Teaching Software Supply Chain Security

Zhouzhou Li
Zli2@semo.edu

Juefei Yuan
jyan@semo.edu

Mario Alberto Garcia
mgarcia@semo.edu

Southeast Missouri State University
Cape Girardeau, MO

Abstract

Modern software often incorporates open-source and third-party code, inheriting their vulnerabilities and reducing developers' control over security and patches. Notable supply chain attacks, such as the SolarWinds incident, the Log4j exploit, and the outage caused by CrowdStrike's update, have highlighted the severe consequences of compromised software supply chains. In response, the cybersecurity faculty of Southeast Missouri State University introduced a new Software Supply Chain Security course in Summer 2023, aimed at enhancing students' practical skills and job market readiness in facing the challenges from software supply chain vulnerabilities. The course is designed to align with the NCAE-CD Knowledge Units and respect the classic Bloom's Taxonomy. And six learning outcomes, such as "understand the dependencies between the software and the 3rd-party software", "select and integrate appropriate 3rd-party software", and "verify, debug, and update or replace the integrated 3rd-party software" are identified. Attended by 12 Cybersecurity master's students, the course received positive preliminary feedback on its teaching effectiveness, with the majority of students praising the quality of the course materials.

Keywords: Software Supply Chain Security, Cybersecurity Education, Open-source Vulnerabilities, Third-party Code Security, Supply Chain Attacks, SolarWinds Incident, Log4j Exploit, Practical Skills Development, Job Market Readiness, NCAE-CD Knowledge Units.

1. WHY TEACHING IT AND WHY PROPOSING A COURSE

Software Supply Chain Security is important
The software supply chain is made up of everything and everyone that touches the software in its Software Development Life Cycle (SDLC). It includes networks of information about the software, like the components, the people who wrote them, the sources they come from, and all vulnerabilities associated with (DEF, 2024). Most software today is not developed from scratch – it reuses A significant amount of open-source code and/or combines third-party code. Therefore, it inherits all the vulnerabilities from the reused code. Unfortunately, the software developers have less control over the open-source and third-party code as well as their patches. Software supply chain security becomes a big challenge to today's software.

Three examples of the software supply chain attacks are the recent SolarWinds incident, the famous Log4j exploitation, and the outages caused by CrowdStrike's updates. The consequences were catastrophic. The

CrowdStrike case was particular ironic because the third-party software provided by CrowdStrike was for security protection.

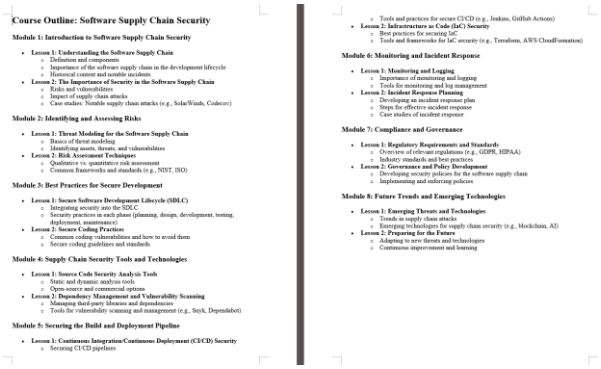
In response to the rise of the Software Supply Chain Security issues, in 2021, the president of the United States highlighted the importance of software supply chains and security with two White House Executive Orders: supply chains and cybersecurity. In 2022, the National Centers of Academic Excellence in Cybersecurity (NCAE-C) Curriculum Task Force started to seek proposals for developing post-secondary educational materials in the software supply chain security area. In 2023, they are still looking for it.

To expose the students to this new practical Cybersecurity sub-area, develop their hands-on skills in an ever-changing discipline, and increase their competency in the job market, it is urgent to develop the course material for Software Supply Chain Security. Southeast Missouri State University, as the national CAE-CD and MO state designated institution, and its Cybersecurity faculty actively take the responsibility to adapt to the new technical trends in Cybersecurity education and training by developing a new Software Supply Chain Security course. The first experimental class was delivered in Summer 2023 as a 6-week summer course with 12 Cybersecurity master students attending. The newly designed course materials included an up-to-date syllabus, six Learning Outcomes, six lecture slides, five assignments and five labs. The preliminary teaching effectiveness evaluation is very positive.

