by each agent. The responses generated by different agents collaboratively contribute to updates in the prompt, enhancing the collaborative process.

3 Experiments

3.1 Data Preparation

Securing high-quality datasets comprising vulnerability-related commits for SV assessment is a formidable challenge, necessitating the demand for qualified expertise.

Data Collection: Our initial step involved acquiring open-source vulnerabilities from Mend (WhiteSource, 2023), which provides extensive vulnerability entries contributed by a community of experts. For each identified vulnerability entry, we extracted security-related commits (i.e., patches) from platforms such as GitHub, Android, and Chrome, recording their associated project and commit messages.

Data Filter: To ensure the relevance and accuracy of our dataset, we employed a filtering methodology to select commits based on two essential criteria: (1) All SV assessment labels must be complete, and (2) The labels for SV assessments must conform to the evaluation standards established by CVSS V3.1. Additionally, we utilized time-based splits for testing the EvalSVA, aiming to closely mimic real-world scenarios where future unseen data is not available.

As presented in Table 1, we have gathered 699, 888, and 1,310 vulnerability-related commits in C++, Python, and Java, respectively. They are col-

Table 2: Dataset evaluation in C++, Java and Python.

Datasets	*Language*	*Accuracy*
Big-Vul (Fan et al., 2020)	C/C++	54.3
D2A (Zheng et al., 2021)	C/C++	28.6
EvalSVA	C++	90.0
	Python	65.0
	Java	70.0

lected according to the CVSS v3.1 standard and encompass 105, 129, and 159 types of vulnerabilities across the 160, 307, and 366 projects, respectively.

3.2 Dataset Evaluation

The previous study (Croft et al., 2023) has demonstrated that vulnerability datasets often exhibit quality problems. Therefore, we conducted an evaluation of our dataset in comparison with existing datasets, despite the absence of specific datasets dedicated to vulnerability assessment. Specifically, we randomly select 20 examples from each programming language and manually analyze the vulnerability. The manual analysis is independently carried out by two developers, each possessing over five years of experience in software security. As presented in Table 2, our dataset demonstrates a higher accuracy compared to previous datasets, underscoring the effectiveness of our data collection and filtration processes.

3.3 Baselines

We primarily focus on few-shot-based methods for SV assessment. This is due to the insufficiency of labeled data for CVSS v3.1 available in programming languages such as C++, Java, and Python. Despite the limited data, these languages pose significant vulnerability threats.

We use the Yin et al. (Yin et al., 2024a) method as baseline, which directly involves a single LLM to generate a response for the given commit (i.e., Single). This approach tests the LLM's ability for SV assessment. For the LLMs utilized in EvalSVA, we have selected ChatGPT (ChatGPT, 2022) and GPT-4 (OpenAI, 2023), given their robust capabilities in handling code-related tasks.

3.4 Evaluation Metrics

In this paper, we employ the evaluation framework delineated by the CVSS v3.1 for SV assessment results derived from various methods. Specifically, we compute the Accuracy (i.e., Acc), which quantifies the ratio of accurately classified instances to the total number of instances, and calculate the F1 score (i.e., F1) to evaluate issues of class imbalance situation.

3.5 EvalSVA Results

As illustrated in Table 3, these LLM-based approach tasks are to achieve consistency with the SV assessment results of human experts in the CVSS v3.1 framework. Our findings reveal that: **(1) SV assessment is an arduous task for a single agent**. Existing single-based LLMs perform poorly across all metrics SV assessment with commit input, with average performances as low as 48.50% and 34.73% on the accuracy and F1 metrics, respectively. This underscores the complexity and difficulty of SV assessment for the single LLM. **(2) Superior performance of EvalSVA.** EvalSVA significantly enhances the performance of the SV assessment process, achieving higher alignment with human preference compared to single-agent-based methods. Specifically, the multi-agent-based method improves the F1 by 53.71% for ChatGPT and 32.88% for GPT-4. This demonstrates EvalSVA's advanced ability to evaluate the different aspects of SV assessment. **(3) GPT-4 can aid human experts in SV assessment.** The ChatGPT method shows more substantial improvements in the exploitability aspect, with average increases of 72.35% and 49.35%, respectively. In contrast, the GPT-4 shows more significant improvements on the impact metric, with an absolute improvement of 5.64% and 12.64% on the accuracy and F1 Score, respectively. Overall, GPT-4 performs well on the *AV*, *PR*, and *UI* metrics, significantly aiding human experts in SV assessment. **(4) SV assessment in Python and Java presents the greatest challenge.** Language-specific results reveal that C++ tasks typically exhibit higher accuracy than Python and Java across all multi-agent methods. This discrepancy may be attributed to C++ providing features like manual memory management and extensive use of pointers. However, these same complexities might make it easier for EvalSVA because they follow certain patterns typical to C++ programming.

We also study the different types of vulnerabilities misreported by EvalSVA. Despite achieving optimal results in various scenarios, we find that EvalSVA still exhibits an error rate with certain types of vulnerabilities, notably those related to XML. For instance, EvalSVA incorrectly reported seven instances of CWE-79 (CWE, 2024b) vulnerabilities in Python, and the single agent reported 23 false positives showing a more severe error rate.

Exploitability Metrics			Attack Vector		Access Complexity		Privileges Required		User Interaction		Average	
Lang	*Baselines*		Acc	F1	Acc	F1	Acc	F1	Acc	F1	Acc	F1
Java	ChatGPT	Single	0.4778	0.4075	0.4000	0.3132	0.2889	0.2425	0.3667	0.3532	0.3834	0.3291
		EvalSVA	0.4889	0.4564	0.4444	0.2167	0.5778	0.3613	0.5000	0.4994	0.5028	0.3835
	GPT-4	Single	0.8111	0.6052	0.5222	0.2338	0.6444	0.4633	0.7111	0.6928	0.6722	0.4988
		EvalSVA	**0.8667**	**0.6296**	**0.5556**	**0.3189**	**0.8333**	**0.6671**	**0.7333**	**0.7091**	**0.7472**	**0.5812**
Python	ChatGPT	Single	0.3206	0.1909	0.2137	0.1318	0.3359	0.3004	0.3282	0.2984	0.2996	0.2304
		EvalSVA	0.3282	0.2014	0.4351	0.2510	0.5954	0.4761	0.4504	0.4496	0.4523	0.3445
	GPT-4	Single	0.8168	**0.3905**	0.1985	0.1311	0.6870	0.5516	0.6718	0.6480	0.5935	0.4303
		EvalSVA	**0.8931**	0.3656	**0.5573**	**0.3251**	**0.7176**	**0.6089**	**0.7557**	**0.7292**	**0.7309**	**0.5072**
C++	ChatGPT	Single	0.3333	0.3088	0.2754	0.1873	0.1449	0.0921	0.5072	0.4569	0.3152	0.2613
		EvalSVA	0.4203	0.3611	0.4928	0.2686	0.6957	0.3058	0.5652	0.5629	0.5435	0.3746
	GPT-4	Single	0.7971	0.5907	0.2174	0.1970	0.8551	0.3073	0.5652	0.5492	0.6087	0.4111
		EvalSVA	**0.8551**	**0.6025**	**0.6667**	**0.4705**	**0.9420**	**0.4851**	**0.5942**	**0.5741**	**0.7645**	**0.5331**

