Contents lists available at ScienceDirect

Computers & Security

journal homepage: www.elsevier.com/locate/cose

Pwnable-Sherpa: An interactive coaching system with a case study of pwnable challenges

Sung-Kyung Kim[a], Eun-Tae Jang[a], Hanjin Park[b,*], Ki-Woong Park[c,**]

[a] SysCore Lab, Sejong University, Seoul 05006, South Korea
[b] The Affiliated Institute of ETRI, Daejeon, South Korea
[c] Department of Information Security, and Convergence Engineering for Intelligent Drone, Sejong University, Seoul 05006, South Korea

ARTICLE INFO

Article history:
Received 27 November 2021
Revised 1 November 2022
Accepted 7 November 2022
Available online 15 November 2022

Keywords:
Architectures for educational technology system
Evaluation methodologies
Improving classroom teaching
Simulation
Teaching/learning strategies

ABSTRACT

To improve their cybersecurity knowledge and skills, students participate in a competitive game called capture the flag (CTF). CTF is used as an educational tool to improve students' cybersecurity competency by solving challenges. As system vulnerabilities are the leading cause of cyberattacks on critical systems, cybersecurity personnel with knowledge and skills to detect system vulnerabilities are increasingly in demand. In this context, the importance of challenges concerning system vulnerabilities, such as pwnable, is gradually increasing in CTF competitions. Unlike other CTF challenges, solving a pwnable challenge requires considerable knowledge and skill. However, traditional evaluation methods in CTF (i.e., pass or non-pass) provide limited feedback regarding knowledge and skill gaps. To investigate this issue, we analyzed the results of the CTF competitions held by our research team over the past three years (2017, 2018, and 2020). Our analysis revealed the necessity for a new evaluation system that can provide detailed feedback to students, while reducing the grading burden on educators. Thus, to provide detailed feedback, we propose a cybersecurity training platform, Pwnable-Sherpa, which sets three detailed evaluation points for a given pwnable challenge. In addition, we designed our training platform with a multi-container architecture and an LLVM dummy pass, thereby saving time by grading each detailed assessment simultaneously rather than sequentially.

1. Introduction

Students passionate about cybersecurity participate in a competitive game called capture the flag (CTF), which helps improve their knowledge and skills. CTF is a game in which two or more teams have flags in each other's systems, and each team must steal the flags of the opposing team while safeguarding their own flags. Alternatively, flags are located in the system operated by the competition management team, and each participant attempts to steal these flags. To obtain flags from the system, the participants must possess considerable cybersecurity knowledge and skills. Several studies have suggested introducing CTF for cybersecurity education (Chothia and Novakovic, 2015; Rege, 2015; Vykopal and Barták, 2016). CTFs are increasingly used in cybersecurity education because educators can measure students' cybersecurity competency based on their challenge solving performance (Chapman et al., 2014; Chung, 2017). CTFs such as DEF CON CTF (DEF CON CTF), HITCON CTF (Hitcon CTF), and Codegate CTF (CODEGATE CTF) bridge the gap between technical demand and learner competency by reflecting the technical demand for cybersecurity in the real world.

In recent years, the main cause of cyber-attacks on critical systems, such as oil pipelines (US-CERT Alert) and other operational technologies (OT) (Forescout Research Labs and JFrog Security Research), has been system vulnerabilities. From this perspective, cybersecurity personnel with the knowledge and skills to detect system vulnerabilities are increasingly in demand.[1]

Hence, system vulnerability challenges, such as pwnable, are becoming increasingly important. The term "pwn" originated from the mistyping of the word "own" on a keyboard and is used by

* Corresponding author.
** Principal corresponding author.
E-mail addresses: jotun9935@gmail.com (S.-K. Kim), euntaejang@gmail.com (E.-T. Jang), hjpark001@nsr.re.kr (H. Park), woongbak@sejong.ac.kr (K.-W. Park).

[1] In this study, we consider the Pwnable-Sherpa training platform that can improve the skills required to detect a program's vulnerability and write an exploit. Studying a training platform that can improve patch skills by testing students' submitted patched programs is interesting, because people who possess patching skills are increasingly in demand. We leave this concept to future studies.

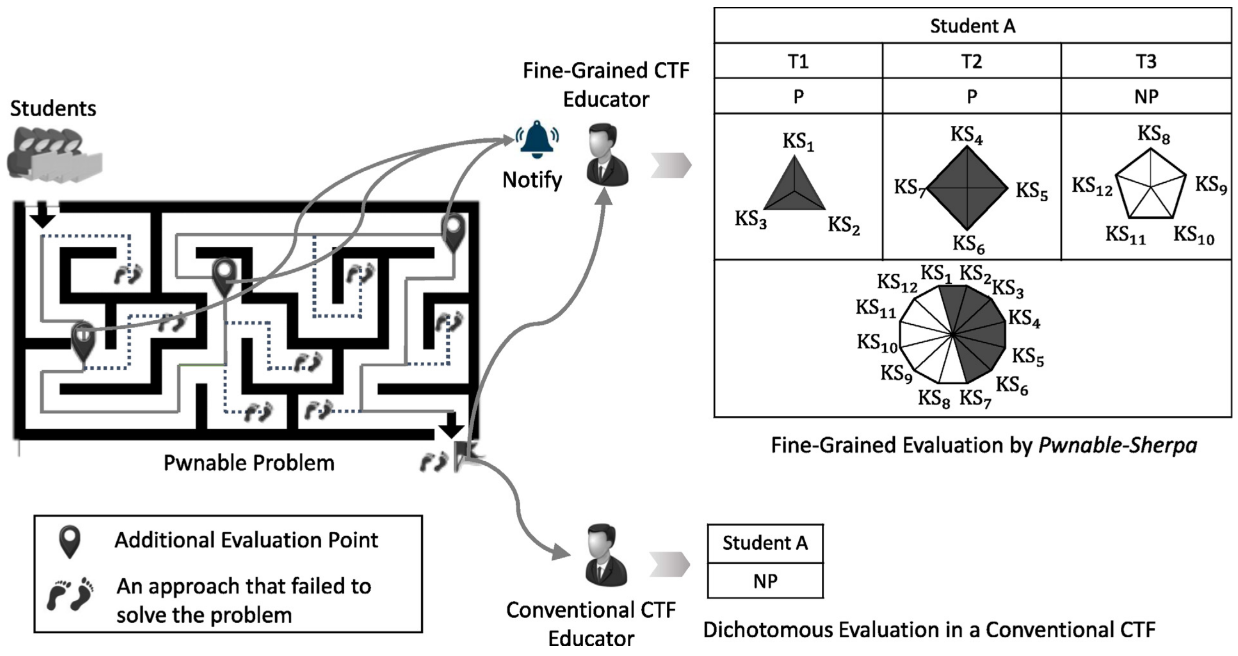

Students
Fine-Grained CTF Educator
Notify
Student A
T1
T2
T3
P
P
NP
KS1
KS2
KS3
KS4
KS5
KS6
KS7
KS8
KS9
KS10
KS11
KS12
Fine-Grained Evaluation by Pwnable-Sherpa
Pwnable Problem
Additional Evaluation Point
An approach that failed to solve the problem
Conventional CTF Educator
Student A
NP
Dichotomous Evaluation in a Conventional CTF

Fig. 1. Comparison of the two assessment methods: conventional CTF (for competition) and fine-grained CTF (for cybersecurity education).

the hacker community to obtain unauthorized control over someone's system. A pwnable challenge is testing whether a participant can read a flag (string) from the target system by exploiting vulnerabilities. After successfully acquiring unauthorized control, the participant can obtain the flag and authenticate it using a scoring server. A pwnable challenge consists of the binary of an application, network access (e.g., IP address and port number) for a target system where the binary runs as a service, and a challenge description. To solve this pwnable challenge, reverse engineering, the exploit writing, operating systems, software vulnerability, and network programming knowledge and skills are required.

However, because a pwnable challenge requires a deep understanding of the system (e.g., operating system, application, and instruction sets), many difficulties limit educational effectiveness, unlike other CTF challenges. To examine these difficulties, we analyzed the results of CTF competitions held by our research team over the past three years (2017 and 2018 Park et al., 2018), and 2020 CTFs for 87 high school students). We identified a significant variation in scores among participants using the Kolmogorov-Smirnov normality test (Kolmogorov-Smirnov), indicating that a small number of students could solve the problem correctly.

In addition, the existing CTF evaluation method (pass or non-pass) is more applicable to competition ranking than to improving student competency. Therefore, it cannot provide detailed feedback to students, limiting the enhancement of their knowledge and skill (KS). However, conducting an evaluation to provide detailed feedback to a class (e.g., formative assessment) involves providing individual feedback to students, which demands significant labor and time from educators (Chothia and Novakovic, 2015).

Thus, we propose a cybersecurity training platform called Pwnable-Sherpa, which sets three detailed evaluation points for a pwnable challenge that provides detailed feedback by referring to the control flow hijacking attack stage (Szekeres et al., 2013). For each evaluation point, the following tasks (Ts) need to be performed to solve the challenge: (T1) identify vulnerabilities in the application (e.g., memory corruption bugs) using program analysis and then trigger them, (T2) hijack the control flow of the application using the vulnerabilities, and (T3) bypass mitigations (i.e., protection techniques in the system, such as stack canary (Bufferoverflow), data execution protection (DEP) (Executable space protection), no executable (NX) (Designer, 1997; van der Veen et al., 2012; van de Ven, 2004), and address space layer randomization (ASLR) (Shacham et al., 2004, Theo de Raadt, Chew and Song, 2002, Executable space protection)).

Let us now present an example to better illustrate our approach. Suppose a student successfully performs tasks (T1, T2) but can not perform T3 because the student does not have the KS required for T3. Under conventional CTF evaluation (pass or non-pass), the student would not obtain a score. In contrast, if a new formative assessment method could provide scores for T1 and T2 alongside feedback (e.g., the KS required to perform T3), it would help students improve their learning. Through such a detailed assessment, the educator can measure a student's capability using evaluation points, as shown in Fig. 1 (e.g., radial shape chart).

To realize this educational evaluation philosophy as an automated and practical system, we designed our training platform with a multi-container architecture, thereby saving time by performing these evaluations simultaneously, rather than sequentially, for grading each assessment in an isolated manner, as shown in Fig. 2. The system architecture of Pwnable-Sherpa can simultaneously perform detailed evaluations from the exploit code submitted by the student (see Section 4.2). For instance, Pwnable-Sherpa may consider the memory offset between binary files in which two binaries are applied to different mitigation techniques. For example, for binary layouts with and without a stack canary, the two binary layouts are different. These technologies enable educators to provide students with more detailed feedback while saving time and labor.

The remainder of this paper is organized as follows. In Section 2, we analyze the compositions of pwnable challenges in major CTF competitions and explain the details of control flow hijacking attacks. Section 3 describes a standard workflow for solving the pwnable challenge of a control flow hijacking attack and the evaluation points for student achievement for each task. Section 4 presents the design and implementation of the Pwnable-Sherpa. Section 5 highlights the limitations of the conventional CTF evaluation method using the experimental results. Finally, Section 6 concludes the paper.

