Finding the Balance Between Guidance and   
Independence in Cybersecurity Exercises

**Abstract**

In order to accomplish security tasks, one needs to know how to analyze complex data and when and how to use the tools. Many hands-on exercises for cybersecurity courses have been developed to teach these skills. There is a spectrum of ways that these exercises can be taught. On one end of the spectrum are prescriptive exercises, in which students follow step-by-step instructions to run scripted exploits, perform penetration testing, do security audits, etc. On the other end of the spectrum are open-ended exercises and capture-the-flag activities, where little guidance is given on how to proceed.

This paper reports on our experience with trying to find a balance between these extremes in the context of a suite of cybersecurity exercises that we have developed. We have found that students are most successful in these exercises when they are given the right amount of prerequisite knowledge and guidance as well as some opportunity to find creative solutions. Our scenarios are specifically designed to develop analysis skills and the security mindset in students and to complement the theoretical aspects of the discipline and develop practical skills.

# 1 Introduction

When we choose hands-on exercises for our classes, we are faced with an apparent dilemma. Do we provide a step-by-step description with nothing left for the student to discover, or do we only describe a challenge and students must figure out what to do and how to do it? Many faculty believe that inquiry-based learning is the gold standard [cite inquiry-based lit], while others are frustrated by the slow pace that this approach could entail. In addition, the structure of many classes does not allow students to explore the material at their own rate.

Hands-on exercises provide an opportunity for differential instruction. To the extent that instructors have multiple exercises to choose from and exercises that are flexible, they can decide how much guidance to give students. Note that difficulty and degree of guidance are related but not identical parameters for instruction. Difficulty has to do with the level of abstraction and the complexity of a system that is being described. Guidance has to do with the size of the steps from the user’s current level of knowledge or understanding to that which is required to complete the exercise. Often learning a concept or knowledge area is broken down into multiple levels of difficulty. Students can progress from one level to the next once they have mastered the skills and concepts of the former. The degree of guidance has to do with the hints, clues, examples, and explanations that are given to the student, as well as the difference in difficulty of successive levels. [Does Soloway deal with this? How do we measure these scales?] In their Usenix presentation, Chung and Cohen discuss the escalation in difficulty of challenges in the CSAW CTF and the fact that stdudents can become discouraged when faced with difficult challenges in a competition [youtube.com]

One of our primary goals is to create exercises that nurture *analysis skills*. When speaking of analysis skills, we mean the ability to reason about large, complex, and opaque data and systems. Strong analytical skills enable people to impose structure and meaning on such artifacts, reason about these relationships, and draw meaningful conclusions or inferences. These are precisely the kinds of skills that we believe are useful in many cybersecurity scenarios, from security policy design to reverse engineering to vulnerability analysis. Analysis skills complement the *security mindset*, which is the ability to think about how systems can fail, and be made to fail in different ways, even as one is designing a system. Questioning assumptions plays an important role in both defense and attack. In designing our exercises, we focus on the following analysis skills:

1. Verifying assumptions by checking network messages, protocols, file formats and other input data constraints to see if layers of abstraction are coherent and correct; enumerating and checking if failure modes, exceptions, and errors are controlled, caught or anticipate
2. Gaining understanding of program, network, or system behavior and semantics, network topology or organization, or a defense posture; observing and enumerating how software components or network elements are actually composed
3. Extracting Information from opaque artifacts. For example, analyzing a raw dump of network traffic, intrusion alerts, firewall logs, system call traces, or executable files and recognizing anomalies in a mass of otherwise normal data.
4. Creating Emergent Resilience by understanding a system well enough to design and propose enhancements to reliability, fault tolerance, or availability.

Most students, and even many computer science faculty, do not have these skills or the prerequisite knowledge to distinguish normal behavior from abnormal.

In this paper, we report on four hands-on cybersecurity scenarios that we developed and what we discovered about the issues of guidance for our students in the context of these scenarios.