Scope and Impact Metrics			Scope		Confidentiality		Integrity		Availability		Average	
Lang	*Baselines*		Acc	F1	Acc	F1	Acc	F1	Acc	F1	Acc	F1
Java	ChatGPT	Single	0.1444	0.1392	0.5111	0.2591	0.4556	0.2406	0.4556	0.2427	0.3917	0.2204
		EvalSVA	0.4000	0.3619	0.5333	0.3572	0.4889	0.2729	0.4667	0.2390	0.4722	0.3078
	GPT-4	Single	0.4778	0.4173	0.6222	0.4305	0.5333	0.3158	0.5667	0.3579	0.5500	0.3804
		EvalSVA	**0.5556**	**0.4591**	**0.7222**	**0.6458**	**0.6556**	**0.5261**	**0.5667**	**0.4153**	**0.6250**	**0.5116**
Python	ChatGPT	Single	0.2443	0.2133	0.5038	0.3379	0.4504	0.3424	0.3511	0.2471	0.3874	0.2852
		EvalSVA	0.4504	0.4386	0.5115	0.4151	0.4733	0.3914	0.4580	0.3432	0.4733	0.3971
	GPT-4	Single	0.5878	0.5326	0.6412	0.5843	0.5191	0.4082	0.5802	0.4328	0.5821	0.4895
		EvalSVA	**0.6742**	**0.5416**	**0.6718**	**0.6483**	**0.5496**	**0.4981**	**0.7176**	**0.4997**	**0.6533**	**0.5469**
C++	ChatGPT	Single	0.1449	0.1384	0.4638	0.2622	0.4928	0.3192	0.5072	0.2509	0.4022	0.2427
		EvalSVA	0.5217	0.3907	0.5507	0.4029	0.5217	0.3436	0.6812	0.4059	0.5688	0.3858
	GPT-4	Single	0.6087	0.3784	**0.6087**	0.4230	0.6087	0.3739	0.7101	0.3778	0.6341	0.3883
		EvalSVA	**0.7246**	**0.5043**	**0.6087**	**0.5163**	**0.6812**	**0.5935**	**0.7246**	**0.4825**	**0.6848**	**0.5242**

Table 3: Experimental results across Java, Python and C++. We **bold** the best-performing method for each metric.

This count is the highest among all types of vulnerabilities misreported in terms of the AC. Furthermore, both single agent and EvalSVA record 15 false positives in the confidentiality metric, which underscores the ongoing need for EvalSVA to enhance its detection capabilities for XML vulnerabilities.

These findings suggest the potential benefits of incorporating related code snippets for expert agents to better assess language-specific vulnerabilities, particularly for programming languages with complex structures like Python and Java.

3.6 Communication Strategy

To answer Q2, we propose four different communication strategies termed as *preceding one expert*, *previous communication*, *simultaneous assessment*, and *summarizer assessment* for the SV assessment task. We experiment with these strategies in Python and the detailed results are described in Table 4. The remaining experiment results of Java and C++ are presented in Appendix. Our observations indicate that **(1) Employing either communication strategy proves advantageous for SV assessment.** Integrating a multi-agent strategy with ChatGPT results in an improvement of 8.83% and 8.07% in accuracy and F1 score, demonstrating the effectiveness of the communication strategy methodology, respectively. **(2) The efficacy of distinct communication strategies should be tailored to the tasks.** Communication strategies exhibit varying performance depending on the task configuration, which can be attributed to the inherent nature of these tasks. For instance, the evaluation of *attack complexity* and *user interaction* typically falls under binary classification, whereas the impact aspect (including *confidentiality*, *integrity*, and *availability*) requires multi-classification. This underscores the necessity of adopting task-specific communication strategies in the development of SV assessment methods. **(3) The superior performance of *preceding one expert* strategy for most metrics.** *Preceding one expert* strategy demonstrates superior performance in four tasks, yielding significant F1 improvements of 1.32%, 14.54%, 14.31%, and 10.64% in *scope*, *confidentiality*, *integrity*, and *availability*, respectively. It suggests that excessive reliance on the previous references may lead to deviations in the understanding of expert agents.

3.7 Expert Numbers and Rounds

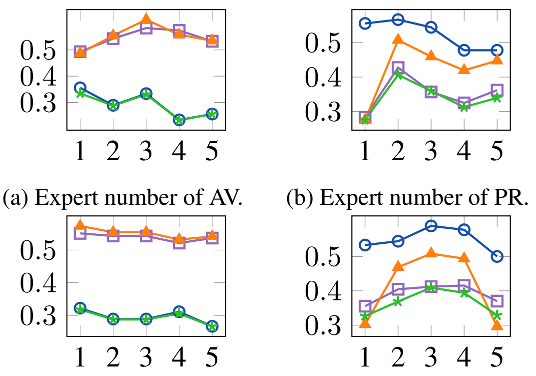
To answer Q3, we conduct the experiment to study the influence of different expert numbers and communication rounds for assessing vulnerability.

Expert Numbers. The number of experts should be selected as medium (2-3). As illustrated in Figure 3 (a)-(b), the correlation between the number of experts and performance demonstrates a pattern of initial improvement followed by a subsequent decrease, with the optimal performance occurring when the number of experts is 2-3. This suggests that diverse expert roles enhance the model's comprehension of SV assessments, aligning with findings reported by (Du et al., 2023; Chan et al., 2023).

Exploitability Metrics	Attack Vector		Access Complexity		Privileges Required		User Interaction		Average	
Communication Strategy	Acc	F1	Acc	F1	Acc	F1	Acc	F1	Acc	F1
Single Agent	0.3206	0.1909	0.2137	0.1318	0.3359	0.3004	0.3282	0.2984	0.2996	0.2304
Previous Communication	0.2366	0.1509	**0.4351**	**0.2510**	**0.5954**	**0.4761**	0.3511	0.3487	0.4046	0.3067
Preceding One Expert	0.2137	0.1398	0.3893	0.2181	0.5496	0.4607	**0.4504**	0.4465	0.4008	0.3163
Simultaneous Assessment	0.2672	0.1697	0.4580	0.2494	0.5878	0.4710	0.4427	0.4426	**0.4389**	0.3332
Summarizer Assessment	**0.3282**	**0.2014**	0.4122	0.2310	0.5573	0.4552	**0.4504**	**0.4496**	0.4370	**0.3343**

Scope and Impact Metrics	Scope		Confidentiality		Integrity		Availability		Average	
Communication Strategy	Acc	F1	Acc	F1	Acc	F1	Acc	F1	Acc	F1
Single Agent	0.2443	0.2133	0.5038	0.3379	0.4504	0.3424	0.3511	0.2471	0.3874	0.2852
Previous Communication	0.4198	0.4190	0.4656	0.3624	0.4275	0.3039	0.4351	0.2921	0.4370	0.3444
Preceding One Expert	**0.4504**	**0.4386**	**0.5115**	**0.4151**	**0.4733**	**0.3914**	**0.4580**	**0.3432**	**0.4733**	**0.3971**
Simultaneous Assessment	0.4351	0.4311	0.4733	0.3445	0.4122	0.2879	0.4351	0.3102	0.4389	0.3434
Summarizer Assessment	**0.4504**	0.4329	0.4122	0.2949	0.4198	0.3368	0.4122	0.2645	0.4237	0.3323

Table 4: Experimental results of different communication strategies of ChatGPT in Python. We bold the best-performing communication strategy for each metric.



(a) Expert number of AV. (b) Expert number of PR.

(c) Round of AV. (d) Round of PR.

Figure 3: The impact of expert numbers and communication rounds on EvalSVA in the Java dataset. The blue, purple, orange, and green lines denote the accuracy, precision, recall and F1 score metrics, respectively.

Furthermore, it indicates that an excessive number of experts involved in the decision-making process may misguide the LLM-based method decisions, potentially due to the extended context length.

Communication Rounds. Multiple rounds of communication are required to facilitate the model's understanding of vulnerability due to its lack of domain-specific knowledge. However, communication across numerous rounds does not necessarily even result in a decline. This could be attributed to the fact that excessively long contexts are detrimental to the model's ability to effectively process the task of SV assessment. It is noteworthy that different tasks may necessitate varying numbers of communication rounds. For instance, the *PR* exhibits optimal performance after three rounds, while *AV* reaches peak performance in the first rounds. These findings underscore the need for a more sophisticated appreciation of the balance between the number of communication rounds and the specific task to optimize performance.

4 Discussion

4.1 Case Study

Figure 4 is a vulnerability example from CVE-2023-46502 (Detail, 2024b), which uses the EvalSVA to evaluate the *Attack Complexity*. The vulnerability arises from the improper configuration of *DocumentBuilderFactory* (shaded in brown in Figure 4), which allows XML external entity attacks (i.e., CWE-611 (CWE, 2024a)). We observe that there initially exists a discrepancy in opinions between the different agents during the first round of responses. Then, a consensus is reached in the subsequent round. This case mirrors real-world situations where multiple experts assess a single vulnerability. Specifically, EvalSVA demonstrates several human-like decision-making processes observed in the industry. **(1) Opinions diversity**: Initially, Expert 1 and Expert 2 present differing judgments when assessing the same vulnerability commit. This diversity broadens the perspective and encompasses a more comprehensive range of considerations in SV assessment. **(2) Revision**: Upon considering the viewpoints of other experts, Expert 1 learns from different aspects and revises its previously erroneous judgment. This indicates that EvalSVA, when informed by the perspectives of multiple experts, possesses the capability to revise. **(3) Interpretability**: Each expert provides explanations for their assessments. This practice aligns with industry standards set by FIRST (fir, 2024), which mandates that CVSS must adhere to documented guidelines and include both the scoring vector and a detailed rationale, enabling others to understand the derivation of the scores. Previous methods (Le et al., 2021; Li et al., 2023b) often provided scores without the explanations needed for comprehensive SV assessment. **(4) Evolutionary adaptation**: EvalSVA can be adapted to differ-

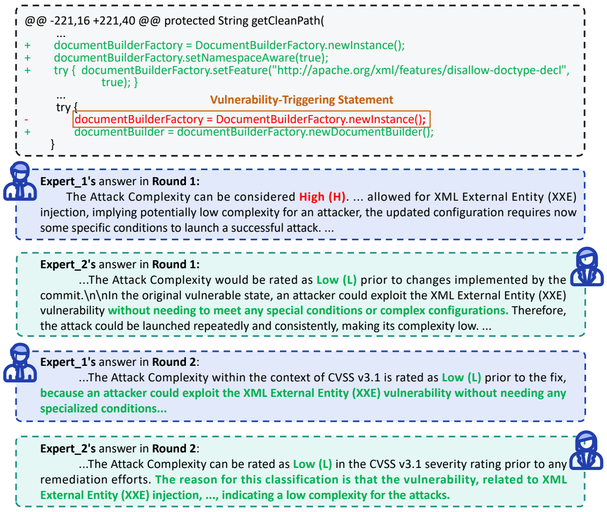


Figure 4: A EvalSVA example presents a discussion process involving two expert agents. The text shaded in red and green denote the wrong and right responses from LLMs, respectively.

ent versions of SV assessment systems based on the prompts. Unlike prior works, EvalSVA swiftly integrates current version-specific domain knowledge to conduct SV assessments without training, demonstrating its agility and relevance in evolving systems.

4.2 Limitation

Transferability on other types of SV assessment. In this paper, we only focus on SV assessment with commit input and CVSS v3.1 standard, excluding SV and bug report-based methods. In the future, we intend to explore the efficacy of EvalSVA regarding the upgrade of the assessment system.

Constraints of domain knowledge in prompts. For the context limited of LLMs, EvalSVA only contains the prompt-based domain knowledge and chat history to facilitate the SV assessment. In the future, we will explore more expert-based examples as prompts for the LLM-based SV assessment.