Software Supply Chain Security Education is new

Because the Software Supply Chain Security is quite a new sub area of Cybersecurity, it is hard to find an existing textbook or tutorial for reference when the authors were developing the course material. However, there are several hints we can get from the existing textbooks or other documents talking about the general supply chain, and from the "brainstorming" conducted by ChatGPT. We will address the prior later, here we just discuss the "answers" from ChatGPT.

In Figure 1, ChatGPT answered the question: can you develop a course talking about software supply chain security? Obviously, ChatGPT thought SSCS is a component of software development life cycle, which is good. However, from the eight modules proposed by ChatGPT, we did not see sufficient technical topics unique to software supply chain security. We can find similar modules in general Secure Software Development courses. The answer did not well address the core question when a faculty try to develop a new course: what is the unique content in the new SSCS course? This was why we did not follow ChatGPT's recommendations.



Course Outline: Software Supply Chain Security

Module 1: Introduction to Software Supply Chain Security

- **Lesson 1: Understanding the Software Supply Chain**
  - Definition and components
  - Importance of the software supply chain in the development lifecycle
  - Historical context and notable incidents
- **Lesson 2: The Importance of Security in the Software Supply Chain**
  - Risks and vulnerabilities
  - Impact of supply chain attacks
  - Case studies: Notable supply chain attacks (e.g., SolarWinds, Codecov)

Module 2: Identifying and Assessing Risks

- **Lesson 1: Threat Modeling for the Software Supply Chain**
  - Basics of threat modeling
  - Identifying assets, threats, and vulnerabilities
- **Lesson 2: Risk Assessment Techniques**
  - Qualitative vs. quantitative risk assessment
  - Common frameworks and standards (e.g., NIST, ISO)

Module 3: Best Practices for Secure Development

- **Lesson 1: Secure Software Development Lifecycle (SDLC)**
  - Integrating security into the SDLC
  - Security practices in each phase (planning, design, development, testing, deployment, maintenance)
- **Lesson 2: Secure Coding Practices**
  - Common coding vulnerabilities and how to avoid them
  - Secure coding guidelines and standards

Module 4: Supply Chain Security Tools and Technologies

- **Lesson 1: Source Code Security Analysis Tools**
  - Static and dynamic analysis tools
  - Open-source and commercial options
- **Lesson 2: Dependency Management and Vulnerability Scanning**
  - Managing third-party libraries and dependencies
  - Tools for vulnerability scanning and management (e.g., Snyk, Dependabot)

Module 5: Securing the Build and Deployment Pipeline

- **Lesson 1: Continuous Integration/Continuous Deployment (CI/CD) Security**
  - Securing CI/CD pipelines
  - Tools and practices for secure CI/CD (e.g., Jenkins, GitHub Actions)
- **Lesson 2: Infrastructure as Code (IaC) Security**
  - Best practices for securing IaC
  - Tools and frameworks for IaC security (e.g., Terraform, AWS CloudFormation)

Module 6: Monitoring and Incident Response

- **Lesson 1: Monitoring and Logging**
  - Importance of monitoring and logging
  - Tools for monitoring and log management
- **Lesson 2: Incident Response Planning**
  - Developing an incident response plan
  - Steps for effective incident response
  - Case studies of incident response

Module 7: Compliance and Governance

- **Lesson 1: Regulatory Requirements and Standards**
  - Overview of relevant regulations (e.g., GDPR, HIPAA)
  - Industry standards and best practices
- **Lesson 2: Governance and Policy Development**
  - Developing security policies for the software supply chain
  - Implementing and enforcing policies

Module 8: Future Trends and Emerging Technologies

- **Lesson 1: Emerging Threats and Technologies**
  - Trends in supply chain attacks
  - Emerging technologies for supply chain security (e.g., blockchain, AI)
- **Lesson 2: Preparing for the Future**
  - Adapting to new threats and technologies
  - Continuous improvement and learning

Figure 1: Ask ChatGPT to design a "Software Supply Chain Security" course.

The rest of this article is organized as follows. First, we will follow some standard and philosophy (KU, 2024) (TAX, 2024) to define the outline, technical scope, and focus of the new course. The referred documents include the "2020 CAE Cyber Defense (CAE-CD) Knowledge Units" (KU, 2024) and the Bloom's Taxonomy for categorizing educational goals (TAX, 2024). Then, we will provide the details of the course materials including the teaching environment construction, course outline, labs. After that, an experimental but real teaching effectiveness evaluation will be provided for analysis. The shortage of the teaching experiment and potential enhancement directions will be discussed in the "Future Work" section. And finally, we summarize this article in the "Conclusion" section.

