Pwn Lessons Made Easy With Docker: Towards an Undergraduate Vulnerability Research Cybersecurity Class

TJ OConnor
toconnor@fit.edu
Florida Institute of Technology
Melbourne, FL, USA

Alex Schmith
aschmith2019@my.fit.edu
Florida Institute of Technology
Melbourne, FL, USA

Chris Stricklan
cstricklan@fit.edu
Florida Institute of Technology
Melbourne, FL, USA

Marco Carvalho
mcarvalho@fit.edu
Florida Institute of Technology
Melbourne, FL, USA

Sneha Sudhakaran
ssudhakaran@fit.edu
Florida Institute of Technology
Melbourne, FL, USA

ABSTRACT

Developing expertise in vulnerability research is critical to closing the cybersecurity workforce shortage. However, very few institutions have adopted vulnerability research into their cybersecurity curriculum, and fewer have examined how to teach this skill to students. The recent emergence of lightweight, container-based virtualization presents a unique opportunity to address this challenge by offering reproducible environments that ease course facilitation. This paper presents an undergraduate vulnerability course design. Our approach leverages a hands-on methodology that challenges students to develop complex binary exploits over our lectures, labs, and exams. We share our detailed design, labs, experiences, lessons learned, and a lightweight virtual environment for this course for others to build on our initial success.

CCS CONCEPTS

• **Social and professional topics** → **Model curricula**; **Computing education programs**.

KEYWORDS

cybersecurity education, vulnerability research, virtualization

ACM Reference Format:
TJ OConnor, Alex Schmith, Chris Stricklan, Marco Carvalho, and Sneha Sudhakaran. 2024. Pwn Lessons Made Easy With Docker: Towards an Undergraduate Vulnerability Research Cybersecurity Class. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE 2024), March 20–23, 2024, Portland, OR, USA*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3626252.3630911

1 INTRODUCTION

We exist in the proliferation era of offensive cyber weapons. Zero days, or vulnerabilities unknown to vendors, comprise the core of these nation-state weapon systems. While developing in-house government expertise is vital, nations rely on semi-regulated markets to discover and disclose vulnerabilities and develop exploits [26]. Developing expertise in vulnerability research is critical to closing our workforce shortage. The NSA Cyber Operations Outcomes and NICE Workforce Competencies capture Vulnerability Research as a Knowledge Unit [27]. However, very few institutions have adopted vulnerability research into their cybersecurity curriculum [30, 36, 41, 42], and fewer have examined how to teach this skill. Despite the workforce demand for vulnerability researchers, few institutions present formal coursework. Instead, the topic is studied orthogonally in cybersecurity clubs and applied capture-the-flag competitions. However, there is strong evidence to support this approach is failing to develop vulnerability researchers at scale. At the 2022 US National Cyber League tournament [24], only twenty (out of six thousand) college students solved the binary exploitation problem. Despite the importance of vulnerability research, less than one percent could solve a challenge that required overflowing a buffer and chaining together return-oriented programming gadgets.

A lack of instructor expertise in a constantly evolving topic may introduce significant barriers. In contrast to calculus, where an instructor must master pre-defined rules, a vulnerability research instructor must periodically examine how the continually evolving mitigations depreciate techniques. The following work presents an approach that helps overcome instructors' expertise. Relying on Docker lightweight virtualization, we provide a reproducible course environment with lecture examples, challenges, and labs. Further, our course approach follows best software development practices by pushing the course environment to a continuous integration environment where changes are automatically built, tested, and integrated into a shared repository. The following paper shares our experiences with a vulnerability research course. Developing complex binary exploits is critical to closing the cybersecurity workforce shortage. We acknowledge that reverse engineering and binary exploitation are complex topics to present in the classroom. Therefore, we share our systematic approach to inspire others to teach vulnerability research and build upon our work. This paper makes the following contributions:

(1) We share our course design, labs, exams, and lessons learned for a hands-on undergraduate vulnerability research course.
(2) To allow other instructors to build on our initial success, we publish our classroom slides, labs, exams, solutions, and virtualization containers at https://github.com/tj-oconnor/undergrad-vr.

2 PRIOR WORK

Previous works [30, 36, 37, 41, 42] have explored teaching vulnerability research. Due to the reliance on contemporary exploitation materials, many of these courses have been student-driven and focused on technical content instead of pedagogical approaches.

Modern Binary Exploitation: Eight years ago, Rochester Polytechnic Institute students developed the *Modern Binary Exploitation* course. Their course, taught entirely by students, focused on developing skills in vulnerability research, reverse engineering, and binary exploitation. For others to build upon their work, they archived their materials on GitHub [36]. Unfortunately, their course predominately focuses on the legacy x86 32-bit architecture and has not been updated since its release.

Automated Exploitation Competition: Our previous coursework developed an undergraduate reverse engineering course that culminates in an automated exploitation competition [30]. While their work examines teaching initial stack-based buffer overflows and format string attacks, it fails to address more complex exploit techniques in the stack, heap, and kernel.