5 Related Work

Public security databases, such as the NVD (NIST, 2024), and expert-based scoring systems, such as the CVSS (CVS, 2023) have been pivotal in furnishing detailed datasets for SV. In recent years, the CVSS framework has witnessed significant enhancements (Feutrill et al., 2018), evolving from v2 (CVS, 2024d) to v3.0 (CVS, 2024b), and subsequently to v3.1 (CVS, 2024c). Specifically, the existing methods can be broadly divided into two aspects: SV report-based and commit-based methods. The majority of existing methods for automated SV assessment depend on SV reports (i.e., SV reported-based methods) (Han et al., 2017; Lamkanfi et al., 2010; Le et al., 2019; Spanos and Angelis, 2018) from the NVD. These methods typically focus on predicting either a single metric (Fu et al., 2024; Kudjo et al., 2019; Wang et al., 2019) or a set of metrics associated with the CVSS (Le and Babar, 2022; Yamamoto et al., 2015; Ognawala et al., 2018). For instance, Han et al. (Han et al., 2017) introduced a Convolutional Neural Network-based method to automate and predict the overall severity rating by analyzing SV descriptions. However, these user-submitted SV reports often exhibit significant delays (Thung et al., 2012; Sawadogo et al., 2021; Bosu and Carver, 2012; Thongtanunam et al., 2015), potentially exceeding 1000 days. To expedite SV assessment and reduce the extensive labor required by human experts for evaluations, In addition, the recent research also explored the potential of commit-based methods (Le et al., 2021; Zhou et al., 2022; Li et al., 2023b; Yin et al., 2024b). This type of method involves utilizing commit changes to assess all aspects of SVs. For instance, Le et al. (Le et al., 2021) introduced DeepCVA, a model that applies multi-task learning to perform commit-based SV assessment. Li et al. (Li et al., 2023b) proposed a neural framework dedicated to SV detection and assessment simultaneously.

6 Conclusion

In this paper, we propose the first multi-agent-based framework EvalSVA to simulate vulnerability assessment strategies in real-world scenarios. Furthermore, we construct the first multilingual SV assessment dataset based on the new standard of CVSS, comprising 699, 888, and 1,310 vulnerability-related commits in C++, Python, and Java, respectively, which can serve as a foundation dataset for future research. We emphasize the necessity of developing multi-agent evaluators for SV assessment due to the continuous evolution of CVSS. Our experimental results confirm the effectiveness of EvalSVA, especially in scenarios with limited labeled data. We also find that EvalSVA offers a human-like process, providing both rationale and responses for SV assessment. This underscores the effectiveness and possibility of EvalSVA for the next generation of SV assessment.

References

2023. What is cvss score. https://debricked.com/blog/what-is-cvss-score/.

2024a. Common vulnerability scoring system (cvss). https://www.first.org/cvss/.

2024. Common vulnerability scoring system sig. https://www.first.org/cvss/.

2024b. Common vulnerability scoring system v3.0: Specification document. https://www.first.org/cvss/v3.0/specification-document.

2024c. Common vulnerability scoring system v3.1: Specification document. https://www.first.org/cvss/v3.1/specification-document.

2024d. A complete guide to the common vulnerability scoring system version 2.0. https://www.first.org/cvss/v2/guide.

2024a. Cwe-611: Improper restriction of xml external entity reference. https://cwe.mitre.org/data/definitions/611.html.

2024b. Cwe-79: Improper neutralization of input during web page generation ('cross-site scripting'). https://cwe.mitre.org/data/definitions/79.html.

2024. "Common Vulnerabilities and Exposures (CVE)". https://cve.mitre.org/.

Leyla Bilge and Tudor Dumitras. 2012. Before we knew it: an empirical study of zero-day attacks in the real world. In *the ACM Conference on Computer and Communications Security, CCS'12, Raleigh, NC, USA, October 16-18, 2012*, pages 833–844. ACM.

Amiangshu Bosu and Jeffrey C. Carver. 2012. Peer code review in open source communitiesusing reviewboard. In *Proceedings of the ACM 4th Annual Workshop on Evaluation and Usability of Programming Languages and Tools, PLATEAU 2012, Tucson, AZ, USA, October 21, 2012*, pages 17–24. ACM.

Chi-Min Chan, Weize Chen, Yusheng Su, Jianxuan Yu, Wei Xue, Shanghang Zhang, Jie Fu, and Zhiyuan Liu. 2023. Chateval: Towards better llm-based evaluators through multi-agent debate. *CoRR*, abs/2308.07201.

ChatGPT. 2022. Chatgpt. https://chat.openai.com/.

Roland Croft, Muhammad Ali Babar, and M. Mehdi Kholoosi. 2023. Data quality for software vulnerability datasets. In *45th IEEE/ACM International Conference on Software Engineering, ICSE 2023, Melbourne, Australia, May 14-20, 2023*, pages 121–133. IEEE.

Roland Croft, Dominic Newlands, Ziyu Chen, and Muhammad Ali Babar. 2021. An empirical study of rule-based and learning-based approaches for static application security testing. In *ESEM '21: ACM / IEEE International Symposium on Empirical Software Engineering and Measurement, Bari, Italy, October 11-15, 2021*, pages 8:1–8:12. ACM.

Yinlin Deng, Chunqiu Steven Xia, Chenyuan Yang, Shizhuo Dylan Zhang, Shujing Yang, and Lingming Zhang. 2024. Large language models are edge-case generators: Crafting unusual programs for fuzzing deep learning libraries. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering, ICSE 2024, Lisbon, Portugal, April 14-20, 2024*, pages 70:1–70:13. ACM.

CVE-2023-2954 Detail. 2024a. https://nvd.nist.gov/vuln/detail/CVE-2023-2954/.

CVE-2023-46502 Detail. 2024b. https://nvd.nist.gov/vuln/detail/CVE-2023-46502.

Nesara Dissanayake, Asangi Jayatilaka, Mansooreh Zahedi, and Muhammad Ali Babar. 2022. An empirical study of automation in software security patch management. In *37th IEEE/ACM International Conference on Automated Software Engineering, ASE 2022, Rochester, MI, USA, October 10-14, 2022*, pages 7:1–7:13. ACM.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B. Tenenbaum, and Igor Mordatch. 2023. Improving factuality and reasoning in language models through multiagent debate. *CoRR*, abs/2305.14325.

Jiahao Fan, Yi Li, Shaohua Wang, and Tien N. Nguyen. 2020. A C/C++ code vulnerability dataset with code changes and CVE summaries. In *MSR '20: 17th International Conference on Mining Software Repositories, Seoul, Republic of Korea, 29-30 June, 2020*, pages 508–512. ACM.

Andrew Feutrill, Dinesha Ranathunga, Yuval Yarom, and Matthew Roughan. 2018. The effect of common vulnerability scoring system metrics on vulnerability exploit delay. In *Sixth International Symposium on Computing and Networking, CANDAR 2018, Takayama, Japan, November 23-27, 2018*, pages 1–10. IEEE Computer Society.

Park Foreman. 2019. Vulnerability management. Auerbach Publications.

Michael Fu, Chakkrit Tantithamthavorn, Trung Le, Yuki Kume, Van Nguyen, Dinh Q. Phung, and John C. Grundy. 2024. Aibughunter: A practical tool for predicting, classifying and repairing software vulnerabilities. *Empir. Softw. Eng.*, 29(1):4.

Shuzheng Gao, Xin-Cheng Wen, Cuiyun Gao, Wenxuan Wang, Hongyu Zhang, and Michael R. Lyu. 2023. What makes good in-context demonstrations for code intelligence tasks with llms? In *38th IEEE/ACM International Conference on Automated Software Engineering, ASE 2023, Luxembourg, September 11-15, 2023*, pages 761–773. IEEE.

Zhuobing Han, Xiaohong Li, Zhenchang Xing, Hongtao Liu, and Zhiyong Feng. 2017. Learning to predict severity of software vulnerability using only vulnerability description. In *2017 IEEE International Conference on Software Maintenance and Evolution, ICSME 2017, Shanghai, China, September 17-22, 2017*, pages 125–136. IEEE Computer Society.

Jen-tse Huang, Eric John Li, Man Ho Lam, Tian Liang, Wenxuan Wang, Youliang Yuan, Wenxiang Jiao, Xing Wang, Zhaopeng Tu, and Michael R. Lyu. 2024. How far are we on the decision-making of llms? evaluating llms' gaming ability in multi-agent environments. *CoRR*, abs/2403.11807.

Mia Mohammad Imran, Preetha Chatterjee, and Kostadin Damevski. 2023. Uncovering the causes of emotions in software developer communication using zero-shot llms. *CoRR*, abs/2312.09731.

Marzena Karpinska, Nader Akoury, and Mohit Iyyer. 2021. The perils of using mechanical turk to evaluate open-ended text generation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*, pages 1265–1285. Association for Computational Linguistics.

Saad Khan and Simon Parkinson. 2018. Review into state of the art of vulnerability assessment using artificial intelligence. In Simon Parkinson, Andrew Crampton, and Richard Hill, editors, *Guide to Vulnerability Analysis for Computer Networks and Systems - An Artificial Intelligence Approach*, Computer Communications and Networks, pages 3–32. Springer.

Patrick Kwaku Kudjo, Jinfu Chen, Minmin Zhou, Solomon Mensah, and Rubing Huang. 2019. Improving the accuracy of vulnerability report classification using term frequency-inverse gravity moment. In *19th IEEE International Conference on Software Quality, Reliability and Security, QRS 2019, Sofia, Bulgaria, July 22-26, 2019*, pages 248–259. IEEE.

Ahmed Lamkanfi, Serge Demeyer, Emanuel Giger, and Bart Goethals. 2010. Predicting the severity of a reported bug. In *Proceedings of the 7th International Working Conference on Mining Software Repositories, MSR 2010 (Co-located with ICSE), Cape Town, South Africa, May 2-3, 2010, Proceedings*, pages 1–10. IEEE Computer Society.

Triet Huynh Minh Le and Muhammad Ali Babar. 2022. On the use of fine-grained vulnerable code statements for software vulnerability assessment models. In *19th IEEE/ACM International Conference on Mining Software Repositories, MSR 2022, Pittsburgh, PA, USA, May 23-24, 2022*, pages 621–633. ACM.

Triet Huynh Minh Le, Huaming Chen, and Muhammad Ali Babar. 2023. A survey on data-driven software vulnerability assessment and prioritization. *ACM Comput. Surv.*, 55(5):100:1–100:39.

Triet Huynh Minh Le, David Hin, Roland Croft, and Muhammad Ali Babar. 2021. Deepcva: Automated commit-level vulnerability assessment with deep multi-task learning. In *36th IEEE/ACM International Conference on Automated Software Engineering, ASE 2021, Melbourne, Australia, November 15-19, 2021*, pages 717–729. IEEE.

Triet Huynh Minh Le, Bushra Sabir, and Muhammad Ali Babar. 2019. Automated software vulnerability assessment with concept drift. In *Proceedings of the 16th International Conference on Mining Software Repositories, MSR 2019, 26-27 May 2019, Montreal, Canada*, pages 371–382. IEEE / ACM.

Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. 2023a. CAMEL: communicative agents for "mind" exploration of large scale language model society. *CoRR*, abs/2303.17760.

Yi Li, Aashish Yadavally, Jiaxing Zhang, Shaohua Wang, and Tien N. Nguyen. 2023b. Commit-level, neural vulnerability detection and assessment. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2023, San Francisco, CA, USA, December 3-9, 2023*, pages 1024–1036. ACM.

Tian Liang, Zhiwei He, Wenxiang Jiao, Xing Wang, Yan Wang, Rui Wang, Yujiu Yang, Zhaopeng Tu, and Shuming Shi. 2023. Encouraging divergent thinking in large language models through multi-agent debate. *CoRR*, abs/2305.19118.

NIST. 2024. "National Vulnerability Database (NVD)". https://nvd.nist.gov/.

Saahil Ognawala, Ricardo Nales Amato, Alexander Pretschner, and Pooja Kulkarni. 2018. Automatically assessing vulnerabilities discovered by compositional analysis. In *Proceedings of the 1st International Workshop on Machine Learning and Software Engineering in Symbiosis, MASES@ASE 2018, Montpellier, France, September 3, 2018*, pages 16–25. ACM.

OpenAI. 2023. GPT-4 technical report. *CoRR*, abs/2303.08774.

Yun Peng, Chaozheng Wang, Wenxuan Wang, Cuiyun Gao, and Michael R. Lyu. 2023. Generative type inference for python. In *38th IEEE/ACM International Conference on Automated Software Engineering, ASE 2023, Luxembourg, September 11-15, 2023*, pages 988–999. IEEE.

Arthur D. Sawadogo, Quentin Guimard, Tegawendé F. Bissyandé, Abdoul Kader Kaboré, Jacques Klein, and Naouel Moha. 2021. Early detection of security-relevant bug reports using machine learning: How far are we? *CoRR*, abs/2112.10123.

Vincent Smyth. 2017. Software vulnerability management: how intelligence helps reduce the risk. *Netw. Secur.*, 2017(3):10–12.

Georgios Spanos and Lefteris Angelis. 2018. A multi-target approach to estimate software vulnerability characteristics and severity scores. *J. Syst. Softw.*, 146:152–166.

Statista. 2024. Number of common it security vulnerabilities and exposures (cves) worldwide from 2009 to 2024 ytd. https://www.statista.com/statistics/500755/worldwide-common-vulnerabilities-and-exposures/.

Patanamon Thongtanunam, Shane McIntosh, Ahmed E. Hassan, and Hajimu Iida. 2015. Investigating code review practices in defective files: An empirical study of the qt system. In *12th IEEE/ACM Working Conference on Mining Software Repositories, MSR 2015, Florence, Italy, May 16-17, 2015*, pages 168–179. IEEE Computer Society.

Ferdian Thung, David Lo, Lingxiao Jiang, Lucia, Foyzur Rahman, and Premkumar T. Devanbu. 2012. When would this bug get reported? In *28th IEEE International Conference on Software Maintenance, ICSM 2012, Trento, Italy, September 23-28, 2012*, pages 420–429. IEEE Computer Society.

Peichao Wang, Yun Zhou, Baodan Sun, and Weiming Zhang. 2019. Intelligent prediction of vulnerability severity level based on text mining and xgbboost. In *Eleventh International Conference on Advanced Computational Intelligence, ICACI 2019, Guilin, China, June 7-9, 2019*, pages 72–77. IEEE.

WhiteSource. 2023. “Mend bolt”. https://www.mend.io/free-developer-tools/.

Yasuhiro Yamamoto, Daisuke Miyamoto, and Masaya Nakayama. 2015. Text-mining approach for estimating vulnerability score. In *4th International Workshop on Building Analysis Datasets and Gathering Experience Returns for Security, BADGERS@RAID 2015, Kyoto, Japan, November 5, 2015*, pages 67–73. IEEE.

Xin Yin, Chao Ni, and Shaohua Wang. 2024a. Multitask-based evaluation of open-source LLM on software vulnerability. *CoRR*, abs/2404.02056.

Xin Yin, Chao Ni, and Shaohua Wang. 2024b. Multitask-based evaluation of open-source LLM on software vulnerability. *CoRR*, abs/2404.02056.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Yunhui Zheng, Saurabh Pujar, Burn L. Lewis, Luca Buratti, Edward A. Epstein, Bo Yang, Jim Laredo, Alessandro Morari, and Zhong Su. 2021. D2A: A dataset built for ai-based vulnerability detection methods using differential analysis. In *43rd IEEE/ACM International Conference on Software Engineering: Software Engineering in Practice, ICSE (SEIP) 2021, Madrid, Spain, May 25-28, 2021*, pages 111–120. IEEE.

Jiayuan Zhou, Michael Pacheco, Zhiyuan Wan, Xin Xia, David Lo, Yuan Wang, and Ahmed E. Hassan. 2021. Finding A needle in a haystack: Automated mining of silent vulnerability fixes. In *36th IEEE/ACM International Conference on Automated Software Engineering, ASE 2021, Melbourne, Australia, November 15-19, 2021*, pages 705–716. IEEE.

Yaqin Zhou, Jing Kai Siow, Chenyu Wang, Shangqing Liu, and Yang Liu. 2022. SPI: automated identification of security patches via commits. *ACM Trans. Softw. Eng. Methodol.*, 31(1):13:1–13:27.

Appendix

A Prompt Template

The example of a prompt template is illustrated in Figure 5. We incorporate numerous commit details, such as commit information, CVE Description, Commit message, and domain knowledge of CVSS. In this context, we substitute the highlighted (red) square brackets with corresponding information from each commit before querying the LLMs.

B Additional Communication Strategies Results

In this appendix, we present the detailed experiment results that focus on different communication strategies for Java and C++. Our study is also conducted in ChatGPT as Q2.

C Additional Experimental Setting

C.1 Implementation Details

For ChatGPT ("gpt-3.5-*turbo-0125*") and GPT-4 ("gpt-4-*turbo*"), we use the public APIs provided by OpenAI. To mitigate the risk of data leakage and effectively evaluate the methods' ability for SV assessment, we adopt a time-split setting based on the "commit date" of vulnerability patches. Specifically, the vulnerability-related commit after 2023-11-27 of Python, 2023-11-28 of Java, and 2023-12-02 of C++ are designated for testing in this paper.

C.2 Additional Metrics

We also use the following two widely used performance metrics for SV assessment:

Precision: It is the ratio of true positives (TP) to the sum of true positives and false positives (FP), calculated following: Precision $= \frac{\text{TP}}{\text{TP+FP}}$.

Recall: It is the ratio of TP to the sum of TP and false negatives (FN), calculated following: Recall $= \frac{\text{TP}}{\text{TP+FN}}$.

D Algorithm of Communication Strategies

In this appendix, we formally define the following four different communication strategies, including *preceding one expert* (Refer to Algorithm 1), *previous communication*(Refer to Algorithm 2), *simultaneous assessment*(Refer to Algorithm 3), and *summariser assessment*(Refer to Algorithm 4) for the SV assessment task.

Algorithm 1 Referencing the preceding one expert

Require: Agents Number: N, Expert Agents: $E_1, E_2, \cdots, E_N$, Communication Rounds: R, Chat History List $History$, Vulnerability Assessment Tasks $T_1, T_2, \cdots, T_8$
Ensure: Results for Vulnerability Assessment Task $Answer$

Initialize a vulnerability assessment task T_i
for $r \leftarrow 0$ to R **do**
  for $n \leftarrow 0$ to N **do**
    if $History \neq \emptyset$ **then**
      $h_{role} \leftarrow E_n$
      $h_{answer} \leftarrow E_n(T_i, History)$
      $History \leftarrow \{h_{role}, h_{answer}\}$
    else
      $h_{role} \leftarrow E_n$
      $h_{answer} \leftarrow E_n(T_i)$
      $History \leftarrow \{h_{role}, h_{answer}\}$
    end if
    $Answer \leftarrow$ Final h_{answer}
  end for
end for
return $Answer$

Algorithm 2 Referencing the previous communication

Require: Agents Number: N, Expert Agents: $E_1, E_2, \cdots, E_N$, Communication Rounds: R, Chat History List $History$, Vulnerability Assessment Tasks $T_1, T_2, \cdots, T_8$
Ensure: Results for Vulnerability Assessment Task $Answer$

Initialize a vulnerability assessment task T_i
for $r \leftarrow 0$ to R **do**
  for $n \leftarrow 0$ to N **do**
    if $History \neq \emptyset$ **then**
      $h_{role} \leftarrow E_n$
      $h_{answer} \leftarrow E_n(T_i, History)$
      $History \leftarrow \{h_{role}, h_{answer}\}$
    else
      $h_{role} \leftarrow E_n$
      $h_{answer} \leftarrow E_n(T_i)$
      $History \leftarrow History + \{h_{role}, h_{answer}\}$
    end if
    $Answer \leftarrow$ Final h_{answer}
  end for
end for
return $Answer$

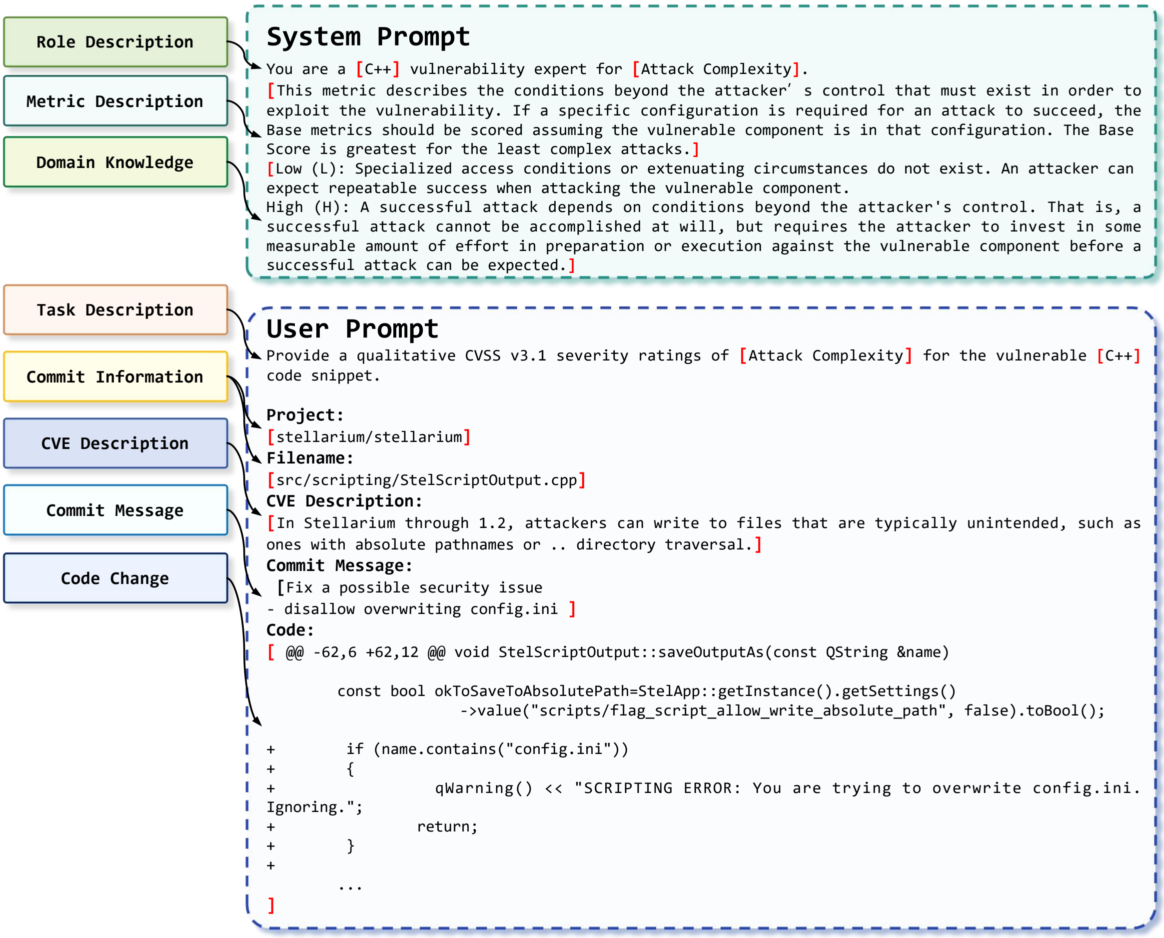


Figure 5: The prompt template for commit-based SV assessment.

Algorithm 3 Simultaneous Assessment

Require: Agents Number: N, Expert Agents: $E_1, E_2, \cdots, E_N$, Communication Rounds: R, Chat History List $History$, Vulnerability Assessment Tasks $T_1, T_2, \cdots, T_8$

Ensure: Results for Vulnerability Assessment Task $Answer$

Initialize a vulnerability assessment task T_i
for $r \leftarrow 0$ to R **do**
  Initialize current round chat history $History_c$
  for $n \leftarrow 0$ to N **do**
    if $History \neq \emptyset$ **then**
      $h_{role} \leftarrow E_n$
      $h_{answer} \leftarrow E_n(T_i, History)$
      $History_c \leftarrow \{h_{role}, h_{answer}\}$
    else
      $h_{role} \leftarrow E_n$
      $h_{answer} \leftarrow E_n(T_i)$
      $History_c \leftarrow \{h_{role}, h_{answer}\}$
    end if
  end for
  $History \leftarrow History + History_c$
  $Answer \leftarrow$ Final h_{answer}
end for
return $Answer$

Algorithm 4 Summarizer Assessment

Require: Agents Number: N, Expert Agents: $E_1, E_2, \cdots, E_{N-1}$, Summarizer Agent S, Communication Rounds: R, Chat History List $History$, Vulnerability Assessment Tasks $T_1, T_2, \cdots, T_8$

Ensure: Results for Vulnerability Assessment Task $Answer$

Initialize a vulnerability assessment task T_i
for $r \leftarrow 0$ to R **do**
  Initialize current round chat history $History_c$
  for $n \leftarrow 0$ to N **do**
    if $n \neq N$ **then**
      $h_{role} \leftarrow E_n$
      $h_{answer} \leftarrow E_n(T_i, History)$
      $History_c \leftarrow \{h_{role}, h_{answer}\}$
    else
      $s_{role} \leftarrow S$
      $s_{answer} \leftarrow S(T_i, History + History_c)$
      $History_c \leftarrow \{s_{role}, s_{answer}\}$
    end if
  end for
  $History \leftarrow History + History_c$
  $Answer \leftarrow$ Final s_{answer}
end for
return $Answer$

Exploitability Metrics	**AV**		**AC**		**PR**		**UI**	
Communication Strategy	**Acc**	**F1**	**Acc**	**F1**	**Acc**	**F1**	**Acc**	**F1**
Single Agent	0.4778	0.4075	0.4000	0.3132	0.2889	0.2425	0.3667	0.3532
Previous Communication	0.3222	0.3078	0.4444	0.2167	0.5778	0.3603	0.5000	0.4994
Preceding One Expert	0.3111	0.3056	0.3556	0.1943	0.5556	0.3492	0.4667	0.4643
Simultaneous Assessment	0.4444	0.3939	0.4111	0.2158	0.5778	0.3613	0.4222	0.4219
Summarizer Assessment	0.4889	0.4564	0.3222	0.1728	0.5333	0.3387	0.4000	0.4000

Scope and Impact Metrics	**S**		**C**		**I**		**A**	
Communication Strategy	**Acc**	**F1**	**Acc**	**F1**	**Acc**	**F1**	**Acc**	**F1**
Single Agent	0.1444	0.1392	0.5111	0.2591	0.4556	0.2406	0.4556	0.2427
Previous Communication	0.3333	0.3168	0.5333	0.3457	0.5000	0.2496	0.4556	0.2169
Preceding One Expert	0.3778	0.3623	0.5333	0.3572	0.4889	0.2729	0.4222	0.2189
Simultaneous Assessment	0.3444	0.3202	0.5000	0.2582	0.5111	0.2495	0.4667	0.2390
Summarizer Assessment	0.4000	0.3619	0.5333	0.2819	0.4889	0.2731	0.4667	0.2317

Table 5: Evaluation of different communication strategies of ChatGPT in Java.

Exploitability Metrics	**AV**		**AC**		**PR**		**UI**	
Communication Strategy	**Acc**	**F1**	**Acc**	**F1**	**Acc**	**F1**	**Acc**	**F1**
Single Agent	0.3333	0.3088	0.2754	0.1873	0.1449	0.0921	0.5072	0.4569
Previous Communication	0.2319	0.2214	0.3333	0.1950	0.6087	0.2523	0.5507	0.5473
Preceding One Expert	0.4203	0.3611	0.4203	0.2372	0.5942	0.2485	0.5652	0.5629
Simultaneous Assessment	0.1884	0.1869	0.3913	0.2129	0.6812	0.2701	0.5072	0.5063
Summarizer Assessment	0.2899	0.2746	0.4928	0.2686	0.6957	0.3058	0.5072	0.5071

Scope and Impact Metrics	**S**		**C**		**I**		**A**	
Communication Strategy	**Acc**	**F1**	**Acc**	**F1**	**Acc**	**F1**	**Acc**	**F1**
Single Agent	0.1449	0.1384	0.4638	0.2622	0.4928	0.3192	0.5072	0.2509
Previous Communication	0.4203	0.3155	0.4928	0.3074	0.4348	0.2259	0.6812	0.4059
Preceding One Expert	0.4928	0.3538	0.5072	0.3616	0.4203	0.2920	0.5942	0.3459
Simultaneous Assessment	0.5217	0.3907	0.5507	0.4029	0.4783	0.3024	0.6522	0.3548
Summarizer Assessment	0.4928	0.3538	0.4493	0.2313	0.5217	0.3436	0.6232	0.3888

Table 6: Evaluation of different communication strategies of ChatGPT in C++.

E Task-related Prompt

In this appendix, we present the task-related prompt by CVSS v3.1 and design several descriptions as follows.

Attack Vector: *You are a [Language] expert for Attack Vector. This metric reflects the context in which vulnerability exploitation is possible. This metric value (and consequently the Base Score) will be larger the more remote (logically, and physically) an attacker can be to exploit the vulnerable component. Network (N): The vulnerable component is bound to the network stack and the set of possible attackers extends beyond the other options listed below, up to and including the entire Internet. Local (L): The vulnerable component is not bound to the network stack and the attacker's path is via read/write/execute capabilities.*

Attack Complexity: *You are a [Language] expert for Attack Complexity. This metric describes the conditions beyond the attacker's control that must exist in order to exploit the vulnerability. If a specific configuration is required for an attack to succeed, the Base metrics should be scored assuming the vulnerable component is in that configuration. The Base Score is greatest for the least complex attacks. Low (L): Specialized access conditions or extenuating circumstances do not exist. An attacker can expect repeatable success when attacking the vulnerable component. High (H): A successful attack depends on conditions beyond the attacker's control. That is, a successful attack cannot be accomplished at will, but requires the attacker to invest in some measurable amount of effort in preparation or execution against the vulnerable component before a successful attack can be expected.*

Privileges Required: *You are a [Language] expert for Privileges Required. This metric describes* *the level of privileges an attacker must possess before successfully exploiting the vulnerability. The Base Score is greatest if no privileges are required. None (N): The attacker is unauthorized prior to attack, and therefore does not require any access to settings or files of the vulnerable system to carry out an attack. Low (L): The attacker requires privileges that provide basic user capabilities that could normally affect only settings and files owned by a user. High (H): The attacker requires privileges that provide significant (e.g., administrative) control over the vulnerable component allowing access to component-wide settings and files.*

User Interaction: *You are a [Language] expert for User Interaction. This metric captures the requirement for a human user, other than the attacker, to participate in the successful compromise of the vulnerable component. This metric determines whether the vulnerability can be exploited solely at the will of the attacker, or whether a separate user (or user-initiated process) must participate in some manner. The Base Score is greatest when no user interaction is required. None (N): The vulnerable system can be exploited without interaction from any user. Required (R): Successful exploitation of this vulnerability requires a user to take some action before the vulnerability can be exploited.*

Scope: *You are a [Language] expert for Scope. The Scope metric captures whether a vulnerability in one vulnerable component impacts resources in components beyond its security scope. The Base Score is greatest when a scope change occurs. Unchanged (U): An exploited vulnerability can only affect resources managed by the same security authority. Changed (C): An exploited vulnerability can affect resources beyond the security scope managed by the security authority of the vulnerable component.*

Confidentiality: *You are a [Language] expert for Confidentiality. This metric measures the impact to the confidentiality of the information resources managed by a software component due to a successfully exploited vulnerability. The impact can vary from none to complete disclosure of all restricted information to the attacker. High (H): There is a total loss of confidentiality, resulting in all resources within the impacted component being divulged to the attacker. Alternatively, access to only some restricted information is obtained, but the disclosed information presents a direct, serious impact. Low (L): There is some loss of confidentiality. Access to some restricted information is obtained, but the attacker does not have control over what information is obtained, or the amount or kind of loss is limited. None (N): There is no loss of confidentiality within the impacted component.*

Integrity: *You are a [Language] expert for Integrity. This metric measures the impact to the integrity of a successfully exploited vulnerability. Integrity refers to the trustworthiness and veracity of information. The Base Score is greatest when the consequence to the impacted component is highest. High (H): There is a total loss of integrity, or a complete loss of protection. For example, the attacker is able to modify any/all files protected by the impacted component. Low (L): Modification of data is possible, but the attacker does not have control over the consequence of a modification, or the amount of modification is limited. None (N): There is no loss of integrity within the impacted component.*

Availability: *You are a [Language] expert for Availability. This metric measures the impact to the availability of the impacted component resulting from a successfully exploited vulnerability. The Base Score is greatest when the consequence to the impacted component is highest. High (H): There is a total loss of availability, resulting in the attacker being able to fully deny access to resources in the impacted component; this loss is either sustained (while the attacker continues to deliver the attack) or persistent (the condition persists even after the attack has completed). Low (L): Performance is reduced or there are interruptions in resource availability. Even if repeated exploitation of the vulnerability is possible, the attacker does not have the ability to completely deny service to legitimate users. None (N): There is no impact to availability within the impacted component.*