2. HOW TO TEACH IT

The "Software Supply Chain Security" course somehow is related to software security analysis. Therefore, ideally, the course developer can reuse a few ideas and materials from the existing Software Analysis course. However, in the CS department of the Southeast Missouri State University (SEMO), we did not offer any software analysis course (this situation was improved in Fall 2024 though). So, at that time, the course developer had to develop the experimental course from scratch.

Decide the outline of the course

In the document of "2020 CAE Cyber Defense (CAE-CD) Knowledge Units", the Supply Chain Security (SCS) is an optional Knowledge Unit

ISSN: 2473-4901
v10 n6192

(KU), which includes both software and hardware components. Per the document, the intent of the SCS KU is to "provide students with an understanding of the security issues associated with building complex systems out of third-party components of unknown (and potentially unknowable) origin." There are several keywords in this statement: Complex Systems, Third-party Components, Unknown and Potentially Unknowable. These keywords should help us identify the scope of the course and the challenges the instructor and students are facing. This document also provides the learning outcomes for the SCS KU.

To complete this KU, students should be able to:

- Describe the issues related to outsourcing hardware and/or software development and/or integration.
- Describe methods to mitigate these issues, and the limitations of these methods.

And it provides typical topics that should be covered in the SCS KU.

To complete this KU, all Topics must be completed:

- Global Development
- Offshore Production
- Transport and Logistics of IT Components
- Evaluation of 3rd Party Development Practices
- Understanding of the Capabilities and Limits of Software and Hardware Reverse Engineering

Simply skip the hardware, transport, and logistics parts of the above information, we can tell that SDLC, Third-party Software, and Reverse Engineering are the focus of the course, and the course should be practical to the students.

As a summary, the newly developed Software Supply Chain Security course should teach students the practical skills (such as reverse engineering) to solve the security issues embedded in the SDLC when third-party component is integrated to the software.

Based on the above summarized guideline, we decided the outline of the course, as shown in Figure 2.

Figure 2: Our outline for the Software Supply Chain Security course.

Integrate Teaching Philosophy: Hands-on, practical, competency including soft skills

Also, the Software Supply Chain Security course needs to align with the existing program, supporting the learning outcomes required by ABET, NCAE-CD assessment process. According to Bloom's Taxonomy, as shown in Figure 3, there are different levels of educational goals the students need to achieve during the course study.

We designed the course specific labs around the different levels of education goals and gave priorities to hands-on and practical content with the hope to improve our students' competency in the job market.

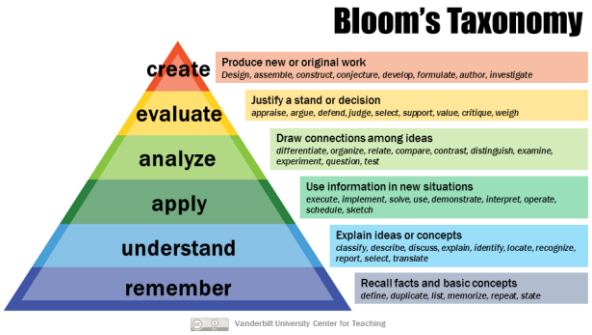


Figure 3: Bloom's Taxonomy for different levels of education goals.

Decide the teaching and learning environment

Different software execution environments require different considerations on tools and lab designs. Windows itself is not free, not to mention the relied third-party components. IoT environment seems more attractive because of the following advantages:

- IoT hardware and software are either open-source or available at very low cost. Teachers and students do not have to worry about the corruption of the

hardware and software even though cybersecurity labs are usually destructive.
- IoT applications integrate a huge number of third-party components, which are the desired target of SSCS. IoT application development usually relies on existing libraries. The pure software development has been reduced to less than hundreds of new lines. This feature will speed up the exploration of mysterious third-party code while helping ignore the unnecessary distraction from coding.

However, as a matter of fact, IoT applications rely on the cross-compilation to compile the application under one environment then run the application under another environment. Linux (virtual machine) is the recommended compilation platform for IoT applications due to its availability and versatility. Therefore, the teaching and learning environment was Linux in our first experimental course.