# 2 Related Work

Our philosophy on information security education stems from our understanding and teaching of the hacker curriculum as described by Bratus [1]. This approach is predicated on the utility of understanding failure modes. Rather than teaching students the “success” cases, we attempt to deliver a culture shock that makes them disrespect API boundaries and adopt a cross-layer view of the CS discipline as described by Bratus et al. [3]. We also routinely encourage our students to adopt a dual frame of mind (attacker and defender) when solving problems to prevent artificial abstraction layers from becoming boundaries of competence [19]. The importance of analysis skills as explained by Bratus et al. [2] is based on linking expected behavior to actual behavior as seen in network traces, log files, program binaries, rules/policies, system call traces, network topologies, network interactions, unknown protocols, injected backdoor code, etc. All of our exercises are based on these skills. A tool that actually applies this type of analysis is NetCheck [20], which is used to debug network applications. Using a simplified model of normal network behavior, NetCheck collects information about network applications using strace and compares it to the normal model.

There are a number of well-known exercises that are on the prescriptive side of the spectrum. For example, Towson’s “Security Injections” [16] mainly focus on several important secure programming patterns, but do not emphasize analysis. The SEED [8] project presents a mature, well-documented set of exercises, which are not typically interactive or dynamic and require significant work to set up and run. They are very prescriptive in terms of their description of what the student does. Some of them were designed for the graduate level. The lab manual written by Nestler et al [22] provides a broad overview of cyber security, yet is also prescriptive. The exercises described by Yuan et al [21] seem to emphasize tools for auditing software, but not the analysis skills.

[make a distinction from 2nd paragraph] Cybersecurity games and capture-the-flag competitions are known to engage students [17, 11]. This includes competitions such as CCDC[[1]](#footnote-2), Plaid[[2]](#footnote-3), notsosecure[[3]](#footnote-4), iCTF[[4]](#footnote-5) [7], CSAW[[5]](#footnote-6) [11], TRACER FIRE[[6]](#footnote-7), Packetwars[[7]](#footnote-8), and many others. These activities often provide little guidance and because they require a significant amount of infrastructure and preparation by the organizers, they only reach a small number of students. Some competitions such as CCDC and Packetwars require the installation of physical hardware, and they often require that students and their faculty travel to participate. However, they are moving to virtual environments and qualifying rounds for CCDC are run remotely. There are also a number of non-technical games with the goal of interesting students with no technical background in cybersecurity. These include Control-Alt-Hack [6], [d0x3d!] [12], Security Cards[[8]](#footnote-9), CyberCIEGE[[9]](#footnote-10) [5] and Werewolves [9]. The last of these introduces players to the concept of covert channels in a non-technical context. Our exercises are intended to create scenarios that are closer technically to real-world situations that a security professional would face. We want exercises that our students can use in the classroom and even as training for some of these competitions if they are really attracted to them. We view the exercises described in the next section as a middle ground between the two ends of the spectrum.

# 3 Description of the Scenarios

For this paper, we focus on degree of guidance that was needed by the students and what they learned.

## 3.1 strace

strace is a Linux tool that generates a trace of all the system calls made during the execution of a program. These include memory, file, process, and networking operations. It is used to analyze the runtime behavior of programs, especially executables for which there is no source code. In the context of cybersecurity, strace is useful for detecting if a program is doing something unusual and potentially malicious, such as reading information from unexpected sources, writing information to an obscure file, surprisingly forking a child process, or performing suspicious network communication.

There are mulitple goals for this exercise. One of them is for students to learn how to use strace to understand the operation of normal programs and detect abnormal behavior in malicious programs. As with many tools for security analysis, effectively using strace to detect abnormal behavior requires (1) sifting through a large amount of data and (2) being able to distinguish abnormal behavior from normal behavior. For example, the output of strace when running the empty C program compiled from int main () {} has 23 lines involving 11 different system calls, many of which (e.g., brk, ftstat64, mmap2, munmap, mprotect) only make sense to those with some background in the Linux operating system. This output summarizes system calls made when running any program. Students must learn to ignore them in order to focus on calls made by the program itself. So the strace scenario begins with an exercise on studying the output of strace running on the empty program. However, it is important to let students know not to focus on understnding these sytem calls. We found that without that guidance,

students readily explored that path which did not help them solve the challenges of the exercise.

In the next exercise, students are asked to explain the output of strace for a C program that performs a character-by-character copy of an input file to an output file. This exposes them to system calls for opening and closing files and reading and writing file information, and they can correlate aspects of the dynamic execution reported by strace with static features of the program.

Running strace on even simple programs can easily generate hundreds or thousands of lines of output, so it is important for students to learn ways to filter and summarize this information. There are options for counting the number of calls made to different system routines and summarizing the time spent in them as well as options for showing only certain calls or categories of calls (e.g., file operations, process management operations, networking operations). By default, strace does not trace calls of child processes forked from the main process, but there is an option for doing this. This is important to know, because malicious code often create new child processes. There are exercises introducing all of these options.