How2Heap: The Shellphish team, aligned with the University of California Santa Barbara and Arizona State University, created the How2Heap heap exploitation repository [37]. The repository maintains the state of heap exploitation techniques, practical demonstration code, and capture-the-flag challenges. Further, their web application implements a web-based debugger to demonstrate required adaptions for each glibc version. As we discuss in Section 4, several of our students enjoyed referencing How2Heap when creating their heap exploitation presentations.

Pwn.College: As part of their CSE466 course, Arizona State University faculty created the *Pwn.College* [41] educational platform to deliver modules on binary exploitation. By creating an open forum, Pwn.college makes the topic of binary exploitation accessible to students. Further, their approach allows other faculty to incorporate their contributions. For example, we reference their materials on shellcoding [38, 40] in our coursework and leverage their *pwn.kernel* to introduce kernel exploitation [39].

Temple of Pwn: This North Dakota State University student-driven *Temple of Pwn* [42] course examined vulnerability research topics of stack-based buffer overflows, format string vulnerabilities, heap exploits, file stream-oriented programming, and kernel exploits. The course archive provides the challenge binaries, exploit scripts and fourteen lesson videos.

3 COURSE OVERVIEW AND DESIGN

The following section outlines the course model for our undergraduate vulnerability course. The course consists of two 75-minute lectures weekly for 16 weeks. It is the fifth course in a six-course sequence for our cyber operations concentration, following a rigorous reverse engineering course that culminates in an autonomous exploitation competition [28, 30, 31, 43]. The prerequisite coursework includes an assembly programming course. We designed the course to satisfy the optional NSA CAE Cyber Operations Knowledge Units for vulnerability research [27]. Based on this standard, the student outcomes include the following:

(1) List the various types of vulnerabilities, their underlying causes, their identifying characteristics, how they are exploited, and potential mitigation strategies; apply fundamental security design principles during system design, development, and implementation to minimize vulnerabilities.
(2) Identify a vulnerability in a given context; develop and replicate exploits to alternate contexts of vulnerabilities.
(3) Analyze existing source code for functional correctness; construct test cases demonstrating vulnerabilities; apply industry standard tools that analyze software for vulnerabilities.

Table 1: The course balances theory-based lessons with practical labs and contemporary research.

Lesson	Lab	References
Return to Libc	Speciality ROP	[3, 13]
Ret2CSU (The Universal Gadget)	Speciality ROP	[25]
Ret2DLResolve	Speciality ROP	[44, 49]
SigReturn Oriented Programming	Speciality ROP	[9, 51]
Jump Oriented Programming	Advanced ROP	[6, 47]
Blind Hacking Techniques	Advanced ROP	[5]
ROP on Aarch64 Architecture	Advanced ROP	[1, 8]
Integer Overflows	Type Vulns	[7]
Array Index Abuse	Type Vulns	[15, 45]
Type Confusion	Type Vulns	[48]
Shellcoding under Seccomp	Shellcoding	[38, 40]
Heap Overflows	Beginner Heap	[16, 37]
House of Force	Beginner Heap	[19, 32]
Fastbins Attack	Beginner Heap	[21]
Tcache Poisoning	Advanced Heap	[29, 50]
Unsafe Unlink	Advanced Heap	[20, 23]
Kernel Ret2Usr & Mitigations	-	[22, 35, 39]

The course topics, listed in Table 1, graduate students from the basics of stack-based binary exploitation to advanced techniques in the kernel and heap. These topics provide a comprehensive introduction into fundamentals for gaining arbitrary execution, corrupting memory, and exploiting the design in complex data structures. In each lecture, we deliver lessons that expose the technical underpinnings of exploit techniques. We follow lessons with hands-on labs that require students to replicate the methods to construct exploits. As recommended in [10, 17, 30], we incorporate materials from academic publications, technical reports, industry-standard documents, conference presentations, capture-the-flag competition write-ups, and security influencer videos.

4 COURSE IMPLEMENTATION

The following section explains our design and decisions behind the course's infrastructure, labs, and exams.

4.1 Dockerized Student Environment

We also standardized the student environment by providing a Docker container with all the appropriate course tools. We pushed the course container image to Dockerhub to allow for rapidly distributing the image to students. As recommended in [18, 30], Docker containers present an opportunity to baseline students and ensure

```
# python3 pwn-resolve.py BIN=./resolve GDB
[*] '/root/workspace/ret2dlresolve/resolve'                    | pwndbg> x/1x 0x4004b8
[*] Loaded 14 cached gadgets for './resolve'                   | 0x4004b8:       0x00404018
[*] symbtab found at 0x4003c0                                  | pwndbg> x/1i 0x00404018
[*] strtab found at 0x400420                                   | 0x404018 <gets@got.plt>:     push   rax
[*] jmp_rel found at 0x4004b8                                  | pwndbg> got
[*] writeable_mem at 0x404e00                                  | [0x404018] gets@GLIBC_2.2.5 -> 0x7f40252a6060 (gets <- push r13
[+] The JMPREL stores the Reloc. Table, mapping each entry     | pwndbg>
[+]  to a symbol. Examine 0x4004b8 to see it pointing to the   |
[+]  GOT entry for the gets() function.                        |
[*] Paused (press any to continue)                             |
```

