Using Terminal Histories to Monitor Student Progress on Hands-on Exercises

Jelena Mirkovic
sunshine@isi.edu
USC Information Sciences Institute

Aashray Aggarwal
aashraya@usc.edu
University of Southern California

David Weinman
david@weinman.com
Evergreen State College

Paul Lepe
paullepe@usc.edu
University of Southern California

Jens Mache
jmache@lclark.edu
Lewis & Clark College

Richard Weiss
weissr@evergreen.edu
Evergreen State College

ABSTRACT

Hands-on exercises are often used to improve student engagement and knowledge retention in systems, networking and cybersecurity classes. *Even when students comprehend the concepts,* they may lack the skills to complete an exercise. Teachers need effective tools to identify these problems *during* an assignment and offer targeted and timely help.

This paper explores the use of terminal histories with milestone detection to enable rapid, automated and on-going assessment of student work while performing hands-on exercises. We describe our system, called ACSLE, which monitors terminal input and output for each student, compares it with desired milestones, and produces both summaries and detailed statistics of student progress.

We analyze data from undergraduates at two colleges, as they performed several well-structured cybersecurity assignments on a network testbed. We show how ACSLE's output can help teachers identify students that struggle, understand why they struggle and offer timely help. ACSLE can also help teachers identify challenging tasks and plan class-wide interventions.

CCS CONCEPTS

• **Social and professional topics → Student assessment**; • **Applied computing → Interactive learning environments**.

KEYWORDS

testbeds, networks, learning, assessment

ACM Reference Format:
Jelena Mirkovic, Aashray Aggarwal, David Weinman, Paul Lepe, Jens Mache, and Richard Weiss. 2020. Using Terminal Histories to Monitor Student Progress on Hands-on Exercises. In *The 51st ACM Technical Symposium on Computer Science Education (SIGCSE '20), March 11–14, 2020, Portland, OR, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3328778.3366935

1 INTRODUCTION

Practical homework and lab assignments, aka "hands-on exercises," can help students internalize concepts taught in class [9, 21]. Systems, networking and cybersecurity classes use hands-on exercises to teach students specific skill sets and tools, as well as critical and adversarial thinking. Recent years have produced many exercises, some of which come with their own testbed, such as DeterLab [34], EDURange [37], NICE-challenge [25], Security Injections [13], Seattle in the classroom [7], and GENI education modules [1]. Other exercises come as virtual machines that can be run on student devices, e.g. SEED [14], SecKnitKit [5], etc.

While hands-on exercises aim to help students apply concepts taught in class, they may require specific practical skills and prerequisite knowledge that some students may lack. Further, many exercises are hosted on a specific testbeds, and students must be familiar with the testbed environment to successfully complete the exercise. These challenges can impede a student's progress on the exercise, *even when that student understands the concepts taught in class.*

Since hands-on exercises are often assigned as homework, opportunities for students to ask for help are limited. Thus, teachers have minimal insight into their class's progress on an exercise until it is submitted and graded. By that time it is too late to intervene, and the submitted material (usually the output of high-level tasks) may lack details necessary for the teacher to understand causes of poor performance. Simply put, the hands-on exercises can become an obstacle for some students, instead of enriching their knowledge.

We have designed and implemented a system, called ACSLE, for monitoring and evaluating student progress on hands-on exercises using terminal histories. ACSLE captures all terminal input and output from the students, as they interact with an exercise, and uses a set of exercise-specific milestones, to quantify student progress. ACSLE's summary statistics can help teachers identify struggling students, and analyze their actions to offer targeted and timely help *before the assignment is due.* ACSLE statistics can also help teachers identify challenging tasks, where multiple students struggle, and implement whole-class interventions, such as additional guidelines or extended office hours. ACSLE also produces detailed analysis of common error patterns, which may indicate specific gaps in student knowledge.

We survey related work in Section 2. In Section 3 we discuss hands-on exercises and outline challenges for monitoring student progress on these exercises. In Section 4 we present the ACSLE system in detail. We have implemented ACSLE on the DeterLab testbed [10, 34] and recently on the EDURange testbed [6]. We have evaluated ACSLE on four hands-on exercises, on the DeterLab testbed. These exercises were completed in three undergraduate cybersecurity classes, at two different colleges. We illustrate in Sections 5 and 6 how ACSLE statistics can help us measure individual and collective student progress on an exercise, identify challenging tasks and identify common obstacles to learning. We offer conclusions and future work directions in Section 7. All tools described in this paper are publicly available at `https://github.com/STEELISI/ACSLE`.

2 RELATED WORK

Although many researchers have studied how students learn, there have not been many attempts to write tools to record and analyze sequences of students' terminal actions. Park and Wiedenbeck [27] used help forums to examine students' questions and comments as an indication of what they did not know or had trouble with. Participants' questions were used as a measure of their understanding of the HTML5 and CSS languages, as well as feedback on pedagogy for the instructors. In our work, we used terminal data that was explicitly linked to the accomplishment of the assigned tasks. Our work could be complemented by also recording and analyzing student questions on forums.

Piech et al. [30] instrumented Eclipse to determine the sequence of intermediate programs that students wrote for an assignment. They applied unsupervised learning to cluster program snapshots and used them as abstract milestones. A hidden Markov model (HMM) was constructed to recognize these milestones. Another level of clustering was performed on the paths through the HMM. The authors took the important step of examining the paths and labeling them in human-understandable terms. The main difference between that work and ours is that we analyze system interactions via terminal, and not coding activities. We also focus on hands-on exercises with well-defined smaller tasks, that can be measured via terminal inputs and outputs.

In [15] Elkherj and Freund developed a web-based system that can send adaptive hints to struggling students, after they have made several attempts to solve a problem. The instructor must develop the hints and decide when to send them, and the system automates the sending tasks. We hope to build on the insights of this work in our intervention phase (Section 7), to develop fully automated hints.

In [18, 35] researchers measure students' compilation behaviors and correlate these with students' learning outcomes in programming courses. In [11] Carter et al. extend the model to include editing and debugging behaviors, and in [12] they also include student participation in the online social learning environment. Our work is orthogonal to these efforts. We focus on monitoring terminal inputs and outputs, to provide actionable and timely feedback to teachers about their students' progress.

Our work builds on our earlier efforts to use terminal histories to manually evaluate student learning [36]. A terminal history only logs user input, while ACSLE logs timestamp, location, input and output, and provides automated analysis of this data. Publication [19] is a two-page report of our early experiences with ACSLE.

3 HANDS-ON EXERCISES

Some computer science classes, such as systems, networking and cybersecurity, have historically used practical exercises to supplement lectures [1, 5, 6, 14, 23, 31]. These *hands-on* exercises usually specify a set of well-defined tasks that a student should accomplish. Sometimes these tasks include writing a short piece of code. But more often they include use of existing tools and programs to understand, configure or monitor applications, hosts and network elements, and to diagnose and rectify problems. These exercises enrich student understanding of concepts taught in class, and also help them learn popular systems tools.

While hands-on exercises aim to demonstrate, solidify and deepen the concepts taught in class, they rely on a student's ability to master tools and environments necessary to complete the exercise. For example, a student may need to know how to use Linux, `ssh` and `scp` utilities, how to write Bash scripts, how to tell if a process is running, etc. Some exercises can be hosted locally in a lab, while others are hosted on public testbeds [1, 6, 7, 34]. In both cases, a student must learn some specifics of the environment hosting the exercise, such as how to access it, where to store the files, etc. Lacking skills in either of these domains can impede a student's progress on the exercise and interfere with the student's learning.

Hands-on exercises are often assigned as homework, so students may have limited opportunity to ask for help. Some students may also be reluctant to ask for help, if they believe that they are the only ones struggling, and that admitting this reflects poorly on them. Thus, teachers often have a hard time assessing progress on an exercise before it is submitted and graded. By that time it is too late for interventions. Ideally, a teacher should be able to monitor student progress *as they work on the exercise.* Such monitoring would help inform a myriad of interventions, such as: (1) encouraging students to start work early, (2) identifying struggling students and offering targeted and timely help, (3) modifying or clarifying tasks that prove challenging for the entire class, (4) identifying and addressing common error patterns. While some of these goals can be achieved with a learning management system (LMS) or autograding system, none have been explored for hands-on exercises that require terminal-based student interactions.

4 ACSLE

We have developed a system, called ACSLE, whose architecture is shown in Figure 1. ACSLE automatically monitors and analyzes student progress in Linux-based environments, during well-structured hands-on exercises. We focus on exercises that mostly require terminal-based interaction. ACSLE consists of the Monitor and the Analyzer components. The Monitor component collects students' terminal input and output on all machines in the exercise, and collates it into a single log file per student. The Analyzer processes students' log files and produces individual and summary reports showing students' progress on the exercise tasks, and highlighting common mistakes.

4.1 Monitoring

Many public hands-on exercises specify tasks that require students to interact with a remote testbed environment, often accessing multiple machines simultaneously, using SSH and a terminal. We focus on monitoring student interaction in this setting, but note that our solution could be easily modified to monitor interaction in a local lab or home environment.

The Monitor component of ACSLE consists of a program, which records student input to the terminal, the output that they see, terminal identifier, username, current working directory, and the time of each interaction. We wanted to record student input and output to evaluate their progress on the exercise. We needed to record time, so we could evaluate how long the student spent on each part of the exercise. The current working directory, username and terminal identifiers provide context for the interaction, especially when the machine may be shared by multiple users, or host multiple simultaneous terminal sessions.

We investigated several tools to record the needed information: `script` [3], `history` [2], `snoopy` [16] and `ttylog` [4]. Of these, only `ttylog` could log both the input and the output of a terminal.

Our first Monitor prototype, which was used to collect data for this paper, combined `ttylog` and `snoopy`. `ttylog` captures everything that comes from a serial device `/dev/tty*` using `strace`, and thus it successfully logs all terminal input and output. `snoopy` intercepts all `exec` and `execve` calls on a Linux system, and logs the time, user, group, current directory, terminal identifier and executed command to `syslog`. During analysis we matched logs for `ttylog` and `snoopy` to arrive at the final set of records for each student and each exercise.

Challenges and Limitations. Our monitoring approach had some challenges and limitations. Our first challenge was how to reliably and accurately record terminal input and output. `ttylog` logs every keystroke and output character. This presents two general challenges: (1) reconstructing which command the user actually executed, (2) dealing with large terminal outputs. We reconstruct commands by interpreting escape sequences, such as backspace, arrow key press, control key combinations, etc. in the `ttylog` records. We also truncate long outputs to 500 lines.

Our second challenge was that matching `ttylog` and `snoopy` records did not always result in a complete and unique match. Some lines appeared only in `ttylog` but not in `snoopy` output. This sometimes happens because a command bypasses `exec`/`execve` calls, e.g., shell built-ins such as `cd`. Sometimes we could not identify the cause of missing `snoopy` lines. These limitations resulted in a loss of timestamp information on 53% of the lines in our dataset, resulting in ambiguity in the ordering of the records in the merged logs. The third challenge was tied to our implementation platform. We first implemented ACSLE on the DeterLab testbed [34]. Many students use the testbed in classes, and they usually allocate a single experiment for each exercise. We saved the logs in a directory related to the student's experiment. In some cases, a student collected the data they needed and terminated the experiment before we could collect the logs. This resulted in our dataset missing records for up to 2/3 of the students.

In our current Monitor prototype, we have addressed the above limitations. We modified `ttylog` to also record terminal identifier, current working directory, username and time of each input, and we abandoned `snoopy`. We can now accurately reconstruct timing and order of all user actions, and reliably log each action. For DeterLab, we have also relocated logs to another directory on the testbed, so that they persist after the experiment has been terminated. In

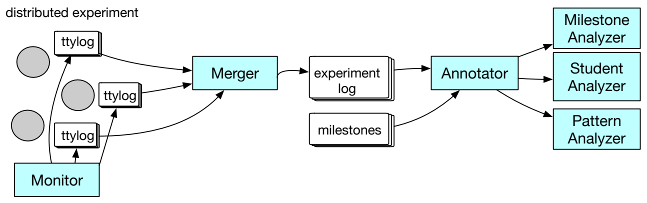


Figure 1: ACSLE System Architecture.

```
milestones :  milestone+                              node_name :  name on the testbed
 milestone :  node ',' input ',' output <EOL>          input_cmd : regular expression to match
      node :  node_name                                            with cmd line
           |  node_name '|' node                      output_cmd : regular expression to match
           |  '*'                                                  with cmd line
     input :  input_cmd
           |  input_cmd '|' input
           |  '*'                                     client,ip\s+route\s+get|ifconfig,*
    output :  output_cmd                              client,tcpdump,^((?!denied).)*$
           |  output_cmd '|' input                    *,^find|^locate,/var/log/136intro-3.jpg
           |  '*'
```

Figure 2: EBNF for milestones, and three examples.

our EduRange implementation we have also ensured that logs are stored in a permanent location.

4.2 Analyzing Student Actions

The Analyzer component of ACSLE consists of five analysis programs: the Merger, the Annotator, and the Student-, the Milestone-, and the Pattern-Analyzer. The Merger works on the set of logs generated by the students completing a given exercise. It pulls all of the log files for each student, and collates them based on the timestamp of each line. This merged log is then passed to the Annotator.

The Annotator takes as input the merged student logs and a set of *milestones* for a given exercise. A milestone is a specific, smaller learning task that a student is asked to complete, which results in a specific terminal input, output or both. For example, we can create milestones for tasks where a student needs to run a specific command, change the state of the system, or produce a specific terminal output. Tasks that require students to write code with a specific functionality are not currently addressed.

To use the Annotator, the instructor encodes milestones – homework tasks – in the special format, illustrated in Figure 2. A milestone consists of node, input and output fields. Each field can contain a wildcard, or have one or more values, separated by "|". The input and output fields can be specified using regular expressions, and should match the log on the specified node. The Figure also shows three sample milestones.

The Annotator attempts to match each line of a student's log file with each milestone. If all three parts (node, input and output) match, the line is tagged as *success* in meeting a given milestone. Otherwise, if there is at least a partial match on the input, the line is tagged as a *failed attempt* to meet the given milestone. In all other cases the line is tagged as *unrelated* to a milestone. A line may successfully match multiple milestones, or it could be a failed attempt for multiple milestones. In those cases we tag the line with multiple tags.

The Analyzer programs produce actionable summaries for the instructor. The Student-Analyzer produces a summary of each student's progress. It outputs the total time each student interacted

1. On your node, find 5 JPEG files, whose name includes the word "intro" in some form M1-5
2. On your node, find out the "vendor id" of the CPU model M6
3. On your node, find out how large disks are, and how much free space they have M7

(a) intro

1. On your node, find webserver code, compile and run it
2. In webserver.c find buffer overflow M1
3. Create your exploit program starting from `/root/submission/exploit.sh` and `/root/submission/payload` M2-3
4. Exploit the buffer to make the server crash M4
5. Fix the buffer overflow and create a patch using `diff -Naur` M5

(b) buffer

1. Create steady stream of web traffic between client and server nodes
2. Turn off SYN cookies at server M1-2
3. Find out interface leading to server from the client node M3
4. Collect tcpdump statistics on the client node on that interface M4
5. Create SYN flood between attacker and server using flooder tool with spoofing in range 1.1.2.0 and mask 255.255.255.0 M5
6. Turn on SYN cookies and repeat step 4 and 5 M6-8

(c) synflood

1. On the client machine use dig command to look up address of www.google.com M1
2. Use arp and ifconfig commands to detect interface addresses on each machine
3. On attacker, use ettercap command to create ARP poisoning M2
4. On attacker, use ettercap and dns_spoof to spoof DNS replies M3-5
5. On auth use zonesigner to sign zone google.com and reconfigure bind to use DNSSEC M6-8
6. On cache add ZSK to the list of trust anchors M9
7. On client run `dig +dnssec www.google.com A` and verify that you get a signed response M10-11

(d) dns

Figure 3: Milestones for the four exercises

exercise (class)	students		all lines			unique lines		
	total	recorded	succeeded	failed	other	succeeded	failed	other
intro (A, C)	41	38	492	1,433	9,037	251	828	2,371
buffer (A, B, C)	69	37	1,301	3,787	21,433	179	867	2,785
synflood (B, C)	40	19	766	665	2,849	318	231	706
dns (B, C)	40	17	674	1,553	1,814	99	538	488

Table 1: Statistics on the records we analyzed.

with the testbed, their input line count, the number of milestones met and the number of failed attempts. These statistics can help teachers monitor the class' progress and identify students that need help. The Milestone-Analyzer summarizes how many students attempted and met each milestone, which can help identify challenging milestones and implement interventions. The Pattern-Analyzer analyzes failed attempts to reach a milestone. It produces a graphical output showing frequently used commands, command options and argument values.This output can help teachers identify frequent student misconceptions.

4.3 Student Privacy

We treat ACSLE input and output data as sensitive information. We have taken the following steps to protect student privacy: (1) All our programs identify students by their usernames on the host machines. In DeterLab these usernames are generic. Only teachers can link such usernames to student identities, which protects students' privacy. At the end of semester the linkage between usernames and student identities is destroyed and thus student data can be further analyzed by third parties, if needed, without risk to privacy. (2) Some student input, e.g., filenames and experiment names, can inadvertently reveal student identity. We plan to anonymize this information before we release our datasets for public use. (3) Our study, reported in Section 5, was reviewed and approved by our IRB.

5 EVALUATION

ACSLE is currently implemented on the DeterLab [34] and the EDURange [6] testbeds. We report here on data collected from the DeterLab testbed.

5.1 Implementation

The Monitor component was implemented in DeterLab by applying the changes to the OS image used in the public hands-on exercises, called Ubuntu-EDU. The changes ensured that `ttylog` starts on machine startup and runs continuously, and that the logs are frequently saved to the experiment directory. Other components were Perl scripts that we ran manually on the Monitor output.

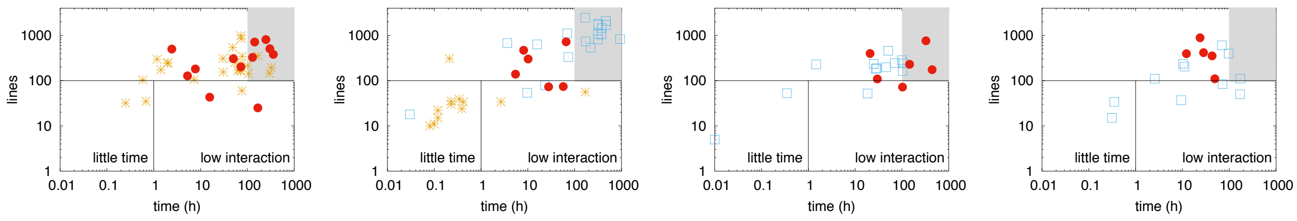
5.2 Classroom Use

ACSLE was evaluated on a set of logs collected from three undergraduate classes, taught by two authors of this paper, over the course of a year. Two of the classes already had four public DeterLab exercises as the required part of their syllabus. The third class asked for student volunteers to complete two select exercises. All students whose data was used in Section 6 were informed of the study and given the ability to opt out. No one opted out. To ensure no adverse effects from our study on students who were required to complete hands-on exercises, we analyzed the data after the classes ended and the grades were submitted.

5.3 Exercises

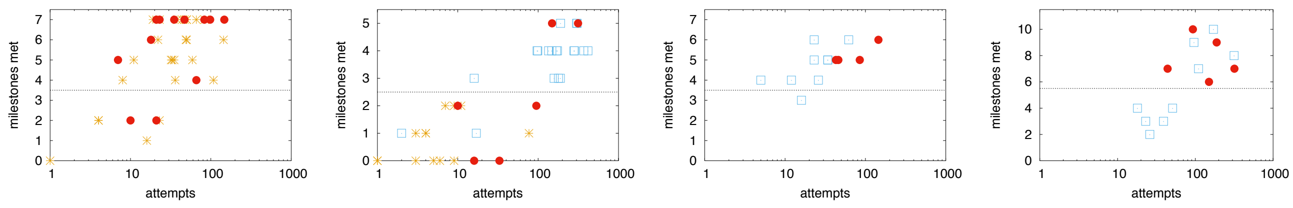
Four different hands-on exercises were completed by students. The tasks in each exercise are given in Figures 3(a) – 3(d), with tags denoting specific milestones. We do not show milestone details, since this would reveal solutions for these public exercises.

The **intro** exercise [28] teaches the students how to use Linux and the DeterLab testbed. The tasks include: (1) finding five files on the file system, (2) finding out the free space on the drive and (3) finding out the CPU model. The **buffer** exercise [29] teaches



(a) intro (b) buffer (c) synflood (d) dns

Figure 4: Time spent on an exercise vs terminal input lines.



(a) intro (b) buffer (c) synflood (d) dns

Figure 5: Unique milestones met vs number of attempts.

the students about buffer overflows [26]. The tasks include locating the buffer overflow in vulnerable code, writing an exploit for it and writing and applying a patch. The **synflood** exercise [22] teaches the students about TCP SYN flood attacks [24], and about one popular defense – SYN cookies [33]. The tasks include: (1) turning off SYN cookies, (2) performing the attack and monitoring its effect, (3) turning on SYN cookies and repeating step (2). The **dns** exercise [32] teaches the students about ARP spoofing [8] and DNS man-in-the-middle (MITM) attack [20]. The tasks include: (1) performing ARP spoofing with DNS MITM attack, (2) applying DNSSEC defense [17], and (3) verifying that DNSSEC works correctly.

6 FINDINGS

Table 1 shows the number of total students in classes A, B or C that completed an exercise, and the number of students whose records we were able to collect. We also show the number of total and unique input lines analyzed for each exercise.

Figure 4 shows the time spent on an exercise by each student on the x-axis vs number of lines a student typed on the y-axis. All graphs use log scales and same scale ranges. We observe that: (1) Students engage with an exercise sporadically, over a long time period. This sporadic engagement may hurt student learning, as it requires them to recall their last interaction, and resume it. (2) Exercises vary greatly in difficulty. The first two take much more time and more lines, than the last two (compare the number of data points in the shaded area of each figure). (3) While most students engage with the exercise, a handful do not put enough effort (lower left quadrant). Similarly, a few low-engagement students worked over a longer time period but still produced too few lines (lower right quadrant). Teachers could use our statistics to identify low-effort and low-engagement students and offer timely help.

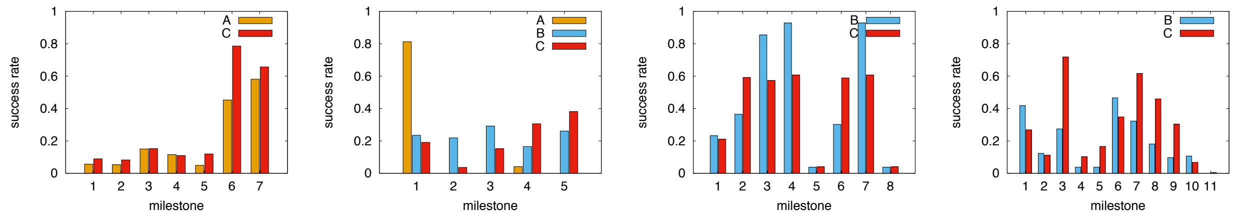
Figure 5 shows the unique milestones met (one or more times) vs total attempts to meet a milestone, for each of the four exercises. In all cases more attempts lead to more milestones met, but the relationship is far from linear. Class A did as well as class C on the "intro" exercise, but both A and C lagged behind B for the "buffer" exercise. The likely reason for this is: (1) for class A this exercise was optional, (2) for class C we had only 6 records for the "buffer" exercise. Dashed lines in each figure indicate half of the total milestones. Students whose performance is below the dashed line are struggling and would benefit from additional help.

Figure 6 shows the ratio of milestone completion over the total number of attempts, and can help the teacher identify challenging tasks for the entire class. In all exercises, tasks that required students to discover the right options and arguments for a command led to the most challenging milestones. These were milestones 1–5 in "intro", milestones 5 and 8 in "synflood", and milestones 2, 4, 5, 10 and 11 in "dns."[1]

Figures 7(a)–8(c) illustrate the output of Pattern-Analyzer for the "intro", "synflood" and "dns" exercise. Figures 7(a)–7(c) show frequently used commands in *failed attempts* on our four exercises, and their relative frequencies (on the scale 0-1). We observed a few common mistakes: (1) 16% of failed attempts in "intro" exercise used the `locate` command, which cannot find all the files, (2) in multi-node exercises –"synflood" and "dns" – some students executed commands on the wrong node – these are circled red in the Figure.

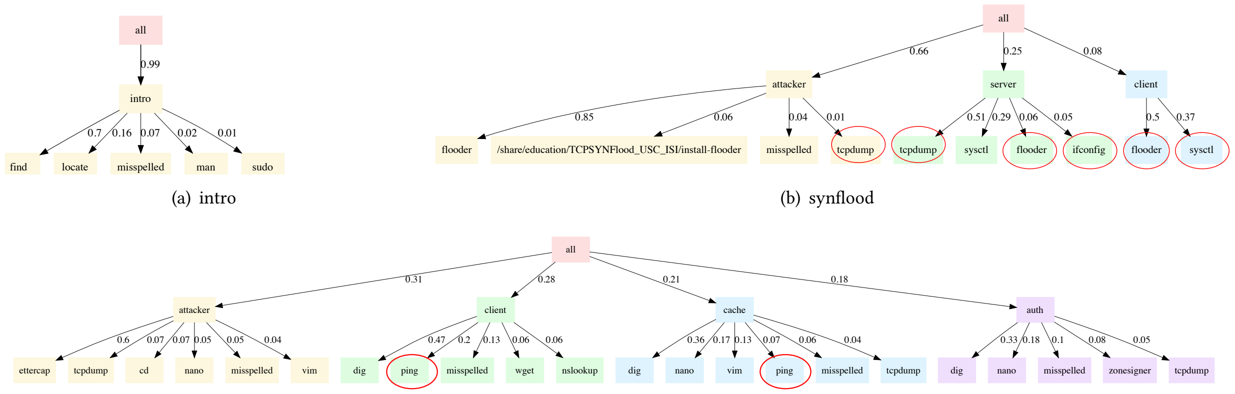
Figures 8(a)–8(c) show the frequently used options and arguments in failed attempts for the "intro", "buffer" and "dns" exercise, for each command. Numbers on the edges show the frequency of different combinations. We have circled patterns that may warrant interventions. One common error in the "intro" exercise is the use of `find -name` command, which cannot perform case-insensitive search. In only 22% of cases, students specified the correct starting point for the `find` – the root directory. In the "buffer" exercise,

[1]Class A seemingly outperformed classes B and C on milestone 1 of the "buffer" exercise, but this is because our sample for class A was small, which skewed our statistics.



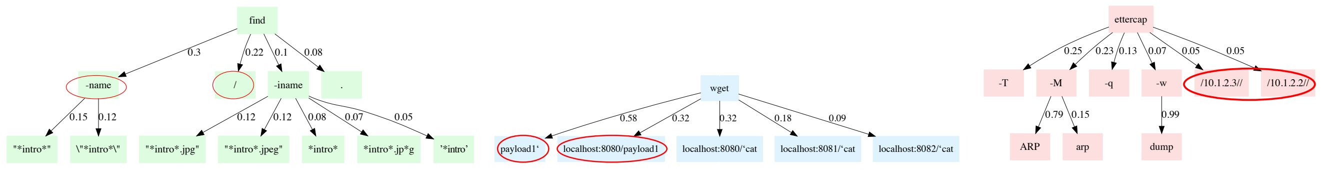
(a) intro (b) buffer (c) synflood (d) dns

Figure 6: Ratio of milestone completion over total number of attempts.



(a) intro (b) synflood (c) dns

Figure 7: Frequently used commands and their locations in failed attempts



(a) intro (b) buffer (c) dns

Figure 8: Frequent combinations of options and arguments in failed attempts (select patterns)

32% of invocations of `wget` command did not specify the correct URL, which was contained in the `payload` file. In the "dns" exercise, students fail to use `-i` option to specify on which network interface to perform ARP spoofing. They also, in some cases, misunderstood how to specify the original and the spoofed address (circled red), which should be specified as a single argument.

7 CONCLUSIONS AND FUTURE WORK

Hands-on exercises help students gain practical skills related to their learning goals, but they can also present obstacles to learning if students lack some background skills or if they are unfamiliar with the exercise environment. We have presented our approach to measure student progress on hands-on exercises using the ACSLE system. We have illustrated how our analysis tools can produce outputs to help teachers identify which students struggle and where, and inform interventions.

In our future work we will continue to evaluate ACSLE in the classroom by offering it to all of the teachers who use DeterLab and EDURange, and soliciting their and their students' feedback. These evaluations will focus on usefulness of ACSLE to teachers and students, and its effectiveness in identifying problems that warrant interventions.

In the longer term, we plan to explore how to automatically detect some struggle patterns, and to implement automated hints for students *as they work on the exercise*, via on-screen messages. We also plan on designing hints and deciding when to display them so that students are motivated to invest a fair effort into the exercise. We want students to be challenged, but we do not want them to be overwhelmed. We look forward to exploring this trade-off in our future work. We will also explore how to port ACSLE to other learning environments, such as other testbeds and university labs.

8 ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation (grants 1723717, 1723714, 1516100, 1723705 and 1516730).

REFERENCES

[1] GENI Exploring Networks of the Future. https://www.geni.net/.

[2] Man page for history command. http://man7.org/linux/man-pages/man3/history.3.html.

[3] Man page for script command. http://man7.org/linux/man-pages/man1/script.1.html.

[4] Man page for ttylog. http://manpages.ubuntu.com/manpages/xenial/man8/ttylog.8.html.

[5] Tennesee Tech University, Security Knitting Kit. http://blogs.cae.tntech.edu/secknitkit/.

[6] EDURange Testbed. http://edurange.org, (accessed Aug 30, 2019).

[7] Seattle Testbed. https://seattle.poly.edu/html/, (accessed Aug 30, 2019).

[8] B. Ballmann. *Understanding Network Hacks: Attack and Defense with Python*. Springer Publishing Company, Incorporated, 1st edition, 2016.

[9] L. ben Othmane, V. Bhuse, and L. T. Lilien. Incorporating lab experience into computer security courses. In *2013 World Congress on Computer and Information Technology (WCCIT)*, pages 1–4. IEEE, 2013.

[10] T. Benzel. The Science of Cyber-Security Experimentation: the DETER Project. In *Proceedings of the Annual Computer Security Applications Conference (ACSAC)* , 2011.

[11] A. S. Carter, C. D. Hundhausen, and O. Adesope. The Normalized Programming State Model: Predicting Student Performance in Computing Courses Based on Programming Behavior. In *Proceedings of the Eleventh Annual International Conference on International Computing Education Research*, ICER '15, pages 141–150, New York, NY, USA, 2015. ACM.

[12] A. S. Carter, C. D. Hundhausen, and O. Adesope. Blending Measures of Programming and Social Behavior into Predictive Models of Student Achievement in Early Computing Courses. *ACM Trans. Comput. Educ.*, 17(3):12:1–12:20, Aug. 2017.

[13] Cyber4All. Cybersecurity Modules: Security Injections. http://cis1.towson.edu/~cssecinj/.

[14] W. Du and R. Wang. SEED: A suite of instructional laboratories for computer security education. *Journal on Educational Resources in Computing (JERIC)*, 8(1):3, 2008.

[15] M. Elkherj and Y. Freund. A System for Sending the Right Hint at the Right Time. In *Proceedings of the First ACM Conference on Learning @ Scale Conference*, L@S '14, pages 219–220, New York, NY, USA, 2014. ACM.

[16] M. A. Eriksen and M. Baker. Snoopy Logger. https://github.com/a2o/snoopy.

[17] ICANN.org. DNSSEC – What Is It and Why Is It Important? https://www.icann.org/resources/pages/dnssec-what-is-it-why-important-2019-03-05-en.

[18] M. C. Jadud. Methods and Tools for Exploring Novice Compilation Behaviour. In *Proceedings of the Second International Workshop on Computing Education Research*, ICER '06, pages 73–84, New York, NY, USA, 2006. ACM.

[19] P. Lepe, A. Aggarwal, J. Mirkovic, J. Mache, R. Weiss, and D. Weinmann. Measuring Student Learning On Network Testbeds. In *Midscale Education and Research Infrastructure and Tools Workshop*, 2019.

[20] M. McLafferty, W. Levesque, and A. Salmon. *Applied Network Security*. Packt Publishing, 1st edition, 2017.

[21] M. Mendicino, L. Razzaq, and N. T. Heffernan. A comparison of traditional homework to computer-supported homework. *Journal of Research on Technology in Education*, 41(3):331–359, 2009.

[22] J. Mirkovic. TCP SYN Flood. https://www.isi.deterlab.net/file.php?file=/share/shared/TCPSYNFloodexercise.

[23] J. Mirkovic and T. Benzel. Teaching Cybersecurity with DeterLab. *EEE Security and Privacy Magazine*, 10(1):73–76, January/February 2012.

[24] J. Mirkovic, S. Dietrich, D. Dittrich, and P. Reiher. *Internet Denial of Service: Attack and Defense Mechanisms (Radia Perlman Computer Networking and Security)*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2004.

[25] V. Nestler. NICE challenge project. https://www.nice-challenge.com/, 2019.

[26] A. One. Smashing the Stack for Fun and Profit. http://www-inst.eecs.berkeley.edu/~cs161/fa08/papers/stack_smashing.pdf.

[27] T. H. Park and S. Wiedenbeck. Learning Web Development: Challenges at an Earlier Stage of Computing Education. In *Proceedings of the Seventh International Workshop on Computing Education Research*, ICER '11, pages 125–132, New York, NY, USA, 2011. ACM.

[28] P. A. H. Peterson. Linux and DeterLab Intro. https://www.isi.deterlab.net/file.php?file=/share/shared/LinuxandDeterLabintro/.

[29] P. A. H. Peterson and P. Reiher. Buffer Overflows. https://www.isi.deterlab.net/file.php?file=/share/shared/BufferOverflows-UCLA.

[30] C. Piech, M. Sahami, D. Koller, S. Cooper, and P. Blikstein. Modeling How Students Learn to Program. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education*, SIGCSE '12, pages 153–160, New York, NY, USA, 2012. ACM.

[31] SeattleTestbed. Seattle in the Classroom. https://seattle.cs.washington.edu/html/education.html.

[32] B. Sheridan and D. Massey. DNS and Man-in-the-Middle Attacks. https://www.isi.deterlab.net/file.php?file=/share/shared/DNSmaninthemiddleattack.

[33] C. Smith and A. Matrawy. Comparison of operating system implementations of SYN flood defenses (cookies). In *2008 24th Biennial Symposium on Communications*, pages 243–246. IEEE, 2008.

[34] USC/ISI and U. Berkeley. DeterLab testbed. http://isi.deterlab.net.

[35] C. Watson, F. W. B. Li, and J. L. Godwin. Predicting Performance in an Introductory Programming Course by Logging and Analyzing Student Programming Behavior. In *Proceedings of the 2013 IEEE 13th International Conference on Advanced Learning Technologies*, ICALT '13, pages 319–323, Washington, DC, USA, 2013. IEEE Computer Society.

[36] R. Weiss, M. E. Locasto, and J. Mache. A Reflective Approach to Assessing Student Performance in Cybersecurity Exercises. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, SIGCSE '16, pages 597–602, New York, NY, USA, 2016. ACM.

[37] R. S. Weiss, S. Boesen, J. F. Sullivan, M. E. Locasto, J. Mache, and E. Nilsen. Teaching Cybersecurity Analysis Skills in the Cloud. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, SIGCSE '15, pages 332–337, New York, NY, USA, 2015. ACM.