Once they have learned the basics of strace, students are asked to use it to analyze some executables for which they have no source code. These exercises culminate in the trojaned cat exercise, where the usual Linux cat command for displaying file contents has been replaced by a trojaned version that additionally writes the contents of every displayed file to a special directory. strace exposes the operations that open, write to, and close the file in the special directory.

The strace scenario was tested in Fall 2014 in a computer security course at a liberal arts college. There were 29 students in this class who worked on the exercises in groups of two or three over two 70-minute class sessions. Initially, the *instructor circulated around the room* while students worked on the strace exercises from a handout, observing what they were doing and answering questions. It soon became apparent that most students did not understand the purpose of examining the output of strace on the empty program. None of the students had operating systems experience, and many were trying to understand the details of system calls like brk and mmap2. The *instructor stepped in to explain* that the point of this exercise was to show that many system calls are made to run any program, and that these form a boilerplate that can be ignored in subsequent operations. Similarly, for the copy exercise, the instructor needed to emphasize that the point of the exercise was to relate file operations from the dynamic trace to lines of the program. By the end of the second session, most groups had spent significant time on the trojaned cat problem, and most had discovered that the program was surreptitiously squirreling away copies of the displayed files.

This test of the strace scenario highlighted issues involving the level of guidance and independence in hands-on security exercises. [some of these points are redundant, we could shorten this]

* Does the instructor tell the students how the tool works, what it can do, what to pay attention to and what to ignore? Or are these left for the student to discover by running simple examples?

Does the instructor present students with a general problem first, or show them an example? The instructor can choose the level of guidance. Our early experience was that students could be distracted by investigating what the empty program actually does vs. moving on to examine programs that do more interesting things. Similarly, students could get caught up reading about all of the different system calls and lose sight of the goal of finding anomalous behavior. In some cases, it may be worthwhile for the instructor to give a demo while the students follow along, rather than having the students try to figure out the same material from an exercise.

* When it comes to understanding options for a tool like strace, there is a tradeoff between having students explore options based on documentation (e.g., man pages) and having an instructor show particularly useful options. The man page approach fosters independence in the teach-a-person-to-fish kind of way; but it can take significant time, and some students may not find options they can use for effective analysis. A middle ground we plan to try in the future is to have the students interactively experiment with options during a relatively short portion of the class and then have a discussion about which options are particularly useful and why.
* Tools like strace involve several levels of knowledge. In order to be able to make sense of simple traces, students need to know something about system calls and their relationship to programs. They must also understand tool options for controlling the amount of information and summarizing it. These basic kinds of knowledge are a prerequisite to performing higher-level analysis, such as distinguishing normal from abnormal behavior, and associating abnormal behavior with particular kinds of malware.
* Exercises need to be carefully written to explain the purpose of the exercise and what students should and should not focus on, and to constrain the exploration of the students. Otherwise, students can spend much of their time possibly unproductively exploring blind alleys. Ideally, exercises should have questions to assess whether students have understood the high-level purpose of the exercise, plus details relevant to subsequent exercises [see appendix]
* When students have limited background knowledge (of operating systems, in this particular case), it can be helpful for an instructor to give a high-level overview of the area and explain what is and is not important about the details. Otherwise, students can spend significant time trying to understand details that are not important.
* In the context of undergraduate colleges and community colleges, it is not uncommon for security courses to be taught by instructors with limited background in the area. So instructors will be learning how to use tools one step ahead of, or alongside, the students. Exercises need to be designed so that instructors are encouraged to modify them. By thinking about how to modify the exercises, the instructors gain more confidence and a deeper understanding of the content.

## 3.2 ELF infection

[remove this section]

The ELF scenario involves detecting malware in ELF files --- the standard Linux file format used for linking and executing compiled programs. The goal of this scenario is to teach students some of the basics of reverse engineering and how to analyze complex binary files. Students are given a directory of ELF executables for some standard Linux commands (such as cat, date, ls, pwd, and whoami), some unknown number of which have been infected with malware. Students must determine which commands are infected, what the infections do, where the infected code is hidden in the ELF files, and how the infections are triggered.

In some cases, the presence of malware is evident from the behavior of the command, which might print something extra or be slower than expected. In other cases, only a tool like strace will expose that the command has been modified to perform clandestine actions in addition to its normal function. File sizes and dates may have been manipulated so that they are not reliable ways to detect which files contain malware.

This scenario requires using several analysis tools. In addition to using strace to find unexpected system calls, readelf is used to understand the structure of the ELF files, objdump is used to disassemble the binary code, a hex editor is used to examine the ELF file bytes, and gdb is used to analyze details of the dynamic program execution, such as dynamic linking of library functions and where code and data are stored in memory. Once again there is an overwhelming amount of information that students must be able to filter and summarize in order to distinguish normal from abnormal code, data, and behavior.

This scenario was tested in the same Fall 2014 class in which strace was tested. At this point of the semester, students were familiar with x86 assembly code and gdb. Over two 70-minute class sessions, students were introduced to ELF file structure, readelf, objdump, and a hex editor. In a series of demonstrations, these tools were first used to show the typical control flow through the assembly code of an uninfected ELF file for a simple program. Then they were used to analyze infected ELF files, focusing on how to find the parasite code, analyze the parasite code, and understand how control flows into and out of the parasite code. Emphasis was place on searching for suspicious system calls and branches to locations that should not have executable code. After this introduction, students were given as a homework assignment the ELF scenario (i.e. determining which of 5 ELF executables for Linux command are infected and how).

The results of this test were mixed. While a few students successfully found and correctly analyzed all the infections, and most were able to determine which files were infected, the majority flailed around with the tools and were not able to find the infections within the ELF files or determine how they were triggered. A complicating factor was that the instructor was initially unfamiliar with ELF files, readelf, and objdump, and did not have deep insight into the scenario problems until late in the process. The instructor suggests that additional materials must be developed to bring students (and instructors!) up to speed for this scenario. This underscores that instructor training and having guidance from a knowledgeable instructor are important factors for success in a complex scenario like ELF infection.

## 3.3 Recon

## [remove section]

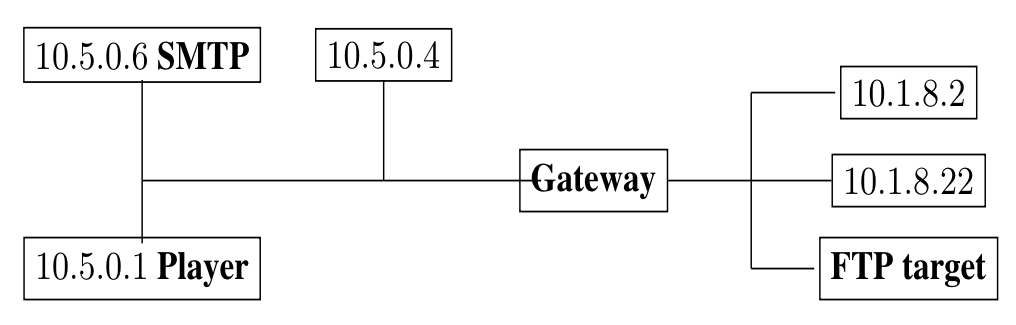
The Recon scenario was inspired by a largely similar activity from PacketWars. The goal is to find all of a small number of hosts on a potentially large network. The tool for this is nmap, a utility for finding out information about active machines on a network, such as their IP addresses, their open ports, their operating systems, etc. With incorrect choices of nmap parameters, this could take a very long time and could miss some of the active hosts. Students must understand both network protocols and how nmap interacts with those protocols in order to find a solution that is both efficient and correct.

We have now run the Recon exercise with over 120 students in 6 different settings spanning four different institutions. The scenario has evolved over time. Early participants complained that they did not have the background knowledge of networking and nmap options to succeed. We remedied this in two ways. First, we prepared for the scenario with lectures and demonstrations that explained network partitioning, TCP and UDP protocols, and nmap usage. Second, during the activity itself, we visited student teams and discussed what they were learning and problems they were encountering. With these changes, student feedback on this scenario improved dramatically. This underscores how activities involving hands-on exploration can be enhanced by balancing them with instruction and guidance.

## 3.4 ScapyHunt

[remove this section]

This scenario is about exploiting weaknesses in network protocols to understand the topology of a network and masquerade as a server on that network to intercept information from another host (Fig. 1).



**Figure 1:** The ScapyHunt Topology. Students must discover this topology by crafting packets and observing how the network reacts.

ScapyHunt has been tested twice: once with a small class of graduate students, and a second time with undergraduates in an intensive cybersecurity training course [SISMAT]. Students found it to be significantly harder than our Recon scenario. In Recon, the nmap tool does all the hard work once the appropriate options are chosen. ScapyHunt is an intrinsically more complex problem because it requires a deeper understanding of network protocols and how to subvert them (e.g., by techniques like ARP cache poisoning). Moreover, students were given far less guidance than in other scenarios. ScapyHunt provides a login prompt and little else beyond the directive “find the hidden resource in this hidden network topology.”

Although ScapyHunt was designed to be an independent discovery activity, in practice students required significant guidance during most of the interaction. For example, the students needed prompting to open a terminal and use standard network command line tools and utilities (e.g., Wireshark) to discover network information. They quickly fixated on nmap, although nmap is ultimately of little utility for this exercise. We prompted students to both “write” to the network (via ping, nmap, netcat, and some packet-crafting tools) and simultaneously “read” from the network to observe both their own actions and the actions of the entities in the software defined network (SDN).

Based on this feedback, we are designing introductory exercises to bridge the gap between what the students already know and what they need to know to succeed in this scenario. There is a constant stream of network data that students must analyze, and the software defined network (SDN) has a complex topology with multiple subnets, which students must discover and diagram. They learned the importance of using notes, drawings, and scripts to analyze complex data and a complex network topology.

# 4 Discussion and Lessons Learned

[Add analysis of student feedback]

There is a broad spectrum of hands-on exercises that are available. Some are very prescriptive and some are so open-ended that the student is not even told what to look for. Different students need different amounts of guidance, and different instructors will make different decisions about the level of scaffolding and guidance that they want to provide. We are not claiming that there is only one way to teach analysis skills and the security mindset. What is important from the perspective of designing hands-on exercises is providing flexibility to the instructor.

There are trade-offs that need to be taken into account. In theory, independent discovery is great. It has the power to engage the creative energy of the student, and can lead to deeper understanding, but that comes at a price. Independent study can require a significant amount of time from the students and the instructors. Students may miss what the goals of the exercise are and may be frustrated or lost. This could put demands on the instructor given that different students are going in different directions. We have found that the exercises that we created have significant potential for students to discover their own solutions when provided a reasonable amount of guidance.

Cyber security is a field that connects with almost all other topics in Computer Science, yet we ought not teach it only when students have all of the prerequisite knowledge. We have used our exercises in several classes at 4-year colleges that serve a diverse student population in terms of prerequisite knowledge and learning rates. As a consequence, we have learned to fill in the gaps in students’ preparation with lectures, demos, discussions, and guidance during the exercises. Our experiences and the experiences of our students have been very positive. In addition, the feedback we have received from other faculty who have tried our exercises is that their interest in cyber security has increased.

Conclusions

Appendix: questions from strace exercise

**References**

[1] Bratus , S. What hackers learn that the rest of us don’t: Notes on hacker curriculum. IEEE Security and Privacy 5 (2007), 72–75.

[2] Bratus , S., D’CU N H A , N., SPA R K S , E., A N D SM I T H , S. W. Toctou, traps, and trusted computing. In Trusted Computing-Challenges and Applications. Springer, 2008, pp. 14–32.

[3] Bratus , S., Shubina , A., and Locasto, M. E. Teaching the principles of the hacker curriculum to undergraduates. In Proceedings of the 41st ACM technical symposium on Computer science education (New York, NY, USA, 2010), SIGCSE ’10, ACM, pp. 122–126.

[4] Cappos, J., and Weiss , R. Teaching the security mindset with reference monitors. In Proceedings of the 45th ACM Technical Symposium on Computer Science Education (New York, NY, USA, 2014), SIGCSE ’14, ACM, pp. 523–528.

[5] Cone, B. D., Irvine, C. E., Thompson , M. F., and Nguyen, T. D. A video game for cyber security training and awareness. Computers & Security 26, 1 (2007), 63 – 72.

[6] Denning, T., Lerner, A., Shostack, A., and Kohno, T. Control-alt-hack: the design and evaluation of a card game for computer security awareness and education. In Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security (2013), ACM, pp. 915–928.

[7] Doupe, A., Egele, M., Caillat, B., Stringhini, G., Yakin, G., Zand, A., Cavedon, L., and Vigna , G. Hit ’em where it hurts: A live security exercise on cyber situational awareness. In Proceedings of the 27th Annual Computer Security Applications Conference (New York, NY, USA, 2011), ACSAC ’11, ACM, pp. 51–61.

[8] Du, W., and Wang, R. Seed: A suite of instructional laboratories for computer security education. J. Educ. Resour. Comput. 8 (March 2008), 3:1–3:24.

[9] Ensafi, R., Jacobi, M., and Crandall, J. R. Students Who Don’t Understand Information Flow Should Be Eaten: An Experience Paper. In Proceedings of the 5th USENIX conference on Cyber Security Experimentation and Test (CSET’12) (2012), pp. 10–10.

[10] Fryer-Biggs, Z. DoD faces cyber expert talent shortage. Computer 33, 12 (2000), 52–59.

[11] Gavas, E., Memon, N., and Britton, D. Winning cybersecurity one challenge at a time. Security & Privacy, IEEE 10, 4 (2012), 75–79.

[12] Gondree, M., and Peterson, Z. N. Valuing security by getting [d0x3d!]: Experiences with a network security board game. In Presented as part of the 6th Workshop on Cyber Security Experimen- tation and Test (Berkeley, CA, 2013), USENIX.

[13] McGettrick, A., Cassel, L. N., Dark, M., Hawthorne, E. K., and Impagliazzo, J. Toward curricular guidelines for cybersecurity. In Proceedings of the 45th ACM Technical Symposium on Computer Science Education (New York, NY, USA, 2014), SIGCSE ’14, ACM, pp. 81–82.

[14] Peterson, P. A., and Reiher, P. L. Security exercises for the online classroom with deter. Proc. of the 3rd USENIX CSET (2010).

[15] Sahami, M., Guzdial, M., McGettrick, A., and Roach, S. Setting the stage for computing curricula 2013: computer science–report from the acm/ieee-cs joint task force. In Proceedings of the 42nd ACM technical symposium on Computer science education (2011), ACM, pp. 161–162.

[16] Turner, C. F., Taylor, B., and Kaza, S. Security in computer literacy: a model for design, dissemination, and assessment. In Proceedings of the 42nd ACM technical symposium on Computer science education (New York, NY, USA, 2011), SIGCSE ’11, ACM, pp. 15–20.

[17] Vigna, G. Teaching Network Security through Live Exercises. In Proc. 3rd Ann. World Conf. Infor- mation Security Education (WISE 03) (2003), Kluwer Academic, pp. 3–18.

[18] Weiss, R., Mache, J., and Nilsen, E. Top 10 hands-on cybersecurity exercises. Journal of Computing Sciences in Colleges 29, 1 (2013), 140–147.

[19] White, G., and Nordstrom, G. Security across the Curriculum: Using Computer Security to Teach Computer Science Principles. In Proceedings of the 19th National Information Systems Security Conference (1996), NIST, pp. 483–488.

[20] Zhuang, Y., Gessiou, E., Portzer, S., Fund, F., Muhammad, M., Beschastnikh, I., and Cappos, J. Netcheck: Network diagnoses from blackbox traces. In 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI’14), USENIX.

[21] Yuan, X., Hernandez, J., Waddell, I, Chu, B, and Yu, H., Hands-on Laboratory Exercises

for Teaching Software Security, Proceedings of the 16 Colloquium th for Information Systems

Security Education, Lake Buena Vista, Florida, June 11 - 13, 2012

[22] Nestler, N., Harrison, K., Hirsch, M., and Conklin, W. A., Principles of Computer Security Lab Manual, McGraw Hill, 2015

1. http://nationalccdc.org/ [↑](#footnote-ref-2)
2. http://www.pwning.net/ [↑](#footnote-ref-3)
3. http://ctf.notsosecure.com/ [↑](#footnote-ref-4)
4. http://ictf.cs.ucsb.edu/ [↑](#footnote-ref-5)
5. https://csaw.isis.poly.edu/ [↑](#footnote-ref-6)
6. http://csr.lanl.gov/tf/ [↑](#footnote-ref-7)
7. http://packetwars.com/ [↑](#footnote-ref-8)
8. http://securitycards.cs.washington.edu/ [↑](#footnote-ref-9)
9. http://www.cisr.us/cyberciege/ [↑](#footnote-ref-10)