Listing 1: Our approach leverages Docker and annotated pwntools scripts to make reproducible teaching materials, easing the facilitation of complex topics like dynamically resolving function addresses.

```
FROM tjoconnor/vr-hosting

# Copy vulnerable binary, flag, and libc version
COPY flag.txt /home/ctf/flag.txt
COPY ./bin/chal.bin /home/ctf
COPY libc/libc.so.6 /opt/libc.so.6
COPY libc/ld-2.27.so /opt/ld.so

# Set permissions to read-only for flag
RUN chown root:root /home/ctf/flag.txt
RUN chmod 644 /home/ctf/flag.txt

# patch binary to run with specific libc version
RUN pwninit --bin /home/ctf/chal.bin --ld /opt/ld.so --libc
     /opt/libc.so.6 --no-template
RUN mv /home/ctf/chal.bin_patched /home/ctf/chal.bin
```

Listing 2: Leveraging Docker allows other students and faculty to share new reproducible examples.

consistency. This approach delivers common platforms that allow better collaboration and support. We also provided exploit scripts for each lesson that display verbose debugging information to help students replicate our lectures. Listing 1 depicts an example script that provides notes with appropriate breakpoints and debugging information. Base-lining the environment ensured these scripts would display similarly on the instructor and student workstations. While Dockerhub routinely scans images for vulnerabilities, this does not present any significant issues since our course relies on custom binaries and scripts. In contrast, Dockerhub scans for known security issues with vulnerable applications, services, or operating system deployments.

4.2 Hosting Vulnerable Binaries

As depicted in Listing 2, we hosted vulnerable binaries in standardized Docker containers for the students to exploit as part of the lab assignments. Each lab instantiated an instance of the binary via the multipurpose network relay tool socat [33]. When appropriate to the specific challenge, we patched the binary using pwninit [4] to run with a particular version of glibc and loader. The binaries ran with restricted permissions with read-only access to a text file that contained a secret *flag*. As recommended by Burns [11], each lab challenged students to exploit the binary and gain arbitrary execution. Students demonstrated success by leveraging the arbitrary execution to read the contents of the secret flag file. We recorded students' progress by having them submit the flag to a CTFd [12] server. Further, we had students submit their solution scripts to our learning management system.

4.3 Course Infrastructure

We hosted our labs and exams on the CTFd capture the flag platform [12]. CTFd provides an accessible user interface to host services and challenges and collect student involvement statistics. To mitigate the security impacts of hosting vulnerable binaries on our university network, we leased CTFd hosting services for $60 monthly. This cost tier allowed us to host 30 remotely vulnerable challenges and permitted three million views of our challenges per month. We deployed our vulnerable binaries as Docker containers to the leased environment. Using Docker, we restricted the target environment to specific operating systems and dynamic library versions, constraining the problems to specific intended solutions. Further, we purchased $79 academic licenses for students for the Binary Ninja reverse engineering framework [46]. Although our students prefer the Python API and workflow for Binary Ninja, we acknowledge that the freely available Ghidra could substitute and offers similar disassembler, decompilation, and plugin support. To support this argument, several students installed a Binary Ninja Plugin that displayed the Ghidra high-level decompilation in a separate window pane.

4.4 Course Labs

We leveraged the course labs to provide students with a process-oriented assessment of their understanding of the course material. During the semester, we assigned students six labs. Each lab consisted of 2-4 vulnerable binaries for students to exploit based on the lesson techniques presented in Table 1. We themed the challenges after well-known music songs. As depicted in Listing 3, we themed the rather esoteric and complex JOP problem after the classic song *Jump Around by House of Pain*. Students proved their solutions by exploiting binaries on a remote server where we hosted a unique flag. We required students to submit a *write-up* of their solutions for their approach for each challenge. We focused our assessment on the students' problem-solving approach rather than

```
<<< Welcome to JOP Binary #1
--------------------------------------------------------
<<< Pack it up, pack it in, let me begin
<<< I came to win, battle me, that's a sin
<<< I won't ever slack up, punk, ya better back up
<<< Try and play the role and yo, the whole crew'll act up
<<< - Jump Around, House of Pain
--------------------------------------------------------
<<< Stack: 0x7ffcd98259ac
So get out your seat and jump around >>>
```

Listing 3: Our course leveraged an active learning approach by presenting students with binary exploitation problems themed after well-known songs.

if students succeeded in exploiting the vulnerable binaries. While we attempted to constrain the problems to narrow solutions that reinforced the lectures, we awarded extra points to students who identified unintended or particularly creative solutions.

Speciality ROP: We designed our Speciality ROP lab to develop the building blocks for the course by forcing students to struggle with debugging binaries, developing exploit strategies, and refining exploit scripts. As argued in [43], this struggle is essential to developing the workflows and mental models needed for reverse engineering and exploit development. The lab specifically tasked students to develop exploits that leveraged *Ret2Libc, Ret2CSU, Ret2DLResolve, and SROP* techniques. While pwntools provides functions to replicate *Ret2CSU, Ret2DLResolve, and SROP* [14], we developed our vulnerable binaries in ways that would cause pwntools functions to fail, requiring students to demonstrate the technique manually. For example, we rewrote the calling convention for the Ret2CSU binary, forcing students to demonstrate an understanding of the stack layout and register use.

Advanced ROP: The Advanced ROP lab forced students to apply more esoteric exploit techniques, including blind ROP (remotely developing exploits without the original binary), Jump Oriented Programming (JOP), and applying ROP to the Aarch64 architecture. These esoteric problems required greater thought and planning than in the earlier lab. For example, the Blind ROP problem required students to identify a crash using black box fuzzing, then attempt to remotely discover the address space of the procedure linkage table, .data, and .text sections by making hypotheses and testing small experiments. Understanding and chaining their experiments' results together allowed them to create a remote exploit without having the original binary. As a further example, the JOP problem required students to develop a dispatcher to chain a series of JMP gadgets into attacker code. Students can only solve this problem by developing a well-thought-out exploit plan.

Type Vulnerabilities: The Type Vulnerability Lab set the conditions for the upcoming block of instruction on heap exploitation with problems on integer overflows, array index abuse, and type confusion. It forced students to specifically examine the types and storage of variables, calculate address offsets, and begin working with pwndbg's visualization tools to understand the layout of struct data types. These problems were more straightforward to complete than the previous lab, allowing students to build confidence while rehearsing the necessary skills to move on to future, more complicated exploit techniques.

Shellcoding: The Shellcoding Lab served as an opportunity to address overcoming protection mechanisms and input restrictions. We provided students with two problems. First, students had to craft shellcode to find a flag at a random memory address with only the SYS_write system call enabled by seccomp. In the second, we restricted the users' input to only even-numbered bytes. This input restriction forced students to examine and craft alternate instructions. Instead of *xor rdi, rdi* to zero the RDI register, they had to select alternate instructions that could leverage the stack and arithmetic operations. We recognize the criticism of this block of instruction and lab. Jump-to-Shellcode exploits have been reliably defeated since implementing the hardware NX-bit and DEP/NX that mark the stack as non-executable. However, learning to shellcode under restricted conditions proves essential to process injection techniques, virtual machine escapes, antivirus evasion, just-in-time ROP, and some kernel exploit techniques.

Beginner Heap: The Beginner Heap lab introduced students to the dynamic memory allocator using three problems. The first problem tasked students to perform a heap overflow by overwriting the metadata and content of an adjacent memory chunk. The next challenge forced students to overflow a chunk adjacent to the heap wilderness to corrupt the top chunk size field, granting an arbitrary write primitive. Finally, the last problem challenges students to analyze how the glibc malloc() and free() algorithms manage dynamic memory. To succeed, students had to construct fake chunks forcing the manager to free the memory, granting a use-after-free vulnerability.

Advanced Heap: The Advanced Heap lab introduced students more to nuanced heap exploitation techniques. First, we tasked students to corrupt pointers in a thread-specific cache (tcache) on a modern glibc implementation. This challenge required students to overcome the safe-linking protection as we discuss in Section 5.1. Next, we tasked students to write an exploit that benefited from the heap memory manager coalescing memory for optimization. Both exploits required a deep understanding of the protection mechanisms and algorithms used by the modern heap memory manager.

4.5 Presentations

We assigned students three presentation subjects for the semester. This experiential learning approach allowed students to gain hands-on experience with contemporary problems. Our first two presentations required students to individually solve *pwn* problems from capture-the-flag competitions. As we published our course container to the DockerHub repository, students could easily build upon this work to distribute reproducible examples for their peers as depicted in Listing 2.

My Pwn Presentations: First, the *My First Pwn* lab tasked students to find and present an archived capture-the-flag competition with existing solutions. Second, the *My First Solve* lab challenged students to develop and present a solution for a current competition. During the second lab, due to the timing and appropriate difficulty, students solved challenges from the *laCTF, diceCTF* and *irisCTF* competitions. The *KnightCTF* competition presented an issue for students and an opportunity for our faculty. During this competition, the organizers presented three problems with intended naive solutions. However, the competition organizers incorrectly compiled the binaries with protection mechanisms that effectively made the challenges unsolvable. Unfortunately, some of our students committed inappropriate time efforts to these challenges. Upon noticing, we patched the binaries to disable the protection mechanisms and reissued the intended and solvable binaries to our students. This pivot for the original presentation topic allowed us to discuss binary patching with students.

House of Student: The final presentation, *House of Student* tasked students to work in groups of 2-3 to present a modern heap exploitation technique in 20 minutes. The title, *House of Student*, pays homage to the Malloc Maleficarium [32] heap exploitation reference, which named heap exploitation techniques after various *houses*. During this presentation, our students explained complex topics such as *The House of Orange*, which abuses the fast bins metadata to create a File Stream Oriented Programming (FSOP) attack. Students often cited and referenced the How2Heap [37] educational repository when presenting complex heap exploitation techniques. The reference, developed by the Shellphish CTF team, provides a web-based debugger to demonstrate techniques across different glibc versions. As the House of Student occurred near the end of the course, it gave us an excellent assessment of students' course-long progress. In several cases, students excitedly exceeded the 20-minute reserved block.

4.6 Exams

We reserved two class periods for the course exam. We presented students with a single binary to exploit during each period. We permitted the use of notes and internet resources during both but restricted any collaboration with other students. The first exam question was a single binary written in assembly code. We directed students to solve using a *signal-return-oriented programming* (SROP) attack. The second exam, written in C and compiled with a legacy libc version, directed students to solve using a *house-of-force attack* heap exploitation technique. We provided students with accommodations, an alternate distraction-free location to take the exam with an appropriate time modification.

Both final questions required satisfying four primitives. For the SROP attack, the student must 1) develop a buffer overflow, 2) set the RAX register to 0xf, and 3) trigger a sigreturn system call, which will 4) restore the state of the registers from a sigreturn frame. For the house-of-force exploit, the student needed to 1) leak the heap and libc base addresses, 2) overwrite the heap top chunk size address, 3) perform an arbitrary write of the malloc hook with the address of the libc *system()* call, and then 4) trigger malloc() on a character pointer to a shell command.

While both challenges have similar difficulty levels based on the number of primitives, our students performed better on the SROP challenge than on the heap exploit. All students completed the SROP challenge in the class period. In contrast, only two students completed the heap challenge within the original class period. We extended the class period by 30 minutes, allowing 70% of the students to solve the heap challenge. We observed that students struggled to correctly identify leaks, often confusing a stack address leak with a libc address leak. Based on this observation, we intend to dedicate more instruction on virtual address space layout in future course offerings.

Table 2: Maintaining vulnerable research classes requires constant diligence to understand how software and hardware mitigations complicate exploit approaches.

Patch	Glibc Version	Date
Top chunk size integrity check added	2.29	2/1/19
Safe-Linking pointer-mangling added	2.32	8/5/20
Lib_csu_init static code removed	2.32	8/5/20
Malloc_hook removed	2.34	8/2/21

5 LESSONS LEARNED

This section shares the challenges and successes we encountered developing our vulnerability research course.

5.1 Challenges

Course Material Maintenance: We hypothesize that one of the reasons for limited formal vulnerability research classes is the continual cat-and-mouse game between novel exploit techniques and mitigations. Ultimately, no instructor wants to be the aging professor teaching return-to-stack exploits twenty-two years after Intel implemented hardware support for the non-executable bit. Maintaining vulnerable research classes requires constant diligence to understand how software and hardware mitigations complicate exploit approaches. This challenge will only continue to grow as Intel moves ahead with its Control-flow Enforcement Technology (CET) that introduces a shadow stack and technology to prevent control-flow deviations. However, exploit migitations are an opportunity to study the unique creativity of cybersecurity professionals. Table 2 lists a set of the glibc patches that affected our course materials. During our course, we demonstrated the *House of Force* heap exploitation technique that corrupts the top chunk size field to create an arbitrary write-primitive outside the heap address space. However, the glibc developers created a successful patch in 2019 that prevents the *House of Force* by ensuring the top chunk size does not exceed the scope of available heap memory. We chose to study this exploitation technique to demonstrate a successful patch. In contrast, glibc version 2.32 introduces safe-linking that leverages address space layout randomization to sign the tcache pointers. During our lesson on tcache poisoning, we show how an attacker can overcome this protection mechanism by leaking the head of the tcache list, which XORs the randomness with zero. Some patches, such as the lib_csu_init static code removal, prevented the *Ret2CSU* technique but only introduced nominal changes to our approach for *Blind Return Oriented Programming (BROP)*. We hypothesize that this very cat-and-mouse game is the essence of vulnerability research and demands student study. As such, we believe in showing four-year-old heap exploitation techniques that no longer succeed on contemporary glibc versions.

Inappropriate cooperation: We provide a healthy collaborative learning environment, allowing students to present in groups and seek peer support on problems. However, we are concerned about inappropriate cooperation. We acknowledge that inappropriate student cooperation presents a unique challenge for the course. In healthy collaboration, a student may explain to their peer how an exploit technique works at a high theoretical level. During inappropriate collaboration, students may give their peers a copy of their write-up script and tell them to duplicate the exploit with slightly different variable names. As examined in [43], unintended and inappropriate student collaboration eliminates the struggle to produce creative vulnerability research skills. This approach is similar to calculus, where students must fight through comprehending basic rules, simplifying expressions, and determining which rules apply in particular contexts. With inappropriate help that skips the struggle, students may complete assignments but require external help to solve problems on future assessments. For example, we provided the students with a Sigreturn Oriented Programming (SROP) challenge that required the primitive of the RAX register to be set to 0xf. However, we did not give the students gadgets to manipulate the RAX register. Instead, we intended for students to struggle until they identified that they could call a function, such as strlen(), that returned a value into the RAX register. We noticed clusters of solutions times during this lab and other labs. It proved challenging to ascertain the level of unintended cooperation. Still, we grew concerned enough that we conducted the course exams in person, constraining them to two separate class periods. While we were pleasantly surprised with positive exam results that indicated students comprehended the material, we still worried about the impact of unintended collaboration. In future course offerings, we reserve the right to and plan to work on autonomously creating unique binary challenges for students.

It is All About the Heap: While students provided overwhelmingly positive course feedback, they argued they would benefit from additional and more advanced heap-based exploitation materials. As the course progresses over the next few years, adopting Intel's Control-Flow Enhancement Technology (CET) will significantly mitigate stack-based exploit techniques. This paradigm shift demands a course focus on the future domain rather than the current problem. We hypothesize that Heap-based exploitation techniques will offer solutions to these challenges. While CET will diminish the relevance of several ROP-based exploitation techniques, it will increase the importance of esoteric exploit techniques such as Function-Oriented-Programming (FOP). We agree with the student responses that we must examine this impact as we look forward to future course offerings.

5.2 Successes

Visualizing Exploit Techniques: Several students reported having anxiety about heap exploitation before the class. Their fears were justified as the dynamic memory allocator relies on complex data structures and memory management algorithms. However, students identified that pwndbg [34], a Python module loaded directly into the gdb debugger, significantly helped to visualize heap exploitation techniques. As pwndbg color codes memory chunks, students reported they could recognize when they had corrupted an adjacent memory chunk. Further, students appreciated pwndbg's exploration analytical tools to inspect unallocated memory bins. Leveraging pwndbg significantly benefitted in allowing students to reproduce classroom lectures and explore additional heap exploitation techniques during their presentations. While this finding did not necessarily surprise us, we were interested that students did not identify pwndbg's contribution to their understanding of stack-based overflows, type confusion attacks, or type confusion. During these portions, students leveraged pwndbg analytical commands, including *canary, stack, got, context* to view the binary state.

Delivering Workforce Opportunities: Early versions of the course were taught by an adjunct faculty member who recruited talented students to work in the vulnerability research career field. This connection to industry is one of the strongest contributions of our course. With later revisions, a tenure-track faculty member took over the course, formalizing it in Docker, so instructors with varying degrees of expertise could repeat it. During the course, we invited several experts from local companies, including Vector35, Cromulence, Raytheon, STR Inc., Research Innovations Inc., and Red Lattice, to present their research and share experiences from the workforce. We received positive feedback from the experts, who observed that reverse engineering and binary exploitation are rarely taught at universities, let alone at the undergraduate level [2], and praised our approach. We asked the experts for recommendations for improving the course for students entering the workforce. They consistently provided feedback about providing instruction on emulation, embedded architectures, and fuzzing (both generative and symbolic.) We will look for opportunities to incorporate this feedback into future course offerings. True to the origin of the course, a majority of our students pursued internships or full-time employment with vulnerability research companies.

6 CONCLUSION

In this paper, we presented our undergraduate course on vulnerability research. Our approach relies on experiential learning and process-oriented assessment labs. Despite limited adoption, vulnerability research is essential to closing the cybersecurity workforce gap. We uncover that despite the challenges, vulnerability research offers workforce opportunities. To take advantage of these opportunities, we rely on lightweight virtualization. We demonstrate how the recent emergence of lightweight, container-based virtualization presents a unique opportunity by providing reproducible environments that ease course facilitation. We share our detailed design, labs, virtual containers, and lessons learned from this course for others to build on our initial success.

ACKNOWLEDGEMENTS

This material is based upon work supported in whole or in part with funding from the Office of Naval Research (ONR) contract #N00014-21-1-2732. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the ONR and/or any agency or entity of the United States Government. The authors also thank John Aycock for his suggestions and feedback to improve an earlier draft of this paper.

REFERENCES

[1] ARM. 2019. Procedure Call Standard for the Arm® 64-bit Architecture. https://developer.arm.com/documentation/102374/0100/Procedure-Call-Standard. Accessed: May 9, 2023.

[2] John Aycock, Andrew Groeneveldt, Hayden Kroepfl, and Tara Copplestone. 2018. Exercises for teaching reverse engineering. In *Conference on Innovation and Technology in Computer Science Education*. ACM, Larnaca Cyprus, 188–193.

[3] Niklas B. 2015. libc-database: a collection of libc versions and corresponding offsets to be used in exploit development. https://github.com/niklasb/libc-database. Accessed: May 9, 2023.

[4] Benjamin Levy. Year. pwninit: A tool for automating starting binary exploit challenges. hhttps://github.com/io12/pwninit. Accessed: December 10, 2023.

[5] Andrea Bittau, Adam Belay, Ali Mashtizadeh, David Mazières, and Dan Boneh. 2014. Hacking blind. In *Symposium on Security and Privacy*. IEEE, San Jose, CA, 227–242.

[6] Tyler Bletsch, Xuxian Jiang, Vince W Freeh, and Zhenkai Liang. 2011. Jump-oriented programming: a new class of code-reuse attack. In *Symposium on Information, Computer and Communications Security*. ACM, Hong Kong, China, 30–40.

[7] blexim. 2002. Basic Integer Overflows. Phrack Magazine, Volume 0x0b, Issue 0x3c, Phile #0x0a of 0x10.

[8] Perfect Blue. 2021. ROPing on Aarch64. https://blog.perfect.blue/ROPing-on-Aarch64. Accessed: May 9, 2023.

[9] Erik Bosman and Herbert Bos. 2014. Framing signals-a return to portable shellcode. In *Symposium on Security and Privacy*. IEEE, San Jose, CA, 243–258.

[10] Sergey Bratus. 2007. What hackers learn that the rest of us don't: notes on hacker curriculum. *IEEE Security & Privacy* 5, 4 (2007), 72–75.

[11] Tanner J Burns, Samuel C Rios, Thomas K Jordan, Qijun Gu, and Trevor Underwood. 2017. Analysis and Exercises for Engaging Beginners in Online CTF Competitions for Security Education. In *2017 USENIX Workshop on Advances in Security Education (ASE 17)*. USENIX, Vancouver, BC, Canada.

[12] Kevin Chung. 2017. Live Lesson: Lowering the Barriers to Capture The Flag Administration and Participation. In *2017 USENIX Workshop on Advances in Security Education (ASE 17)*. USENIX, Vancouver, BC, Canada, 6 pages.

[13] Solar Designer. 1997. Return to Libc Exploit. Bugtraq mailinglist.

[14] Gallopsled et al. 2023. pwntools: CTF Framework and Exploit Development Library. https://github.com/Gallopsled/pwntools. Accessed: June 14, 2023.

[15] Fabian Faessler. 2016. Global Offset Table (GOT) and Procedure Linkage Table (PLT) - bin 0x12. https://youtu.be/kUk5pw4w0h4

[16] Fabian Faessler. 2016. The Heap: what does malloc() do? - bin 0x14. https://youtu.be/HPDBOhiKaD8

[17] Robert Fanelli and TJ OConnor. 2010. Experiences with Practice-Focused Undergraduate Security Education. In *Cyber Security Experimentation and Test (CSET)*. USENIX, Washington, DC.

[18] Kourtnee Fernalld, TJ OConnor, Sneha Sudhakaran, and Nasheen Nur. 2023. Lightweight Symphony: Towards Reducing Computer Science Student Anxiety with Standardized Docker Environments. In *Special Interest Group on Information Technology Education (SIGITE)*. ACM, Marietta, GA.

[19] François Goichon. 2015. Glibc adventures: The forgotten chunks. Context Information Security.

[20] Ir0nstone. 2021. Dream Diary: Chapter 1. https://ir0nstone.gitbook.io/hackthebox/challenges/pwn/dream-diary-chapter-1. Accessed on May 9, 2023.

[21] Sajjad Jadium. 2017. 0CTF 2017 Quals: Babyheap. https://github.com/sajjadium/ctf-writeups/tree/master/ctfs/0CTF/2017/Quals/babyheap. Accessed: May 9, 2023.

[22] Lkmidas. 2021. Linux Kernel Pwn - Part 1. https://lkmidas.github.io/posts/20210123-linux-kernel-pwn-part-1/. Accessed on May 9, 2023.

[23] David Manouchehri. 2017. Overview of glibc Heap Exploitation Techniques. https://0x434b.dev/overview-of-glibc-heap-exploitation-techniques/#unsafe-unlink. Accessed on May 9, 2023.

[24] Daniel Manson and Anna Carlin. 2011. A league of our own: the future of cyber defense competitions. In *Communications of the IIMA*, Vol. 11. IIMA, New Orleans, LA, 1.

[25] Hector Marco-Gisbert and Ismael Ripoll. 2018. Return-to-csu: A new method to bypass 64-bit Linux ASLR. Black Hat Asia.

[26] Charles Miller. 2007. The Legitimate vulnerability market: the secretive world of 0-day exploit sales.. In *Workshop on the Economics of Information Security (WEIS)*. Carnegie Mellon University, Hanover,NH.

[27] NSA. 2022. Academic Requirements for Designation as a CAE in Cyber Operations Fundamental. https://www.nsa.gov/Resources/Students-Educators/centers-academic-excellence/cae-co-fundamental/requirements/

[28] TJ OConnor. 2022. HELO DarkSide: Breaking Free From Katas and Embracing the Adversarial Mindset in Cybersecurity Education. In *Special Interest Group on Cyber Security Education (SIGCSE)*. ACM, Virtual Event.

[29] TJ OConnor. 2022. NiteCTF 2022 : Elementary Tcache Write-Up. https://github.com/tj-oconnor/ctf-writeups/tree/main/nitectf/heapchall. Accessed: May 9, 2023.

[30] TJ OConnor, Carl Mann, Tiffanie Petersen, Isaiah Thomas, and Chris Stricklan. 2022. Toward an Automatic Exploit Generation Competition for an Undergraduate Binary Reverse Engineering Course. In *Innovation and Technology in Computer Science Education (ITiCSE)*. ACM, Dublin, Ireland.

[31] TJ OConnor and Chris Stricklan. 2021. Teaching a Hands-On Mobile and Wireless Cybersecurity Course. In *Innovation and Technology in Computer Science Education (ITiCSE)*. ACM, Virtual Event.

[32] Phantsmal Phantasmagoria. 2005. The malloc maleficarum. Bugtraq mailinglist.

[33] Gerhard Rieger. 2023. socat: Multipurpose relay for bidirectional data transfer. http://www.dest-unreach.org/socat/. Accessed: December 10, 2023.

[34] Zach Riggle. 2018. pwndbg: Exploit Development and Reverse Engineering with GDB. https://github.com/pwndbg/pwndbg. Accessed: June 14, 2023.

[35] Christopher Roberts. 2021. Linux Kernel Exploit Development. https://breaking-bits.gitbook.io/breaking-bits/exploit-development/linux-kernel-exploit-development. Accessed on May 9, 2023.

[36] RPISEC. 2015. MBE: Modern Binary Exploitation. https://github.com/RPISEC/MBE. Accessed: June 14, 2023.

[37] Shellphish. 2018. how2heap. https://github.com/shellphish/how2heap. Accessed: May 9, 2023.

[38] Yan Shoshitaishvili and Connor Nelson. 2021. Lesson 5: Common Challenges Shellcoding. https://pwn.college/modules/shellcode. Accessed: May 9, 2023.

[39] Yan Shoshitaishvili and Connor Nelson. 2021. Module: Kernel Security. https://pwn.college/modules/kernel.html. Accessed: May 9, 2023.

[40] Yan Shoshitaishvili and Connor Nelson. 2021. Sandboxing: Escaping seccomp. https://pwn.college/modules/sandbox.html. Accessed: May 9, 2023.

[41] Yan Shoshitaishvili and Connor Nelson. 2023. *Pwn College*. Arizona State University. Accessed: June 12, 2023.

[42] Logan Stratton. 2020. *Temple of Pwn*. North Dakota State University. Accessed: June 12, 2023.

[43] Chris Stricklan and TJ OConnor. 2021. Towards Binary Diversified Challenges For A Hands-On Reverse Engineering Course. In *Innovation and Technology in Computer Science Education (ITiCSE)*. ACM, Virtual Event.

[44] syst3mfailure. 2021. ret2dl_resolve - A Classic Technique for Modern Times. https://syst3mfailure.io/ret2dl_resolve. Accessed: May 9, 2023.

[45] TIS Committee. 1993. Tool Interface Standard (TIS): Portable Formats Specification Version 1.1. http://refspecs.linux-foundation.org/elf/TIS1.1.pdf

[46] Vector35. 2022. Binary Ninja. https://binary.ninja

[47] violenttestpen. 2022. CTF Writeup: CTFSG CTF 2021 - reverse. https://violenttestpen.github.io/ctf/pwn/reverse/2022/03/11/ctfsg-ctf-21/. Accessed: May 9, 2023.

[48] Caitlin Whitehead. 2021. Unionized Write-Up. https://blog.metactf.com/unionized-cybergames-2021/. Accessed: May 9, 2023.

[49] Rafal Wojtczuk. 2001. The advanced return-into-lib (c) exploits: Pax case study. *Phrack Magazine, Volume 0x0b, Issue 0x3a, Phile# 0x04 of 0x0e* 70 (2001), 227–242.

[50] Dmitry Zakharov. 2020. [PATCH] Fix integer overflow in getaddrinfo. https://sourceware.org/pipermail/libc-alpha/2020-March/111631.html. Accessed: May 9, 2023.

[51] Michal Zalewski. 2001. Unix Signal Reference. https://lcamtuf.coredump.cx/signals.txt. Accessed: May 9, 2023.