Detailed Labs Design

We lectured the course material in Summer 2023 for a 6-week period and a small group of students. Due to the tight schedule, all the technical topics were distributed to 6 modules with one module a week. The course materials were organized based on SDLC phases with special consideration for integrating third-party components. For example, the early phases in third-party software development (such as System Analysis, Architecture Design and coding) have been covered by the vendor of the adopted third-party software. The user of the third-party software only considers how to select the third-party software, and then how to integrate it to the user's software/application. As well as how to test/verify the external components. Debugging third-party software is challenging but also necessary, no matter whether the user can get immediate support from the vendor or not. Finally, the vendor of the third-party software may have their own rhythm, therefore, they may not be able to align with the user's development schedule, which means, the user needs to know how to update or upgrade the third-party software when it is ready.

Lab for Unit 1

In the first week, the instructor will need to explain the composition of a software, which is the fundamental of the Software Supply Chain (Figure 4). We designed a lab directing our students to check the executable's size after compilation, so that they would realize there are a large number of supporting libraries behind the source code they wrote, which will be integrated into the executable. Along with the post-class reading on "Open-Source Security", the students will understand the potential damages that could be caused by the third-party or open-source code. So, they will be cautious when they adopt a third-party or open-source components in their software. Table 1 indicates the different levels of learning goals the students are expected to achieve in the first lab.

An example of good student work is shown below:

"The larger size of the compiled program compared to the source program is primarily due to the inclusion of additional code and data during the linking process. This includes:

Standard library functions: The compiled program needs to link with the C runtime library, which provides standard functions like printf and scanf. These library functions add additional code to the program.

Runtime support code: The linker includes runtime support code necessary for proper program execution, such as initialization and cleanup routines, exception handling (if applicable), and other internal mechanisms.

Symbol tables and debugging information: The compiled program includes additional information for debugging purposes, such as symbol tables that map function and variable names to their memory addresses. This information aids in debugging and error analysis.

Additional sections: The compiled program may have additional sections, such as ".rodata" (read-only data), ".eh_frame" (exception handling information), and .dynamic (dynamic linking information). These sections store various data required by the program's execution and increase the overall size.

It's important to note that the larger size of the compiled program does not necessarily imply that it is less secure or more vulnerable. The additional code and data included during compilation and linking are necessary for the program to run correctly and interact with the underlying operating system and runtime environment."

Lab for Unit 2

Unit 2 (Figure 5) talked about how to securely integrate third-party software. First the instructor introduced the compilation process. Then the instructor indicated the compilation command option to integrate a library into an executable. And finally demonstrated the tools that can be

used to analyze the components of an executable. We designed two labs for Unit 2. The first one required the students to derive the header information of an executable by using tools such as *readelf* and *objdump*. From the information the student should be able to figure out where the initialized global variables, uninitialized global variables, constant strings, and instructions are saved in the executable. The second lab instructed the students to generate their own shared lib and then call it from their software.

An example of good student work is shown below:

"To generate a shared library (also known as a dynamically linked library) and integrate it into your software, you can follow these general steps:

Write the library code: Create the source code for your library in a programming language of your choice. This code should contain the functions and logic you want to expose to your software.
Compile the library: Use the appropriate compiler and build tools to compile the source code into an object file. The commands and options will vary depending on the programming language and platform you're using. Make sure to compile it as a shared library. For example, using GCC on Linux, you can compile a shared library with the following command:
vbnet
Copy code
gcc -shared -o libyourlibrary.so yourlibrary.c
Link the library: When compiling your software, you need to link it with the shared library you created. This step varies depending on your build system or IDE. For example, if you're using GCC on Linux, you can link your software with the shared library using the following command:
bash
Copy code
gcc -o yoursoftware yoursoftware.c -lyourlibrary -L/path/to/library
Here, -lyourlibrary specifies the library name without the "lib" prefix and the file extension, and -L/path/to/library specifies the path to the directory where the library is located (if it's not in the default search path).
Distribute the library: Copy the shared library file (libyourlibrary.so) to an appropriate location where it can be accessed by your software. This could be a system library directory or a custom directory that you configure for your software.
Load the library in your software: Depending on your programming language and platform, you'll need to use specific mechanisms to load the shared library dynamically at runtime. This allows your software to access the functions and symbols provided by the library. The exact steps for this will depend on the programming language or framework you're using. Here are a few examples:
C/C++: Use functions like dlopen and dlsym from the dlfcn.h header to load and access symbols from the library dynamically.
Java: Use the System.loadLibrary method to load the library. