TEACHING CYBERSECURITY AND PYTHON PROGRAMMING IN A 5-DAY SUMMER CAMP

Joshua Eckroth
Math/CS Department
Stetson University
jeckroth@stetson.edu

ABSTRACT

This report documents our experiences teaching cybersecurity and Python programming during a 5-day, 25-hour summer camp at Stetson University in 2017. The camp participants included rising high school juniors, seniors, and entering college freshmen. The camp curriculum covered the fundamentals of cybersecurity including networking, encryption, password management, penetration testing, and the Tor proxy server; and the fundamentals of programming in Python including variables, input/output, conditionals, loops, functions, lists, and dictionaries. The curriculum was designed for students with no prior experience in cybersecurity or programming. This report details our curriculum and offers evidence of the effectiveness of the camp based on pre- and post-surveys.

INTRODUCTION

This report documents our experience teaching our University's first computer science summer camp aimed at junior- and senior-level high school students and entering college freshmen. Our long-term goal is to create a self-sustaining yearly summer camp in which student learning is measurably increased. In this way, our goal aligns with Georgia Tech's [3], who have reported financial details of their self-sustaining camps and student learning outcomes. We also wish to increase recruitment from local high schools to our University, a small private liberal arts college. To encourage recruitment, each participant in our first camp received a $10,000 college scholarship, spread over four years, to discount our University's tuition costs.

We designed our summer camp around two learning objectives: we wanted to expose students to a wide range of cybersecurity topics and practices; and we wanted students to acquire basic Python programming skills. In order to interleave our two topics, cybersecurity and Python programming, cybersecurity topics were demonstrated in many cases with Python code and programming exercises mostly centered around cybersecurity topics. We covered a wide range of cybersecurity topics as a preview of many of the topics they will see later in college and as professionals. For example, a student might later encounter topics like random number generators, cryptographic hashing, or the dark web. We want the students to recall our summer camp topics like entropy generation, good hash algorithms vs. bad hash algorithms, and the rendezvous protocol for Tor hidden services. Rather than wait until an upper-level network security course, we designed our curriculum so that camp participants would be familiar, though not necessarily proficient, with advanced topics from their first day of college. The camp also highlighted our innovative upper-division cybersecurity course [5] with a presentation by Dr. Plante on the final day of camp.

We chose the Python language because few lines of code are required to construct basic programs, and the code is relatively easy to read. Grandell et al. [4] noted that Python is a good choice for introductory computer science courses in high schools because its simple syntax "significantly reduced the notational overhead [...] and thus left more time for actual coding." Python is also a good choice for our camp because it has several libraries for networking and encryption, such as the sockets library and PyCrypto.

Our camp spanned five days, 9am-3pm with a one-hour lunch break, totaling 25 hours of classroom time. Our daily curriculum is detailed below. In order to measure learning outcomes in both cybersecurity and Python programming topics, we conducted a pre- and post-survey (with identical questions) at the start and end of the camp. Below, we describe our research methodology and outcomes, and conclude with discussion of these outcomes and lessons learned.

CURRICULUM

Our goal when designing the curriculum was to seamlessly mix cybersecurity and programming topics while progressing along a "story." The story may be quickly outlined as follows: overview of the topics and definitions; machine-to-machine network connection and message sending/receiving; message routing to connect to more distant machines across multiple hops; adding domain names to identify machines by names rather than numbers; hiding secret messages with encryption; choosing good passwords and keeping track of all of one's randomly generated passwords; hiding the source of messages using a proxy; hiding both the source and destination of messages using proxied hidden services. Throughout this story, we touched on the ethics of learning these topics and techniques by repeatedly emphasizing that one cannot protect one's self without knowing how both the attacks and defenses work.

This story is just one of several possible paths through the diverse topics of cybersecurity. For example, a different camp may focus on the web, social networks, and privacy, while another camp may focus on secure coding. In any case, we believe it is important to follow a progression of increasing sophistication and usefulness, e.g., encrypting messages after learning how to send messages.

Throughout the curriculum, we injected Python programming demonstrations and exercises. Students wrote programs (either by copying a live demonstration or during independent exercises) that send and receive messages over the internet, at first using IP addresses, and later using a custom domain name server. These messages are later encrypted, and the client's location was eventually hidden using the Tor proxy. The programs were limited in their user interface, requiring keyboard input on a text console. We chose to use text interfaces for coding simplicity, but we suspect students would appreciate more visually appealing interfaces to their programs, such as a website or mobile app.

Programming exercises played an important role in our camp, even though it only spanned five days. We wanted students to engage with the material and feel empowered by building programs on their own; in other words, we wanted students to experience high self-efficacy. Aritajati et al. [1] found that higher student self-efficacy translates into higher engagement with computer science subjects. They conducted several summer camps for students age 13 to 17 and found that students' self-efficacy increased when the camp included programming exercises, flexible pacing, and opportunities to explore further outside of the camp classroom. Our camp included programming exercises and optional homework for exploring a subject outside of the classroom. We were limited in our ability to support flexible pacing due to the short duration of our camp and the wide range of topics we wished to cover. To support students during the programming exercises, the professor and a student assistant (a senior computer science major) walked around the room and helped individual students complete their tasks.

Materials

Each student was provided a unique Google Cloud Platform virtual machine (VM) for Linux access. We used virtual machines to ensure every student had an identical platform with which to write code and run security tools, and to ensure outside access to their VM was not blocked by firewall rules that are common in a University environment. For our purposes, the cheapest and lowest-performance VM offered by Google sufficed. After the camp, students are able to set up their own equivalent VM under their own Google account for free (Google's "free tier"). Google's cost estimator indicates a collection of virtual machines under one account (such as a professor's account) with 24/7 access would cost $0.89 per VM for one week of usage.

The VMs ran Debian Linux and Python 3.5. We selected Linux as a common platform due to the availability of security tools (e.g., SSH, GPG, nmap, Tor, hashcat, etc.) and easy install process (e.g., `sudo apt-get install tor`). We did not expect any student to have prior Linux experience, so we demonstrated the relevant Linux commands as needed throughout the week. Students connected to their VM via SSH using either PuTTY on Windows or SSH on Mac OS X. Some students brought laptops while most used machines in the classroom. FileZilla was used to transfer files to and from the VM. Most students used nano on the VM to edit their code. We found that whatever the professor demonstrated during lecture and live-coding sessions (nano in this case), the students replicated.

Notes and code typed during lecture or live demonstrations by the professor were quickly published to a website. This allowed students to feel comfortable just watching rather than typing and trying to keep up. However, we found many students did type code as it was demonstrated. We also found that students routinely referred to the website during independent programming exercises to review techniques from prior examples. Several students requested that the website remain online after the camp. We used free GitHub Pages hosting for the website.

No other materials were used besides the virtual machines, lab computers, and website with notes. We did not use a text book or special hardware. These features make our approach inexpensive, flexible, and easily replicated.

Day 1

Our goals for Day 1 included: conducting the pre-survey, providing each student with access to their unique cloud virtual machine, and engaging with some overview topics in cybersecurity and Python programming. After some introductory remarks, we asked the students to complete a pre-survey of their Python programming and cybersecurity knowledge. This survey took about 30 minutes. Next, we provided each student their login and IP address for a unique Google Cloud Platform virtual machine instance. We showed the students how to use PuTTY on Windows and SSH on Mac OS X to connect to their VM and FileZilla to transfer files to and from their VM.

After these administrative details, we introduced simple Python programming. Variables, data types (integers, floats, and strings), printing, and user input were demonstrated in live coding examples. Next, we demonstrated the use of conditionals (if, elif, else) and Boolean expressions. A Zodiac sign calculator was live-coded to demonstrate a chain of if-elif-else conditionals. The Zodiac calculator asks for the user's birth day and month and determines their Zodiac sign.

After these brief demonstrations of Python programming, we challenged students to create a number-guessing game. The program generates a random number and asks the user to guess the number. If the user's guess is too high or too low, an appropriate message is displayed the user is asked again (up to three times). Before asking the students to start this task, we demonstrated Python's random number generator library. While students completed the task, the professor and student assistant attended to individual students to ensure all students were on the right track.

In the last hour of the day, we engaged the students in a discussion of cybersecurity topics. We defined popular terms such as white hat, black hat, phishing, ransomware, etc. A wide variety of tools and techniques in cybersecurity were defined and grouped into categories: discovery (e.g., phishing, port scanning, key logging), attacks (e.g., common vulnerabilities, denial of service, rootkits), and defenses (e.g., firewalls, software patching, strong passwords, encryption).

Students were asked to consider working on optional homework, consisting of two tasks: (1) update their number guessing game to support infinitely-many guesses (with a loop), and (2) find a news article about a recent cybersecurity incident and prepare to explain the incident to the class.

Day 2

At the beginning of each day starting on Day 2, we reviewed or reflected back on an activity from the prior day. We believed it was important to develop continuity across each day of the camp so that students felt that what they learned each day would be important later and not simply isolated knowledge. To this end, we began Day 2 reviewing solutions to the number-guessing game, including the addition of loops to support infinitely-many guesses. Loops were not introduced on Day 1; instead, they were introduced in the context of the number-guessing game on Day 2 in order to solve a clear need. The transition from conditionals to loops was trivial (in code, at least) and exposed the natural connection between these two programming concepts.

Next, we discussed the design of the internet and packet routing. Using a slide deck, we looked at the history of the growth of the internet, explained the roles of TCP and IP, showed the distinction between non-routable IP addresses like 192.168.0.0/24 and routable IP addresses, and demonstrated a multi-hop route with the traceroute tool. We further explained the role of ports and showed a list of default ports like 80 for web traffic and 22 for SSH traffic.

With these fundamental concepts in mind, we showed students how to write Python code to open a socket on a particular port and send a message to another socket on another machine identified by its IP address and port. We expected that the message sending code was too sophisticated for detailed examination in the time available for this camp, so we quickly moved the code into two functions (`send_message` and `receive_message`) and took advantage of this opportunity to explain the role of functions. We live-coded the solution and asked for student input such as which messages to send and to whom. Students typed along and eventually were able to send each other messages using their own code. Next, we introduced Python lists. The client/server was expanded to store prior messages in a list and send this list to a client when the client connected. Thus, the client could show other users' messages as a kind of bulletin board system (BBS).

At the end of the day, we briefly discussed domain names. Students were asked to optionally update their BBS client to connect to a user-provided domain name instead of a fixed IP address.

Day 3

At the start of Day 3, we reviewed a solution to the prior day's optional homework, namely, adding domain name support to the BBS client. This review naturally led into a discussion of the domain name system (DNS) and to the development of our own "mini-DNS." Students were individually asked to choose a unique domain name for their VM's IP address. We then live-coded the development of a mini-DNS server by adapting the simple message server developed in Day 1. The mini-DNS server required key-value pairs associating domain names (keys) to IP addresses (values) and thus served as an introduction to Python dictionaries. Once the mini-DNS server was coded, students were asked to create a mini-DNS client to send a domain name and receive an IP address. Then students were shown how to integrate the mini-DNS client functionality into their BBS client so that their new mini-DNS domain names could be used to send messages to each other in the room.

The latter half of Day 3 covered a new topic: hashing and encryption. We first discussed the distinction between hashing and encryption by focusing on the irreversibility of a hash. Our discussion of the use of hashing focused mostly on password storage. Next we discussed password complexity and demonstrated high-performance password cracking with best-in-class GPUs using the hashcat tool. We strongly emphasized that passwords should be randomly generated and never reused.

Optional homework included added reverse-DNS support to the mini-DNS server (i.e., send the mini-DNS server an IP, get a domain back), and cracking a list of hashed passwords using hashcat or a similar tool.

Day 4

Day 4 began by reviewing a solution to the reverse-DNS optional homework. We then demonstrated how to update the BBS server to use reverse-DNS to attach the domain name to each message so we have a log of who posted which messages.

Recall that at the end of Day 3, we demonstrated how to crack password hashes. We emphasized that secure passwords are random and unique. Hence, some kind of password manager would be necessary to keep track of one's many passwords. With this motivation, we asked students to write code that generates a random password. This challenge required the use of the random number generator, string concatenation, and a loop. After giving time for students to complete this task, we live-coded a password manager that was able to generate a password and save that password with a key representing a login. Python's dictionaries were used for this purpose. We further introduced Python's “pickle” module to save and restore data structures to disk.

After the development of a password manager, we explained asymmetric encryption with public/private key pairs. The students were shown how to generate PGP keys using GPG software. Since PGP keys require a large number of random bits, we explored the role of entropy in random number generators and installed a program known as HAVEGE [6] to increase our VM's entropy, without which the VMs did not have enough entropy to successfully generate PGP keys. We engaged in a “key-signing party” in which some students physically verified each other's public keys and then signed their partner's public keys with their own private keys, thus building a web of trust. We then demonstrated sending encrypted and/or signed messages using PGP technology, and asked some students to send each other such messages and verify their authenticity. Finally, we discussed how SSL/TLS encryption on websites uses public/private key pairs in addition to certificates signed by certificate authorities.

The latter part of Day 4 was spent on a tour of our university's IT server room. The IT staff kindly answered numerous questions that we developed with the students previously in the day. In the post-survey, several students indicated the IT tour was the best part of the camp. We did not discuss any optional homework for Day 4 because this tour occurred at the end of the day.

Day 5

The first half of Day 5 began by discussing the role of proxies, e.g., to offer many backend services from a single unified IP address and port. Then we transitioned into a discussion of Tor. Nearly every student already knew about Tor, although fewer students knew how it worked. Using a slide deck, we showed how Tor routes data through multiple hops, and that the communication between each hop is encrypted, so each hop only knows about the prior and next hop. By requiring at least three hops, Tor ensures the destination cannot identify the source, and any one machine among the multiple hops is unaware of either the original source or the destination or both. Next, each student installed Tor on their VM and updated their BBS client to route traffic through Tor before connecting to the BBS server. We added GeoIP tracking to the BBS server to show that connections were arriving from around the world, even though all the connections originated in the classroom (or, more accurately, Google's VM cluster). Finally, using a slide deck, we showed how the dark web works by further obscuring the service behind at least three hops, just as the client is obscured. Nearly all students had also heard about the dark web (a.k.a. deep web) but we did not demonstrate connections to any dark web sites. However, we showed that The New York Times offers a dark web service for submitting news tips, in order to emphasize the potentially positive role of Tor and the dark web.

In the latter half of Day 5, we first asked students to complete the post-survey, and then reviewed all the Python and cybersecurity topics covered in the week. Day 5 completed with two invited talks by Dr. Plante, who teaches our upper-division cybersecurity course, and a former student who now works as a security professional for one of the top Silicon Valley companies.

RESEARCH METHODOLOGY

We had two goals for our research agenda: (1) identify the camp students' prior knowledge about Python and cybersecurity; and (2) determine whether our summer camp improved their knowledge about Python and cybersecurity. We note that we are unable to expand goal (1) to identify prior knowledge of the "average" high school student as we did not conduct a random sample of students across a wide demographic. Rather, we are only able to make claims about prior knowledge of our camp students.

In order to meet our research goals, we developed a pre- and post-survey covering introductory topics in Python programming and cybersecurity. The pre- and post-survey questions and answers were equivalent. The answers were all multiple choice with three possible choices with only one correct answer. We wanted the survey to be completed in about 30 minutes and not feel like an extensive exam. The surveys had twelve Python questions and seventeen cybersecurity questions. We included additional free-answer questions to gather information about the students' grade level, prior experience with programming, and expected college major.

The pre-survey was given to the students just after introductory remarks on the first day of the camp. We intentionally scheduled the survey at this time, before reviewing the camp syllabus or curriculum, in order to avoid influencing the students' answers. The post-survey was given to the students on the last day before a final review of all the material covered in the camp. We did not want to remind students what they learned during the week before they completed the post-survey.

In the following section, we detail our findings through these surveys.

OUTCOMES

Our summer camp hosted 33 students, about 1/3 female and 2/3 male. Eighteen (55%) will be college freshmen in the Fall following the camp, eight (24%) will be high school seniors, and seven (21%) will be high school juniors. Fifteen (45%) planned to major in computer science in college, while the rest planned a different major or were undecided. The percentage of computer science majors is lower than we expected for a computer science-focused camp, but we believe the $10k scholarship offering likely enticed a broad range of students. Based on an

Prior Experience	Python Score	
	Mean	Std dev.
None	44%	11%
Some	40%	18%
Lots	68%	11%

Scores	Python Mean	Python Std dev.	Cyber. Mean	Cyber. Std dev.
Pre-survey	49%	18%	53%	15%
Post-survey	63%*	19%	70%*	14%

Table 1 (Left): Pre-survey scores for the Python programming component.
Table 2 (Right): Pre- and post-survey scores for both components. An asterisk (*) indicates statistical significance in the difference of scores in pre- and post-surveys.

open-ended question about prior programming experience, we subjectively categorized students' prior experience in programming into three categories: none, some, and lots. Thirteen students (40%) stated they had no prior experience with programming before the camp; eleven (33%) had some prior experience; nine (27%) had lots of prior experience. A student's intention to major in computer science relates to their prior experience: among students who do not intend to major in computer science, 53% had no prior programming experience; among students who intend to major in computer science, only 27% had no prior programming experience. Comparing pre- and post-survey responses, we find no change in students' intention to major in computer science after completion of the camp.

The pre- and post-survey questions each had three multiple choice answers with only one correct answer. Thus, random guessing should yield a score of 33%, on average. A summary of pre-survey scores for the Python is shown in Table 1. The table segregates scores based on the student's prior programming experience (none, some, lots). In order to show the impact of the summer camp, a comparison of the pre- and post-survey scores for the Python and cybersecurity questions can be seen in Table 2. Using the Student's t-test with paired samples, we determined that the increase in scores is statistically significant, $p < 0.01$.

DISCUSSION

It is clear from Table 2 that student scores improved in the post-survey compared to the pre-survey. Naturally, we attribute this increase to the curriculum and experience during the camp. There are many other factors that may impact student learning, even during such a short time period. The same curriculum taught by different teachers may have different impacts. Some students may practice with the material and engage with the optional homework more than other students. And some students may have reviewed material during the last day of camp, knowing the post-survey would occur on that day. Finally, we cannot report how well students retain the knowledge that we see represented in their post-survey scores. After only a week of camp, it is possible that students retained enough knowledge in short-term memory to improve their scores on the post-survey, but might forget what they learned a short time later. Even with all these possible confounding factors, we find that the pre-/post-survey analysis confirms the adequacy of our curriculum.

Another observation not reflected in the programming and cybersecurity portions of the survey is worth noting. We asked students what they least enjoyed about the camp. Seven students (21%) stated that the material was covered too fast. The curriculum we detailed above covers many topics and does not allow much more than 30-min to an hour for students to practice with a programming concept. It is clear some students need more time to fully grasp a concept and develop a working program.

The Python programming topics in the curriculum allowed students the opportunity to practice with the concepts. For example, after introducing conditionals, students were asked to program a guess-the-number game. On the other hand, the curriculum does not include many opportunities for students to practice with cybersecurity topics. Given the short duration of the camp, we felt that there was insufficient time in some cases to present a cybersecurity topic, e.g., penetration testing using Metasploit, as well as dedicate time for students to practice with Metasploit given the complexity of the software and range of possible exploits.

Our curriculum included optional homework assignments, but we found that students almost never attempted the assignments outside of class. Some students completed some assignments or worked on other programming projects during class (e.g., after they finished the guess-the-number challenge). We cannot say for certain if students practiced with other coding assignments outside of class during the camp week. However, it is worth noting that the optional homework assignments were either insufficiently enticing or too challenging for students to engage with them outside of class.

During the camp, we wrote notes as we lectured or demonstrated techniques with "live-coding." Every example Python program was also added to the notes. These notes were posted to a public website each day.[1] We observed that students frequently visited this website during the camp to copy working code developed during lecture before extending it, and to review various facts, definitions, etc. that were mentioned in class. Our website logs indicate that students frequently visited the site during the camp (as confirmed by our observations), but rarely after class ended each day. Furthermore, the website receives virtually no visitors since the camp ended. The website adequately served its purpose during the camp, but we believe there may be opportunities to increase the usefulness of a website beyond the short camp duration.

FUTURE WORK

While we consider the camp a success, there are several opportunities for improving future iterations of the camp. Several students reported that the camp was "too fast," and we noted above that some topics did not receive much independent programming time. Thus, we recommend that the curriculum be refined in order to focus on fewer topics and allow more time for programming

[1] http://2017.csss.artifice.cc

exercises that reinforce those topics. For example, it might be useful to allow students more independent time developing a password manager program in lieu of discussing PGP public/private key pairs and key signing.

Our camp did not include any competitive exercises. These exercises, like "capture the flag" competitions, are common in cybersecurity education. Deylami, et al. [2] report positive student learning outcomes by introducing cybersecurity competitions in existing secondary school courses. We chose not to include such competitions due to time constraints and an uncertainty about whether some students will feel at a disadvantage because they do not come to the camp with as much prior experience as others. We expect there are ways to develop competitions in which students are segregated according to their prior experience.

CONCLUSIONS

In this report, we detailed a five-day cybersecurity and Python programming Summer camp aimed at high school juniors, seniors, and entering college freshmen. We explained our motivations and curriculum, and demonstrated positive and significant student learning. We discussed possible variations to the curriculum that focus on different aspects of cybersecurity, but nevertheless expect that the curriculum as detailed is easily adopted by other institutions and high schools.

Due to the success of our first iteration of this camp, we are planning a 2018 version. We expect that cybersecurity will continue to be an important topic that students, even those will little or no prior experience in computer science topics, will find interesting. Cybersecurity serves as a good context for introducing fundamental computer science topics and computer programming.

REFERENCES

[1] Aritajati, C., Rosson, M., Pena, J., Cinque, D., Segura, A. A socio-cognitive analysis of summer camp outcomes and experiences, *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, 581-586, 2015.
[2] Deylami, H.M., Mohaghegh, M., Sarrafzadeh, A., McCauley, M., Ardekani, I.T., Kingston, T. Capture the talent: Secondary school education with cyber security education, *International Journal in Foundations of Computer Science & Technology,* 5 (6), 55-66, 2015.
[3] Ericson, B., McKlin, T. Effective and sustainable computing summer camps. *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education*, 289-294, 2012.
[4] Grandell, L., Peltomäki, M., Back, R.-J., Salakoski, T. Why complicate things?: Introducing programming in high school using Python. *Proceedings of the 8th Australian Conference on Computing Education*, 52, 71-80, 2006.
[5] Plante, D., Penney, B. Teaching security to undergraduates as a faculty-administration collaborative endeavor. *Proceedings of the Information Systems Education Conference*, 32, 269-278, 2015.
[6] Seznec, A., Sendrier, N. HAVEGE: A user-level software heuristic for generating empirically strong random numbers. *ACM Transactions on Modeling and Computer Simulation*, 13 (4), 334-346, 2003.