This part of our paper was presented at the WISA 2020 conference and Information Security Application Lecture Notes in Com-

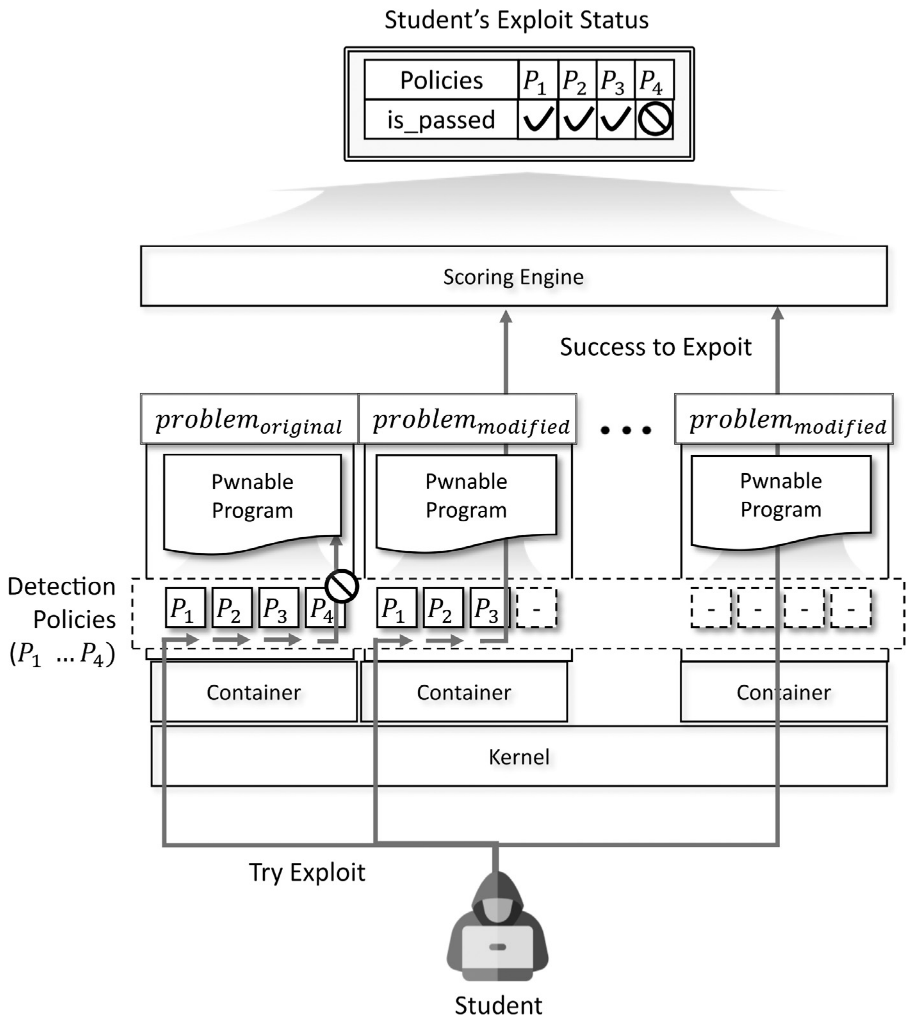


Fig. 2. Detailed evaluation method of a pwnable challenge.

puter Science (LNCS) (Kim et al., 2020), which includes the initial architecture of Pwnable Sherpa. We have made several valuable additions to improve our LNCS paper as follows: (i) Analyzing the pwnable challenge-solving workflow using the Task, Knowledge, and Skill (TKS) framework, which is a standard framework for training implementation; (ii) dealing with post-implementation, such as integrating Pwnable-Sherpa with a course and measuring student improvement based on detailed feedback; and (iii) analyzing the 2020 pwnable CTF competition results held by our research team and demonstrating the problem with the existing CTF evaluation method; and (iv) providing sufficient evidence for why we must address the control flow hijacking pwnable challenge.

2. Background

2.1. Pwnable challenge

In a pwnable challenge, participants are provided with the binary of an application and network access information (e.g., IP address and port number) for a remote server in which the binary is running. If the participants read a flag (i.e., a unique string) on the server by exploiting the vulnerability of the application and submitting the flag to the scoring server, they obtained a score. A pwnable challenge typically contains one of the following vulnerabilities: memory safety violation bugs (Erlingsson et al., 2010; van der Veen et al., 2012), logic errors (Kim et al., 2019, Zalewski), side-channel attacks (Seibert et al., 2014), or encryption misuses (Microsoft, 2020). A memory safety violation, in which an attacker manipulates the memory segment in a program written in low-level languages such as C or C++, is the most frequently tested type of pwnable challenge (Burns et al., 2017). Moreover, this type of vulnerability consistently occurs in low-level system software, such as operating system kernels, run-time libraries, and browsers, and is a highly ranked vulnerability among MITRE's top 25 most dangerous security vulnerabilities (MITRE, 2020).

Figure 3 shows the results of analyzing the vulnerability types of 105 pwnable challenges tested in four major CTF competitions over the past three years: Plaid CTF (Plaid CTF), 0 CTF (0 CTF), DEFCON CTF (DEF CON CTF), and HITCON CTF (Hitcon CTF). The analysis results revealed that, on average, approximately 92% of the pwnable challenges required control flow hijacking using memory safety violations. Based on the statistics of control flow hijacking type pwnable challenges, this study evaluates the control flow hijacking (CFH) pwnable challenge using a memory safety violation bug. Hereafter, we use the terms "pwnable challenge" and "CFH pwnable challenge" interchangeably.

2.2. Control flow hijacking

CFH is a typical tactic employed by programs that violate memory safety because software written in a low-level language, such as C and C++, is susceptible to memory safety viola-

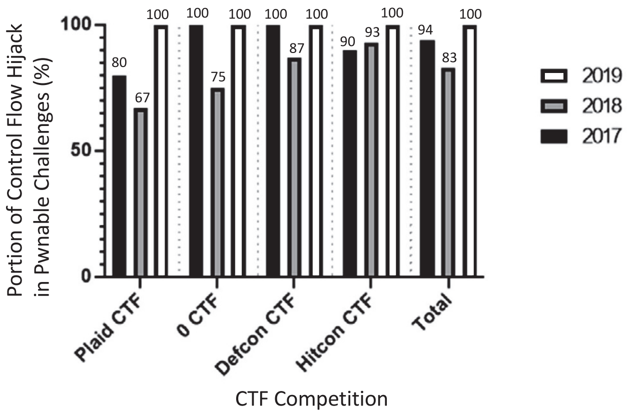


Fig. 3. Proportions of the control flow hijacking in pwnable challenges.

tions, unlike those written in higher-level languages, such as Java (Dhurjati et al., 2003) and Rust (Matsakis and Klock, 2014), which support memory safety at the programming language level. Lack of memory safety can lead to serious security threats. An unauthorized person can seize the targeted program or system by manipulating the program memory, thereby operating the program with a purpose not originally designed by the developer, or even hijacking the program's control flow.

The initial step of CFH is to induce memory errors in vulnerable programs. In this step, the participant can violate the spatial or temporal safety of memory by abusing various bugs in the program. Spatial safety is violated when a participant accesses the memory in an unintended manner, such as referencing outside a pointer's boundary or an object (e.g., buffer overflow or underflow may occur). Temporal safety is violated when a program accesses the memory using invalid references. In general, if memory allocation is explicitly released using the free function and deletion operator, all pointers pointing to the respective memory are dangling. If dangling pointers are not successfully freed, a participant can access the memory released using a dangling pointer, which is called use-after-free vulnerability. In the subsequent steps, free data are manipulated using the previously induced memory error. If the data that a participant can control are related to program control, the participant can hijack the program's control flow by manipulating the program's code segment or memory space. For example, if a participant can manipulate a memory segment correspondings to the return address of a function or pointer of a virtual table, the control flow of the program changes when the return command or indirect call/jmp of the program is executed. Furthermore, unintended code can be executed (e.g., executing shell code, such as /bin/sh, by controlling the program execution flow). A participant requires information related to the data memory layout shown in Fig. 4 to trigger a vulnerability that can be exploited.

Meanwhile, if a protection technique (i.e., mitigation) is present in the system or program, the participant must bypass the protection method to hijack the control flow of the application. Mitigation policies, such as W⊕X (write XOR execute), non-executable (NX) (Designer, 1997; van de Ven, 2004), and address space layout randomization (ASLR) (Shacham et al., 2004, Theo de Raadt, Chew and Song, 2002) of the latest OS, and the stack cookie deployed in a compiler can increase the difficulty level of exploiting a memory corruption bug.

However, several mitigation bypass tactics exist, such as return-oriented programming (ROP) (Return Oriented Programing), jump-oriented programming (Bletsch et al., 2011; Nergal, 2001), information leakage (Serna, 2012; Strackx et al., 2009), and global offset table (GOT) overwrite (c0ntex). In particular, an ROP exploit can bypass various mitigation policies while enabling a participant to achieve Turing completeness without inserting a new code, such as shellcode, into the program. ROP consists of a chain of identified command sequences within the program code called gadgets, in which an exploit is performed by changing the control flow in the code segment while maintaining memory execution rights. The participant requests the address of a fixed gadget to constitute the ROP chain, as shown in Fig. 4.

3. Pwnable challenge-solving tasks and their evaluation

In this section, a challenge-solving workflow is deduced for students participating in a CFH pwnable challenge. The KS required for students to perform each task in the workflow was analyzed. Subsequently, the evaluation points for detailed feedback were deduced based on the derived challenge-solving workflow.

3.1. Tasks for solving a pwnable challenge

A CFH pwnable challenge generally involves capturing the flag file in a system by exploiting the vulnerability of a program operating on the system. In this study, the challenge-solving process for a pwnable challenge was considered to have three tasks: (1) identifying and triggering vulnerabilities, (2) hijacking the control flow of the program, and (3) bypass mitigation, as shown in Fig. 5.

To solve this type of a pwnable challenge, a participant must first identify the vulnerability of the given application through program analysis. The participant then builds an exploit code to capture the control flow of the application by exploiting the vulnerability. If mitigation is present in the environment, the participant must write an exploit code to bypass identified mitigation policies. The details of each task are as follows:

Task 1 (T1). Identify and Trigger the Vulnerabilities: To solve a pwnable challenge, a student (i.e., learner) must first identify the vulnerability within the program by analyzing the given application and then triggering the vulnerability. Examples of KSs required for this task are as follows:

- Knowledge: knowledge of various types of architectures, such as x86-64, ARM, and MIPS, depending on the system environment in which the program operates and the vulnerabilities within the application.
- Skills: static and dynamic analysis skills for identifying the vulnerability of an application, skill to analyze the source code, reverse engineering skill, and programming skills to write an exploit payload to trigger the vulnerabilities of the application.

Task 2 (T2). Hijack the Control Flow of the Program: Most pwnable challenges aim to capture the control flow of a program. A participant captures the control flow by manipulating the data related to the indirect call of a program located within the virtual memory, while maintaining write permissions within the program. Examples of KS required for this task are as follows:

- Knowledge: knowledge of the memory structure, such as the return address of a function, function pointers, v-tables, and global offset tables (GOTs), which are the data used for indirect calls of a program.
- Skills: shell coding skills that capture the control of a program by capturing the control flow by interfering with the indirect call of the program.

Task 3 (T3). Bypass Mitigation: Most current operating systems are subjected to various mitigation techniques that minimize the damage caused by security vulnerabilities. Accordingly, certain challenges in CTF competitions require participants to capture the control of a program while bypassing several protection techniques

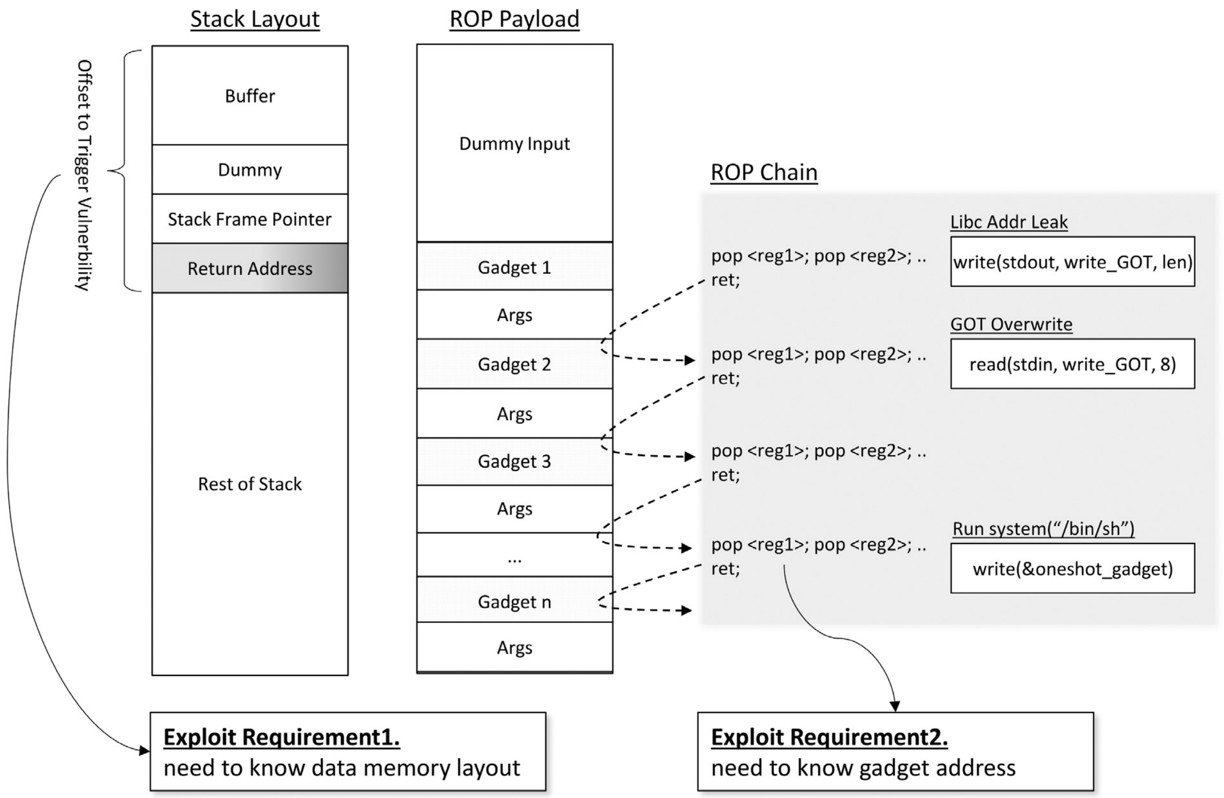


Fig. 4. Control flow hijacking with W⊕X and ASLR mitigation.

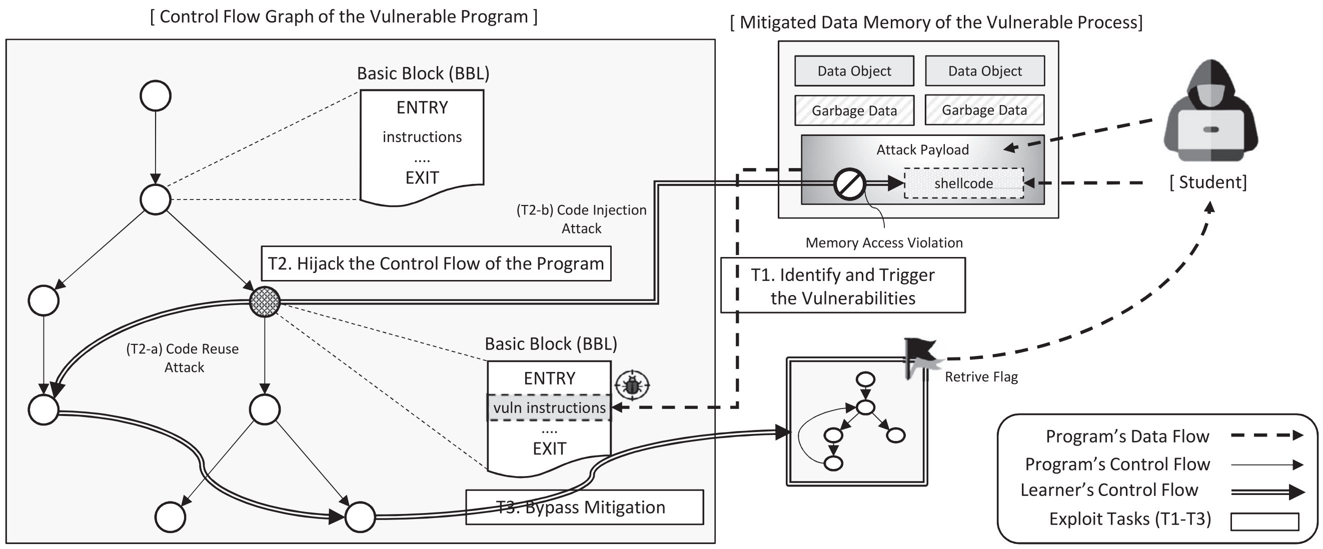


Fig. 5. Challenge-solving workflow for pwnable challenges.

(i.e., mitigation). Examples of KS required for this task are as follows:

- Knowledge: knowledge of the mitigation techniques present in the system or program, such as W⊕X, DEP, NX, and ASLR.
- Skills: programming skills to write exploit code for bypassing the mitigations.

3.2. Evaluation points: How to evaluate whether a learner can perform a task

If an educator can measure students' achievements in a pwnable challenge in a task unit, students' failure can be precisely identified by examining whether the students completed specific tasks. Pwnable-Sherpa has three possible cases in which a student can earn points while solving a challenge: (i) students successfully performed only task 1, (ii) students successfully performed tasks 1 and 2, and (iii) students successfully performed tasks 1, 2, and 3. Hence, we chose detailed evaluation points to judge whether the student has completed specific tasks, as shown in Fig. 6. In addition, the detection policies for each evaluation point were configured based on the workflow to solve a pwnable challenge.

Evaluation Point 1 (EP1): Memory Access Violation Check: This evaluation point assesses whether a student can analyze the

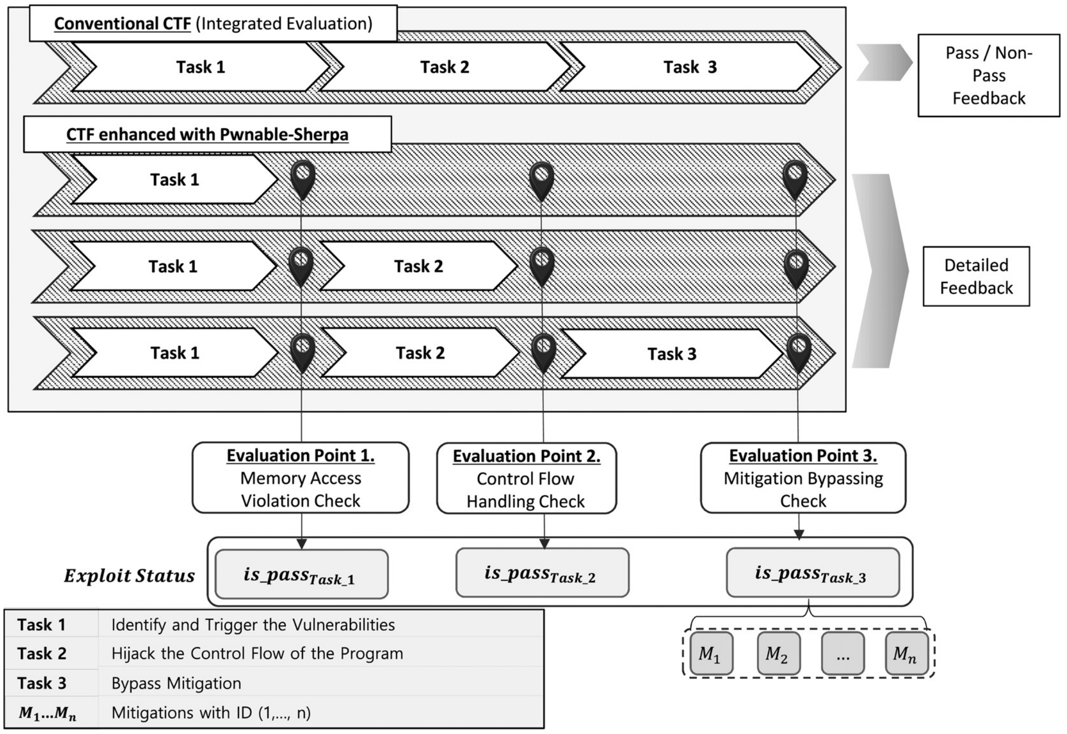

Conventional CTF (Integrated Evaluation)
Task 1
Task 2
Task 3
Pass / Non-Pass Feedback
CTF enhanced with Pwnable-Sherpa
Detailed Feedback
Evaluation Point 1. Memory Access Violation Check
Evaluation Point 2. Control Flow Handling Check
Evaluation Point 3. Mitigation Bypassing Check
Exploit Status
$is\_pass_{Task\_1}$
$is\_pass_{Task\_2}$
$is\_pass_{Task\_3}$
M_1 M_2 … M_n
Task 1 Identify and Trigger the Vulnerabilities
Task 2 Hijack the Control Flow of the Program
Task 3 Bypass Mitigation
$M_1 … M_n$ Mitigations with ID (1,…, n)

Fig. 6. Evaluation points for measuring achievements in a pwnable challenge-solving workflow.

vulnerable program and trigger actual vulnerabilities. To perform unintended memory access using the vulnerability of the executed program, a student must analyze the given program (i.e., binary) to identify the vulnerability and configure the program's input data to trigger the identified vulnerability. Evaluation point 1 assesses whether the exploit code of the student induces a memory access violation in the program (e.g., an error with a message such as "Cannot access memory at that address" or the "SIGFAULT" string in stderr(2)). That is, if a student can induce crashes using the program's vulnerability, the student is considered to have the ability to analyze the vulnerability.

Evaluation Point 2 (EP2): Control Flow Handling Check: This evaluation point assesses a student's ability to capture the control flow of the program. In the challenge-solving process for control flow hijacking, a student must be able to manipulate a program's control flow by exploiting the vulnerabilies identified in the program. The exploit code submitted by the student runs in an environment without mitigation, and whether the code can successfully perform CFH is assessed.

Evaluation Point 3 (EP3): Mitigation Bypassing Check: This evaluation point assesses whether a student knows the mitigations applied to the environment in which the challenge binary is running, and possesses the skills to bypass the mitigation techniques. For example, when a mitigation technique, such as a stack canary, is present in a program, a student must leak the canary data inserted in the program using information leakage for an exploit or brute-force attack. Furthermore, if ASLR or NX mitigation is present, the student must create an exploit, such as a code-reuse exploit, that can bypass the mitigation.

If the student successfully passes a specific evaluation point n, the educator can determine that the student has the KS to perform task n. However, if the student does not pass the evaluation point, the educator can provide appropriate feedback (e.g., the KS required for this task) to help the student overcome the difficulty of completing task n.

4. Design and implementation of Pwnable-Sherpa

4.1. Pwnable-Sherpa design

The proposed framework is expected to function as a CTF framework that can perform formative assessments for each student while relieving educators' burden in terms of labor and time demands. The design of Pwnable-Sherpa has the following characteristics:

Fine-grained assessment and its automation: For the formative assessment of a pwnable CTF, we subdivided the evaluation process of an original pwnable challenge into three evaluation points according to the challenge-solving workflow, as described in Section 3. Because each challenge is divided into multiple sub challenges, the number of evaluation processes increases. To save time in the detailed evaluation process, we applied a parallel architecture design for each detailed evaluation by reconfiguring the challenge, if needed, for the task, as shown in Fig. 7.

Each evaluation point was independently checked in a separate container. The original challenge can be modified for a given evaluation point (Section 4.2). When the exploit code submitted by a student is executed for the original and modified challenges in the evaluation environment, the student's exploit achievement for the problem is precisely measured by determining whether the student realized the exploit in each evaluation environment in which the detection policies were applied. **Providing detailed feedback**: After the evaluation, our framework can provide descriptive feedback to students for each task, such as a list of the required KS. The educator can edit the description of KS for each task before presenting a challenge to the students. In addition, our framework has a function through which educators can provide additional information on challenges, such as hints, to enable students to solve challenges.

Encouraging two-way communication: When a student solves a challenge, he/she can ask the educator questions using a bi-

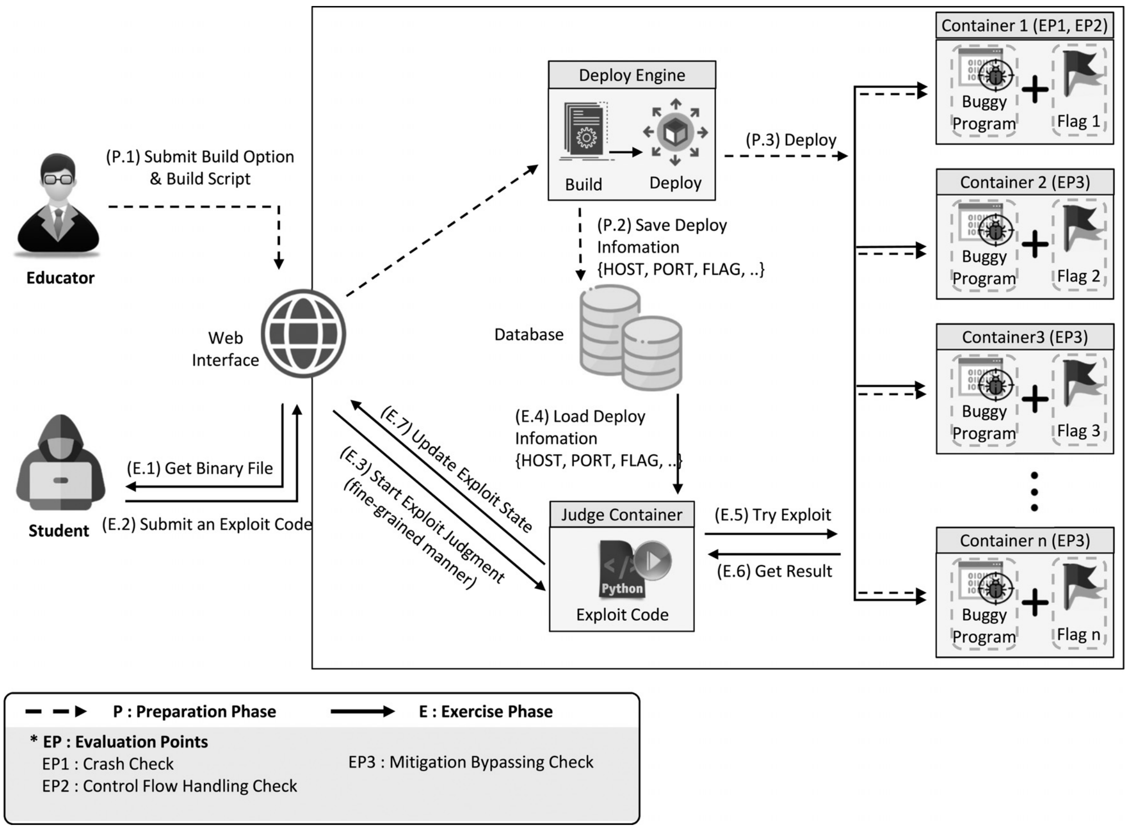


Fig. 7. Overall architecture of Pwnable-Sherpa.

directional communication channel. For the communication channel, we used the existing Discord communication platform (an instant messaging application) (Discord). Analyzing and solving challenges from various perspectives can help improve students' challenge-solving skills.

Providing detailed feedback can burden the educator because the educator may need to edit the detailed feedback for each student. The following methods can alleviate this burden:

Preparing feedback for each challenge and providing it on demand: The educator can prepare the expected feedback for each challenge (when they upload the challenge to Pwnable-Sherpa). Because determining which task the student cannot perform is easy using Pwnable-Sherpa, the KS required to solve the task, which was prepared in advance, can be retrieved from the database and immediately displayed.

Using students who have already succeeded as tutors: Giving students who have successfully solved the challenge opportunities to provide feedback to other students can help alleviate the educator's labor and time constraints.

This framework consists of a preparation phase, in which the educator deploys the challenge, and an exercise phase, in which students solve the challenge. Figure 7 shows the entire process flow, from the educator submitting a challenge to students solving the challenge (P: preparation phase, E: exercise phase).

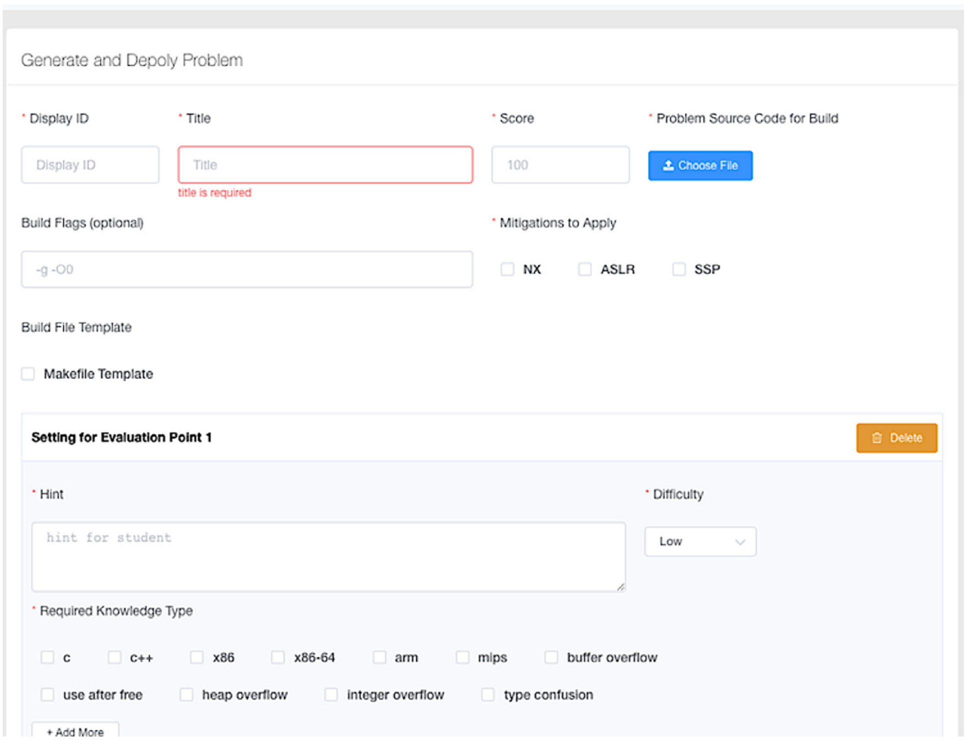
4.1.1. Preparation phase

As shown in Fig. 7, Pwnable-Sherpa consists of web-based services, where an educator submits the problem through a web interface. Subsequently, the challenge is reconfigured in a detailed evaluation environment by an automated process. First, to set a challenge, the educator submits the source code of the challenge required to configure the environment and build scripts, hints, mitigations, and feedback (e.g., required KS) through the web interface (P.1 in Figs. 7 and 8). As shown in Fig. 8, the educator can edit the list of the required KS using our framework to provide feedback to students. The build information of the challenge submitted by the educator is then delivered to the Deploy Engine. While generating the challenge program and its environment, the Deploy Engine reconfigures the challenge environment to determine the achievements of the student in each task of the challenge-solving workflow and deploys the environment in containers (P.3 in Fig. 7).

Each evaluation point is configured as a power set for all mitigation techniques applied to the problem. For example, if a mitigation technique is not applied to a specific container, the respective container corresponds to evaluation point 2 to assess a student's knowledge of control-flow capture. For evaluation point 3, the problem environment is deployed in a container in which mitigation techniques are applied in varying combinations and it is assessed whether the student has the knowledge required to bypass the mitigation techniques. Then, for the container in which the educator designated all mitigation techniques for the challenge, it is determined whether the student can fully exploit the program, which corresponds to evaluation point 4. In addition, the Deploy Engine saves the information required to identify a container corresponding to each evaluation point deployed for a detailed evaluation in the internal database, and the Deploy Engine uses the saved data when evaluating the exploit code of a student (P.2 in Fig. 7).

4.1.2. Exercise phase

After the educator submits the challenge using our framework, the student downloads the challenge program through the web interface, writes the exploit code, and submits it (E.1 in Fig. 7). The exploit code is directly submitted through the web, unlike a conventional pwnable CTF, where an exploit payload is delivered from


Generate and Depoly Problem

* Display ID: Display ID
* Title: Title — title is required
* Score: 100
* Problem Source Code for Build: Choose File

Build Flags (optional): -g -O0

* Mitigations to Apply: ☐ NX ☐ ASLR ☐ SSP

Build File Template: ☐ Makefile Template

Setting for Evaluation Point 1 — Delete

* Hint: hint for student
* Difficulty: Low
* Required Knowledge Type: ☐ c ☐ c++ ☐ x86 ☐ x86-64 ☐ arm ☐ mips ☐ buffer overflow ☐ use after free ☐ heap overflow ☐ integer overflow ☐ type confusion

+ Add More

Fig. 8. Web interface for setting a pwnable challenge.

a remote server through the program port (E.2 in Fig. 7). Subsequently, the exploit code submitted by the student is evaluated in an isolated container environment to prevent malicious behavior that may occur when the code is executed (E.3 in Fig. 7). The exploit code executed in an isolated environment is combined with the identification information of each evaluation point of the challenge saved in the database to be executed in the container corresponding to each evaluation point (E.4 and E.5 of Fig. 7). The results of executing the exploit code for each evaluation point are saved in the database, which can be viewed later by an educator or students through the web interface (Es.6 and 7 of Figs. 7 and 9). Students have multiple opportunities to submit exploits for a challenge. The number of opportunities provided to students is a policy component that educators determine when running the Pwnable-Sherpa platform. Students can obtain points for the exploit that obtains the highest score on the platform among the multiple exploits they submit.

As shown in Fig. 9, the student can determine which evaluation points were and were not solved through the web page, which can help the student concentrate on improving the required KS.

4.2. Implementation issues of Pwnable-Sherpa

Some mitigations, such as the inline reference monitor (IRM) type (e.g., stack canary), cause the program code to change because they insert the reference monitor into the program binary (for example, see the changes by comparing the codes in Fig. 10(a) and (b)). Changes in the binary file can affect the success or failure of an exploit because virtual memory address space layout of the process changes. For example, to develop an exploit that uses memory safety violations, such as buffer overflow and heap chunk attacks, a student must know the program's memory layout information. Furthermore, some exploit skills for bypassing mitigation techniques, such as ROP, using a code gadget in the memory. Therefore, students should develop an exploit according to the situation in which mitigation is applied.

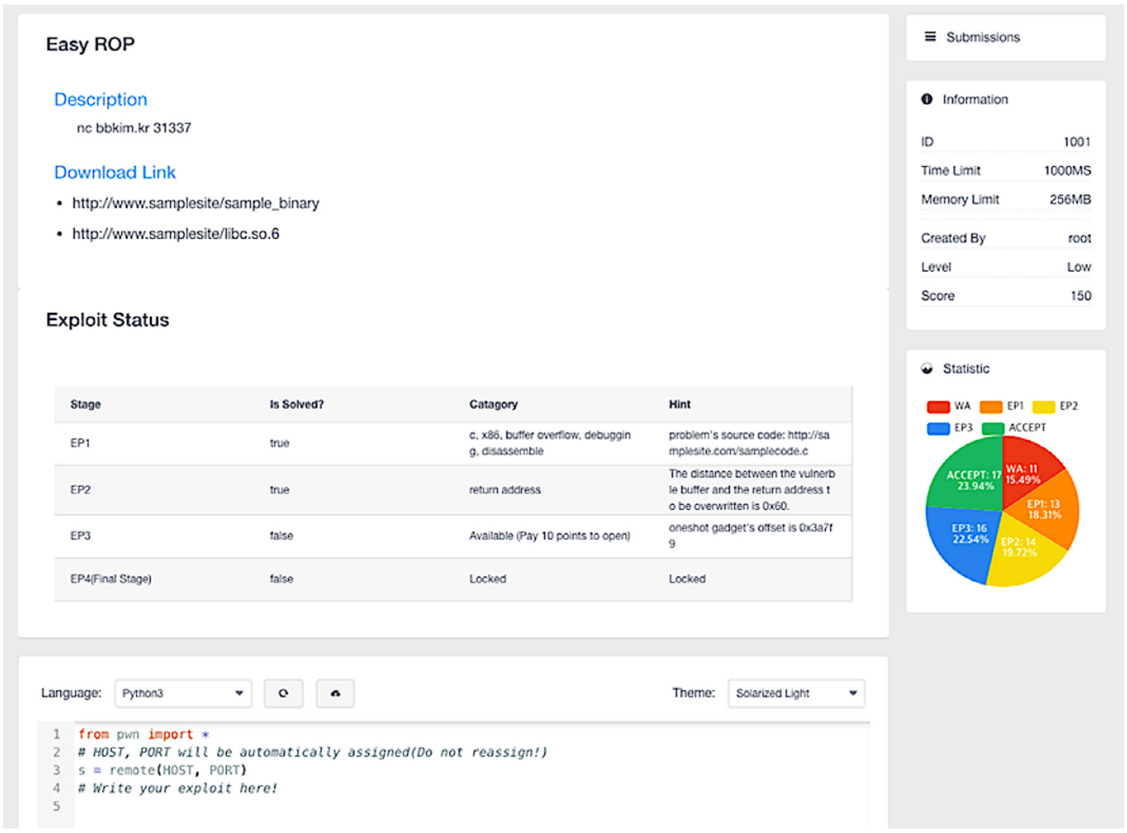
As shown in Fig. 7, in Pwnable-Sherpa, we configured different mitigation environments in several containers for detailed evaluation. However, this burdens students by requiring them to write multiple exploits for a given pwnable challenge to meet each evaluation point condition separately while considering which mitigation type was applied.

To solve this problem, we modified the LLVM-based compiler by adding a new LLVM pass (Dummy StackProtector) that maintains the code offset and virtual memory layout of the pwnable program. Because of the proposed technique, the student writes only one exploit under the assumption of one system environment (all intended mitigations are active); thus, the abovementioned burden is alleviated. Furthermore, in Pwnable-Sherpa, a detailed evaluation is possible in multiple mitigation environments (multiple containers) using the exploit simultaneously.

The LLVM compiler consists of three parts: a front-end that converts C or C++ source code into an intermediate representation (IR), a middle-end that transforms IR to IR throughout the passes, and a back-end that converts IR to machine code (LLVM Compiler Infrastructure), as shown in Fig. 11. The LLVM pass is a building block in which code optimization occurs while transforming IR into IR.

First, we wrote a new LLVM pass (refer to the details about writing an LLVM Pass in (Writing an LLVM Pass), see $Pass_{dummy}$ in Fig. 11) and a Dummy StackProtector, a code with the same instrumentation code size that does not perform the stack canary check. The code with the same instrumentation code size is designed to ensure that the same code offset and virtual memory layout are obtained, regardless of whether the Dummy StackProtector or the original StackProtector is used.

Second, we modified the abstract syntax tree (AST) generation step for the StackProtector in the LLVM code such that the LLVM IR can be generated with the specified option (see IR_{dummy} of Fig. 11). Depending on the compilation options, the platform can use either the original StackProtector or the Dummy StackProtector (by modifying the code to create a basic block to jump to when the stack


Easy ROP

Description

nc bbkim.kr 31337

Download Link

- http://www.samplesite/sample_binary
- http://www.samplesite/libc.so.6

Exploit Status

Stage	Is Solved?	Catagory	Hint
EP1	true	c, x86, buffer overflow, debuggin g, disassemble	problem's source code: http://sa mplesite.com/samplecode.c
EP2	true	return address	The distance between the vulnerb le buffer and the return address t o be overwritten is 0x60.
EP3	false	Available (Pay 10 points to open)	oneshot gadget's offset is 0x3a7f 9
EP4(Final Stage)	false	Locked	Locked

Information

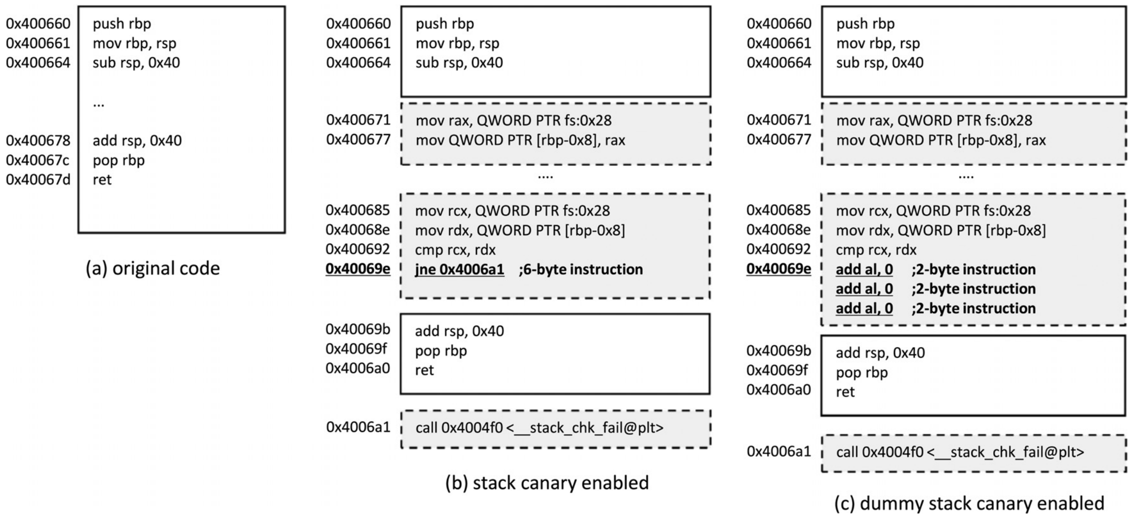
ID	1001
Time Limit	1000MS
Memory Limit	256MB
Created By	root
Level	Low
Score	150

Language: Python3 Theme: Solarized Light

```
from pwn import *
# HOST, PORT will be automatically assigned(Do not reassign!)
s = remote(HOST, PORT)
# Write your exploit here!

```

Fig. 9. Web interface for challenge solving achievements.


(a) original code

```
0x400660  push rbp
0x400661  mov rbp, rsp
0x400664  sub rsp, 0x40
          ...
0x400678  add rsp, 0x40
0x40067c  pop rbp
0x40067d  ret
```

(b) stack canary enabled

```
0x400660  push rbp
0x400661  mov rbp, rsp
0x400664  sub rsp, 0x40

0x400671  mov rax, QWORD PTR fs:0x28
0x400677  mov QWORD PTR [rbp-0x8], rax
          ....
0x400685  mov rcx, QWORD PTR fs:0x28
0x40068e  mov rdx, QWORD PTR [rbp-0x8]
0x400692  cmp rcx, rdx
0x40069e  jne 0x4006a1    ;6-byte instruction

0x40069b  add rsp, 0x40
0x40069f  pop rbp
0x4006a0  ret

0x4006a1  call 0x4004f0 <__stack_chk_fail@plt>
```

(c) dummy stack canary enabled

```
0x400660  push rbp
0x400661  mov rbp, rsp
0x400664  sub rsp, 0x40

0x400671  mov rax, QWORD PTR fs:0x28
0x400677  mov QWORD PTR [rbp-0x8], rax
          ....
0x400685  mov rcx, QWORD PTR fs:0x28
0x40068e  mov rdx, QWORD PTR [rbp-0x8]
0x400692  cmp rcx, rdx
0x40069e  add al, 0    ;2-byte instruction
          add al, 0    ;2-byte instruction
          add al, 0    ;2-byte instruction

0x40069b  add rsp, 0x40
0x40069f  pop rbp
0x4006a0  ret

0x4006a1  call 0x4004f0 <__stack_chk_fail@plt>
```

Fig. 10. Binary reconfiguration for maintaining memory layout regardless of stack canary.

protector check fails (see the details of a member function, CreateFailBB(), in (LLVM StackProtector)) in LLVM CodeGen, StackProtector can be created as a Dummy StackProtector when the option (dummy-ssp) is specified).

As shown in Fig. 10(b) and (c), the program code when the Dummy StackProtector is applied has the same code layout as when the stack canary is applied while not performing the stack canary check. In Fig. 10(b), the program goes to the __stack_chk_fail logic if the canary value changes when performing a canary check (see the cmp and jne instructions at 0x400692 and 0x40069e in Fig. 10(b)). However, in Fig. 10(c), the program does not go to the __stack_chk_fail routine because it does not encounter a jump instruction in the canary failure routine (see the cmp and consecutive add al, 0 instructions (such as a nop instruction) at the same code offset).

4.2.1. Deploy engine

The proposed LLVM dummy pass technique is applied when the deploy engine is built. This ensures that the same code offset and memory layout are maintained by reconfiguring the challenge's original binary such that each assessment in the evaluation container can be executed simultaneously with the exploit submitted by the student. This technique inserts the dummy code shown in Fig. 10(c) into the challenge binary, which has the same

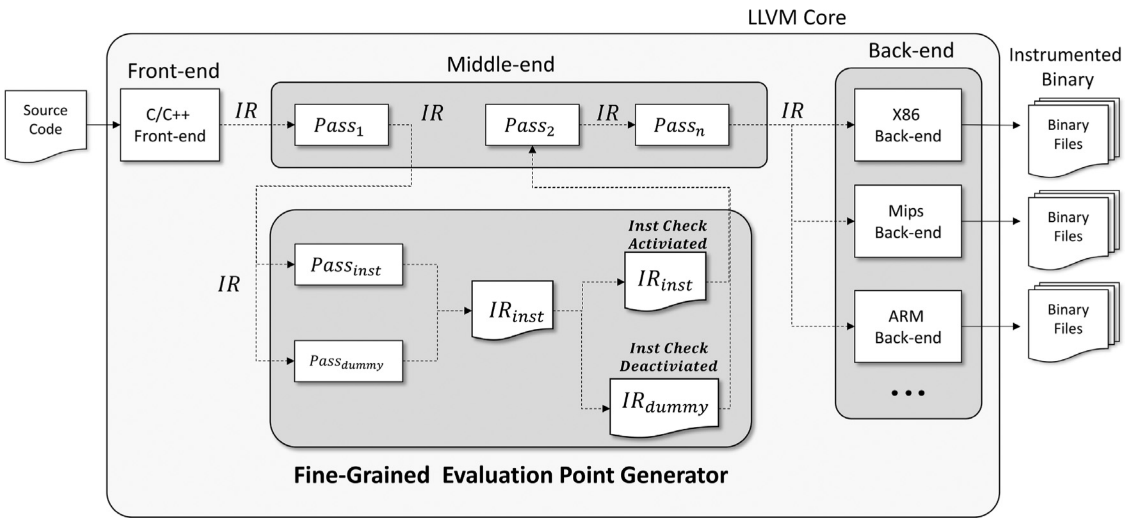


Fig. 11. LLVM dummy pass generation module design.

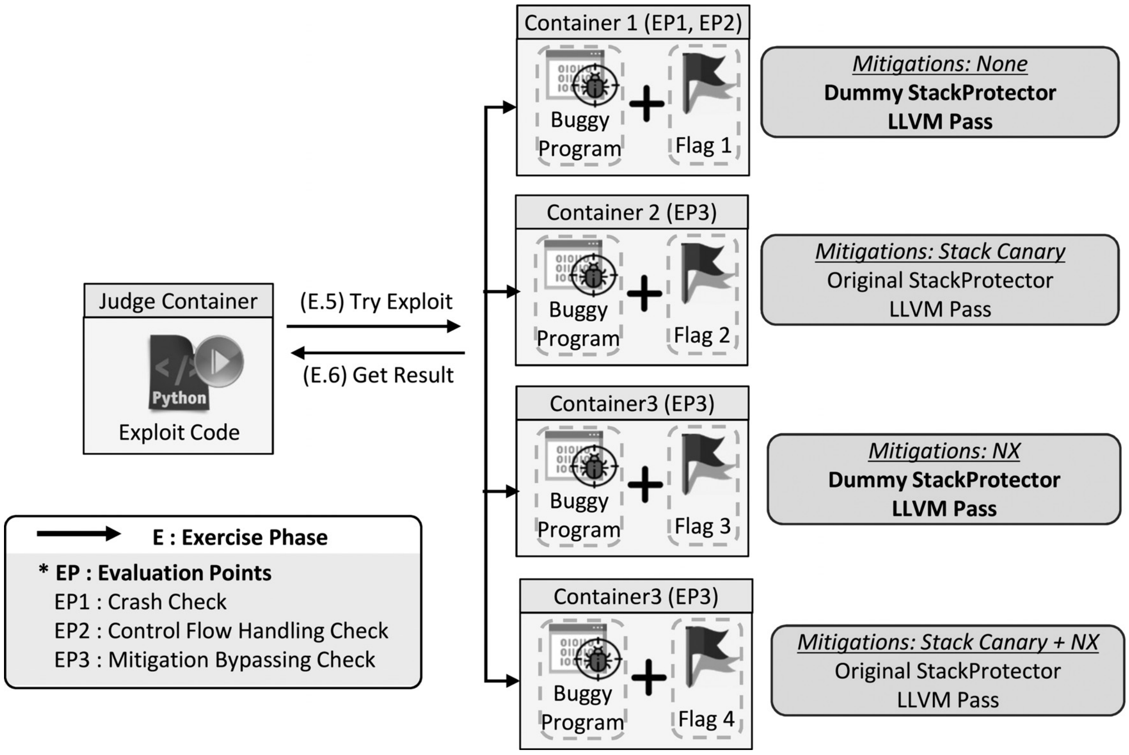


Fig. 12. Evaluation containers and LLVM dummy pass.

size as the inserted code because of the mitigation technique (Fig. 10(b)) and passes through this code area without executing it such that it has the same memory layout as when mitigation is applied.

Suppose we have a pwnable challenge that runs with two mitigations: the stack canary and NX. From the architecture of the proposed framework, the Deploy Engine builds four containers for a given example: Container 1 (EP1 and EP2: no mitigation), Container 2 (EP3: stack canary), Container 3 (EP3: NX), and Container 4 (EP3: stack canary + NX), as shown in Fig. 12. The student writes an exploit code and submits it considering Container 4's system, in which all mitigations are applied. Because stack canary mitigation changes the memory layout affecting the writing of the exploit code, the LLVM dummy pass technique is applied to the other containers in which stack canary mitigation was not applied (Containers 1 and 3) such that it could be evaluated with only one exploit (see Fig. 12).

4.2.2. Judge container

The exploit code submitted by the student undergoes a precise sequential evaluation based on each evaluation point, as shown in Algorithm 1. The judge container sends exploit code to the containers at each evaluation point. As shown in Fig. 12, evaluation points 1 and 2 (EP 1 and EP 2) can be checked simultaneously in one container (i.e., container 1) because whether a student has passed EP 1 or not is checked by determining whether a crash occurs in the system with the exploit code, which can be executed with the process of EP 2 (CFH) using the same exploit code. If the exploit code hijacks the control flow, EP 1 is considered to have been successfully passed.

Algorithm 1 Evaluation process of the judge container.

```
Input :N - challenge number (i.e, challenge identifier)
       E - exploit code submitted by the student
       D - database
       C - crash identifier ("SIGABRT", "SIGSEGV")
Output : S - exploit status
1:  P ← getPortNums (D, N) ▷ get the port numbers of evaluation containers.
2:  H ← getHost (D, N)     ▷ get the host IP address of evaluation containers.
3:  for p ← P do
4:      F ← getFlags (p)   ▷ read the flag stored in the evaluation containers.
5:      s ← tryExploit (E, H, p)   ▷ try exploit and save out stream
6:      if isContain (s, F) then
7:          S ← updateExploitState (S, D, p)   ▷ Update exploit status.
8:      end if
9:      if isContain (s, C) then
10:         S ← updateExploitState (S, D, p)   ▷ Update exploit status (crashed)
11:     end if
12: end for
```

The exploit code submitted by the student through the web interface is evaluated in a container environment that is isolated from the host environment, thereby preventing malicious behavior that may occur when the code is executed. Furthermore, to prevent cheating scenarios, the flags present in each container at different evaluation points are difficult for users to infer, and different flags should be used for each container. Therefore, in this study, randomly generated flags were used during the challenge deployment and exploit-scoring processes. Hence, the flags were different for each user and evaluation point.

4.3. Post-Implementation

In this section, we briefly examine the usage and effectiveness of Pwnable-Shera. In particular, we discuss the use of Pwnable-Sherpa in an educational course and measure the degree of improvement in student KS.

4.3.1. Integrating Pwnable-Sherpa in a course

Pwnable-Sherpa can be used as a tool in educational courses. The following briefly describes how Pwnable-Sherpa can be used in a class. First, when students register for courses, educators test them using Pwnable-Sherpa, classify them into levels, and provide guidelines for suitable classes. For example, educators can offer classes based on level, such as beginner, intermediate, and advanced courses. The students can then enroll in a class suitable for them. In addition, by testing students using Pwnable-Sherpa during a class, it is possible to know precisely which KS the students are lacking; thus, the educator can add more material for these KS, extend class and office hours to explain these KS, and schedule a make-up class for students who want to improve their KS. At the same time, students can ask as many questions as possible about KS. Thus, using Pwnable-Sherpa in a course helps educators flexibly manage classes. Meanwhile, students can efficiently improve their KS using Pwnable-Sherpa.

4.3.2. Evaluation of how Pwnable-Sherpa helps students

Examples of feedback that an educator can provide students with in Pwnable-Sherpa's challenge-solving are as follows: (i) the task(s) that the student could not perform (e.g., Student A failed to perform T3 (NP, Non-Pass), as shown in Fig. 1); (ii) the KS sets that are related to the task(s) of (i) (e.g., $KS_8, KS_9, KS_{10}, KS_{11}$, and KS_{12} are required for T3, bypass mitigation, as shown in Fig. 1); and (iii) more detailed information about each KS.

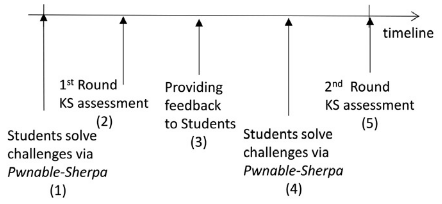
To provide more detailed feedback about KS, the educator may ask the student how much the student knows about each KS while providing the set of KS related to the task that the student could not perform. This may be related to the KS assessment of the students to provide more detailed feedback. There may be several ways to know the student's understanding of KS. Because there are various types of KS, developing separate evaluation methods for each KS is complicated and challenging. As it is somewhat beyond the scope of this paper, a technical evaluation method for assessing whether a student possesses KS will be left for future work.

However, to deal with more detailed feedback, we used the method proposed by Park and Hong (2019) in which students can perform a self-assessment for each KS. The assessment can be designed to accept values between 0 and 5 that are a measure of the student's level of understanding (0: Inexperience, 1: Having basic knowledge, 2: Beginner (limited experience, need professional help), 3: Intermediate (possible to adopt practical usage, need occasional professional help), 4: Expert (challenge solving without external help, subject to inquiries from others), and 5: Professional (certified professional)). A student can be asked to perform a KS self-assessment when the student submits exploits for a challenge or when a student submits exploits for all challenges. More fine-grained feedback can be provided to students through the KS self-assessment and the Pwnable-Sherpa (see the radial graph with dark gray color in Fig. 13(c)).

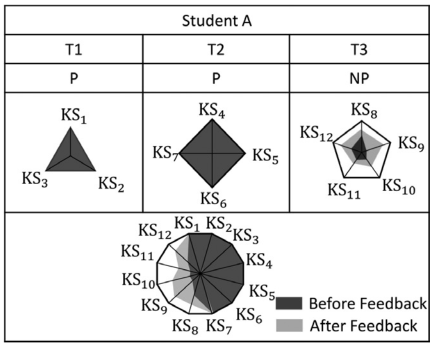
In addition, the educator can measure how much the feedback helped the student by comparing the KS self-assessment results before and after the feedback is given, as shown in Fig. 13. The detailed procedure is as follows. First, students solve challenges with Pwnable-Sherpa (Fig. 13 (a.1)); they self-assess KS, which is called the 1st round KS assessment (Fig. 13 (a.2)), which is related to the task that the students could not perform using Pwnable-Sherpa (i.e., T3, $KS_8, KS_9, KS_{10}, KS_{11}$, and KS_{12}). The results of the 1st round KS assessment of the student can be represented in radial graph form, as shown by the dark gray colored areas in Fig. 13(b) and (c). The educator then gives feedback to the students based on the results of the 1st round of KS assessment (Fig. 13 (a.3)). The students solved the challenges that were similar to those of the previous round (Fig. 13 (a.4)), and conduct a KS self-assessment (Fig. 13 (a.5)). The result of the 2nd round KS assessment of the student can be represented as a light gray colored area in Fig. 13(b) and (c). By comparing the differences between the two colored areas, the educator can measure the effects of feedback. In addition, by analyzing the feedback effect, educators may develop a better form of feedback.

4.3.3. Extension of Pwnable-Sherpa's approach to other CTF challenges

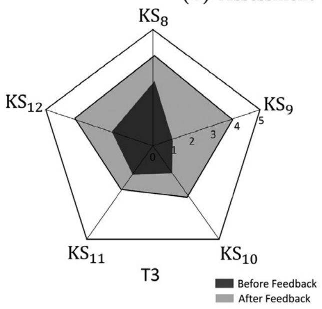
It is possible to extend the approach introduced in Pwnable-Sherpa to other CTF challenges, such as web, forensic, or reversing engineering because solving some challenges also requires multiple tasks. However, as each challenge has its own characteristics, detailed evaluation methods for each step should be developed separately. For example, to solve a web challenge requiring SQL injection, students will approach it step-wise (e.g., (1) find an injection vulnerability; (2) craft an appropriate SQL command according to the DMBS; and (3) gain access to the query results). To evaluate whether a student can find an SQL injection vulnerability, the assessment system may monitor the web server's access log and responses from the web server when the assessment system sends requests using the student's exploit instead of monitoring the program's crash. Details of Pwnable-Sherpa's extension to other CTF


timeline
Students solve challenges via *Pwnable-Sherpa* (1)
1st Round KS assessment (2)
Providing feedback to Students (3)
Students solve challenges via *Pwnable-Sherpa* (4)
2nd Round KS assessment (5)

(a) Timeline of two consecutive KS assessments



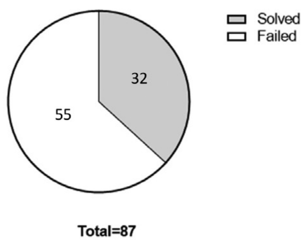
(b) Assessment results of student A



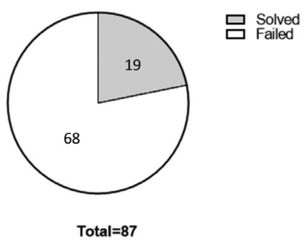
(c) KS self assessment results about T3

Fig. 13. Measuring the feedback's effects by comparison of two KS self-assessment results before and after feedback is given.

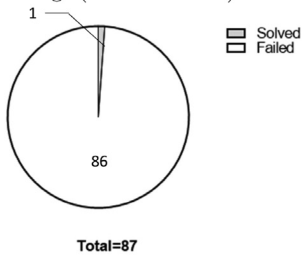
Pwnable-Sherpa



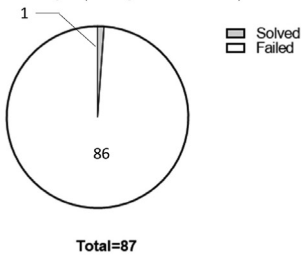
(a) Partial challenge (buffer overflow)



(b) Partial challenge (integer overflow)



(c) Synthesis challenge (buffer overflow)



(d) Synthesis challenge (use after free)

Fig. 14. Comparison of correct answer rate according to pwnable CTF challenges.

challenges seem to be another research topic (e.g., how we can design and implement a detailed assessment system for SQL injection CTF challenges), so we leave this as future work.

5. Lessons learned from the 2020 pwnable CTF competition

5.1. Challenges

Based on the workflow of the pwnable challenge described earlier, the following two types of challenges were presented at the 2020 CTF competition held by our research team at Kongju National University. The partial challenge can be solved by completing a single task (Task 1) for the entire exploit writing process. The synthesis challenge can be solved by completing at least two tasks during the entire exploit process.

5.1.1. Partial challenge

We designed challenges (a) and (b) such that only Task 1 (identifing vulnerabilities and triggering them) was required; students can solve the challenges without KS required for Tasks 2 and 3. For the challenges in Fig. 14(a) and (b), the vulnerabilities are the buffer and integer overflow, respectively. For example, if students successfully identify the vulnerability of a program and trigger it (e.g., buffer overflow), they can directly read the flag inserted into the program using the data area overflow such that they do not need to hijack the control flow of the program. Here, students have access to a core dump. Because the flag of the challenge is inserted into the program, students can print the flag in the core dump by triggering the programs' vulnerability to their local machine. Note that Pwnable-Sherpa was not used in this competition. This analysis was performed to identify a problem in evaluating the pass/non pass method of the existing pwnable CTF.

5.2. Synthesis challenge

The synthesis challenge requires at least two of the several tasks of the challenge-solving workflow to acquire the flag of the pwnable challenge (i.e., T1 and T2: CFH). Most pwnable problems in competitions require comprehensive KS to solve. For the challenges in Fig. 14(c) and (d), the program contains buffer overflow and use-after-free (UAF) vulnerabilities, respectively, and students are expected to have the required KS for these vulnerabilities to solve the challenge. In addition, for the problem in Fig. 14(d), mitigation techniques were not applied in the system or program, thus requiring students to acquire the flag by completing the task of capturing the control flow to solve the problem (i.e., T1 and T2). For the challenge in Fig. 14(c), the challenge binary was operated in an environment in which the NX and ASLR mitigation techniques were applied, thus requiring students to acquire the flag by bypassing the mitigation techniques (i.e., T1, T2, and T3).

5.3. Competition result analysis

The percentages of students who solved partial challenges during the CTF competition were 37% and 22%, as shown in Fig. 14(a) and (b), respectively. To solve challenges (a) and (b), only KS for T1 is required. In contrast, when comprehensive KSs were required to be solved for the synthesis challenges, only one out of 87 students solved the challenge correctly, as shown in Fig. 14(c) and (d). The challenge shown in Fig. 14(c) had the same vulnerability as that shown in Fig. 14(a). This indicated that no significant difference was observed in the knowledge required for T1. Thus, the difference in the number of solvers arises from solving other tasks (i.e., T2). These results reveal that using a dichotomous evaluation method for pwnable challenges complicates the provision of detailed feedback to students, because it is difficult to establish what the student does or does not know.

6. Conclusion

The CTF-style evaluation method widely used in cybersecurity competitions is challenging to apply in education because, in competitions, the focus is on ranking and not on improving students' KS. In this study, based on the results of CTF competitions (2017, 2018, 2020) conducted by our research team over the past 3 years, we determined that to use CTF in education and training classes, a function must provide sufficient feedback to students. For a detailed evaluation of Pwnable-Sherpa, the challenge-solving process was extracted from the stages of the CFH attack, and three detailed evaluation points were presented. However, providing detailed feedback to students can be labor-intensive and time consuming for educators; thus, alleviating this burden is necessary. Therefore, we proposed Pwnable-Sherpa, which can relieve educators' burdens using automatic challenge distribution, multi-container, simultaneous execution of detailed evaluation, and LLVM dummy pass while providing feedback. Based on Pwnable-Sherpa, educators can provide more feedback to students (e.g., KS sets for tasks). This process approaches a one-on-one coaching environment for educators and students.

In future work, we will adopt Pwnable-Sherpa in a training class to study it through interviews with students and educators.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

CRediT authorship contribution statement

Sung-Kyung Kim: Conceptualization, Software, Data curation, Writing – original draft, Visualization. **Eun-Tae Jang:** Software. **Hanjin Park:** Conceptualization, Methodology, Validation, Formal analysis, Writing – original draft, Writing – review & editing, Supervision. **Ki-Woong Park:** Conceptualization, Validation, Investigation, Resources, Writing – original draft, Writing – review & editing, Supervision, Project administration, Funding acquisition.

Data availability

The authors do not have permission to share data.

Acknowledgments

This work was supported by the Institute of Information & Communications Technology Planning & Evaluation (IITP), South Korea (Project No. RS-2022-00165794, Development of a Multi-Faceted Collection-Analysis-Response Platform for Proactive Response to Ransomware Incidents, 30%, and Project No. 2022-0-007010001003, 30%), and a National Research Foundation of Korea (NRF), South Korea grant funded by the Korean government (Project No. NRF-2020R1A2C4002737, 30%), and the ICT R&D Program of MSIT/IITP, South Korea (Project No. 2021-0-01816, A Research on Core Technology of Autonomous Twins for Metaverse, South Korea, 10%).

References

0 CTF, https://0ops.sjtu.cn/. [Online; accessed 09-April-2022].
Bletsch, T., Jiang, X., Freeh, V.W., Liang, Z., 2011. Jump-oriented programming: a new class of code-reuse attack. In: Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security. Association for Computing Machinery, New York, NY, USA, pp. 30–40. doi:10.1145/1966913.1966919.
Bufferoverflow, Buffer overflow protection. https://en.wikipedia.org/wiki/Buffer_overflow_protection#Canaries. [Online; accessed 09-April-2022].
Burns, T.J., Rios, S.C., Jordan, T.K., Gu, Q., Underwood, T., 2017. Analysis and exercises for engaging beginners in online CTF competitions for security education. 2017 USENIX Workshop on Advances in Security Education (ASE 17). USENIX Association, Vancouver, BC. https://www.usenix.org/conference/ase17/workshop-program/presentation/burns.
c0ntex, How to hijack the global offset table with pointers for root shells. http://www.infosecwriters.com/text_resources/pdf/GOT_Hijack.pdf. [Online; accessed 09-April-2022].
Chapman, P., Burket, J., Brumley, D., 2014. PicoCTF: a game-based computer security competition for high school students. 2014 USENIX Summit on Gaming, Games, and Gamification in Security Education (3GSE 14). USENIX Association, San Diego, CA. http://www.usenix.org/conference/3gse14/summit-program/presentation/chapman.
Chew, M., Song, D., 2002. Mitigating Buffer Overflows by Operating System Randomization. Technical Report. http://citeseerx.ist.psu.edu/viewdoc/summary?.
Chothia, T., Novakovic, C., 2015. An offline capture the flag-style virtual machine and an assessment of its value for cybersecurity education. 2015 USENIX Summit on Gaming, Games, and Gamification in Security Education (3GSE 15). USENIX Association, Washington, D.C.. https://www.usenix.org/conference/3gse15/summit-program/presentation/chothia.
Chung, K., 2017. Live lesson: lowering the barriers to capture the flag administration and participation. 2017 USENIX Workshop on Advances in Security Education (ASE 17). USENIX Association, Vancouver, BC. https://www.usenix.org/conference/ase17/workshop-program/presentation/chung.
CODEGATE CTF, http://codegate.org/en/. [Online; accessed 09-April-2022].
DEF CON CTF, https://www.defcon.org/html/links/dc-ctf.html. [Online; accessed 09-April-2022].
Designer, S., 1997. Linux kernel patch to remove stack exec permission. https://seclists.org/bugtraq/1997/Apr/31. [Online; accessed 09-April-2022].

Dhurjati, D., Kowshik, S., Adve, V., Lattner, C., 2003. Memory safety without runtime checks or garbage collection. SIGPLAN Not. 38 (7), 69–80. doi:10.1145/780731.780743.

Discord, https://discord.com/. [Online; accessed 09-April-2022].

Erlingsson, Ú., Younan, Y., Piessens, F., 2010. Low-Level Software Security by Example. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 633–658.

Executable space protection, https://en.wikipedia.org/wiki/Executable_space_protection#Windows. [Online; accessed 09-April-2022].

Forescout Research Labs and JFrog Security Research, Infra:Halt, Jointly discovering and mitigating large-scale OT vulnerabilities. https://www.forescout.com/resources/infrahalt-discovering-mitigating-large-scale-ot-vulnerabilities/. [Online; accessed 09-April-2022].

Hitcon CTF, https://ctf2021.hitcon.org/. [Online; accessed 09-April-2022].

Kim, S., Xu, M., Kashyap, S., Yoon, J., Xu, W., Kim, T., 2019. Finding semantic bugs in file systems with an extensible fuzzing framework. In: Proceedings of the 27th ACM Symposium on Operating Systems Principles. Association for Computing Machinery, New York, NY, USA, pp. 147–161. doi:10.1145/3341301.3359662.

Kim, S.-K., Jang, E.-T., Park, K.-W., 2020. Toward a fine-grained evaluation of the pwnable CTF. In: You, I. (Ed.), Information Security Applications. Springer International Publishing, Cham, pp. 179–190. doi:10.1007/978-3-030-65299-9_14.

Kolmogorov-Smirnov, Kolmogorov-smirnov test. https://en.wikipedia.org/wiki/Kolmogorov-Smirnov_test. [Online; accessed 09-April-2022].

LLVM Compiler Infrastructure, https://llvm.org/. [Online; accessed 09-April-2022].

LLVM StackProtector, https://github.com/llvm/llvm-project/blob/989f1c72e0f4236ac35a35cc9998ea34bc62d5cd/llvm/lib/CodeGen/StackProtector.cpp#L553. [Online; accessed 09-April-2022].

Matsakis, N.D., Klock, F.S., 2014. The rust language. In: Proceedings of the 2014 ACM SIGAda Annual Conference on High Integrity Language Technology. Association for Computing Machinery, New York, NY, USA, pp. 103–104. doi:10.1145/2663171.2663188.

Microsoft, 2020. Microsoft exchange validation key remote code execution vulnerability. https://msrc.microsoft.com/update-guide/vulnerability/CVE-2020-0688. [Online; accessed 09-April-2022].

MITRE, 2020. CWE top 25 most dangerous software weaknesses. https://cwe.mitre.org/top25/archive/2020/2020_cwe_top25.html. [Online; accessed 09-April-2022].

Nergal, 2001. Advanced return-into-lib(c) exploits (pax case study). http://phrack.org/issues/58/4.html. [Online; accessed 09-April-2022].

Park, H., Hong, S., 2019. Strategy for developing cybersecurity workforce in CSTEC: a link between lab-based training and a live-fire competition. In: 32nd Annual FISSEA Conference: Innovations in Cybersecurity Awareness and Training: A 360 Degree Perspective.

Park, J.-G., Choi, S.-H., il Kim, H., Hong, D., Park, K.-W., 2018. Our experiences on the design, build and run of CTF. In: The 4th International Conference on Next Generation Computing (ICNGC 2018).

Plaid CTF, https://plaidctf.com/. [Online; accessed 09-April-2022].

Rege, A., 2015. Multidisciplinary experiential learning for holistic cybersecurity education, research and evaluation. 2015 USENIX Summit on Gaming, Games, and Gamification in Security Education (3GSE 15). USENIX Association, Washington, D.C.. https://www.usenix.org/conference/3gse15/summit-program/presentation/rege.

Return Oriented Programing (ROP), https://en.wikipedia.org/wiki/Return-oriented_programming. [Online; accessed 09-April-2022].

Seibert, J., Okhravi, H., Söderström, E., 2014. Information leaks without memory disclosures: remote side channel attacks on diversified code. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security. Association for Computing Machinery, New York, NY, USA, pp. 54–65. doi:10.1145/2660267.2660309.

Serna, F.J., 2012. CVE-2012-0769, the case of the perfect info leak. In: Blackhat 12 USA Conference. https://paper.bobylive.com/Meeting_Papers/BlackHat/USA-2012/BH_US_12_Serna_Leak_Era_WP.pdf.

Shacham, H., Page, M., Pfaff, B., Goh, E.-J., Modadugu, N., Boneh, D., 2004. On the effectiveness of address-space randomization. In: Proceedings of the 11th ACM Conference on Computer and Communications Security. Association for Computing Machinery, New York, NY, USA, pp. 298–307. doi:10.1145/1030083.1030124.

Strackx, R., Younan, Y., Philippaerts, P., Piessens, F., Lachmund, S., Walter, T., 2009. Breaking the memory secrecy assumption. In: Proceedings of the Second European Workshop on System Security. Association for Computing Machinery, New York, NY, USA, pp. 1–8. doi:10.1145/1519144.1519145.

Szekeres, L., Payer, M., Wei, T., Song, D., 2013. SoK: Eternal war in memory. In: 2013 IEEE Symposium on Security and Privacy, pp. 48–62. doi:10.1109/SP.2013.13.

Theo de Raadt, Exploit mitigation techniques. https://www.openbsd.org/papers/ven05-deraadt/index.html. [Online; accessed 09-April-2022].

US-CERT Alert (AA21-131A), Colonial pipeline ransomware attack. https://us-cert.cisa.gov/ncas/alerts/aa21-131a. [Online; accessed 09-April-2022].

van der Veen, V., dutt-Sharma, N., Cavallaro, L., Bos, H., 2012. Memory errors: the past, the present, and the future. In: Balzarotti, D., Stolfo, S.J., Cova, M. (Eds.), Research in Attacks, Intrusions, and Defenses. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 86–106. doi:10.1007/978-3-642-33338-5_5.

van de Ven, A., 2004. New Security Enhancements in Red Hat Enterprise Linux. RedHat. https://people.redhat.com/mingo/exec-shield/docs/WHP0006US_Execshield.pdf.

Vykopal, J., Barták, M., 2016. On the design of security games: from frustrating to engaging learning. 2016 USENIX Workshop on Advances in Security Education (ASE 16). USENIX Association, Austin, TX. https://www.usenix.org/conference/ase16/workshop-program/presentation/vykopal.

Writing an LLVM Pass, https://llvm.org/docs/WritingAnLLVMPass.html. [Online; accessed 09-April-2022].

Zalewski, M., Bash bug: the other two RCEs, or how we chipped away at the original fix (CVE-2014-6277 and '78). https://lcamtuf.blogspot.com/2014/10/bash-bug-how-we-finally-cracked.html. [Online; accessed 09-April-2022].

Sung-Kyung Kim received the B.S. degree in the department of information security from Sejong University in 2019, the M.S. degree in the department of information security from Sejong in 2021. His research interests include system security.

Eun-Tae Jang received the B.S. degree in the department of information security from Sejong University in 2019, the M.S. degree in the department of information security from Sejong in 2021. He has been a researcher in INETCOP. His research interests include system security.

Hanjin Park received the B.S. degree in computer science from Yonsei University, Seoul, South Korea, in 2007, and the Ph.D. degree in computer Science from Korea Advanced Institute of Science and Technology (KAIST), Daejeon, South Korea, in 2015. He has been a senior researcher in the Affiliated Institute of ETRI. His research interests include computer network security, cybersecurity training, and cybersecurity competition.

Ki-Woong Park received the B.S. degree in computer science from Yonsei University, South Korea, in 2005, the M.S. degree in electrical engineering from the Korea Advanced Institute of Science and Technology (KAIST) in 2007, and the Ph.D. degree in electrical engineering from KAIST in 2012. He received a 2009–2010 Microsoft Graduate Research Fellowship. He worked for National Security Research Institute as a senior researcher. He has been a professor in the department of computer and information security at Sejong University. His research interests include security issues for cloud and mobile computing systems as well as the actual system implementation and subsequent evaluation in a real computing system.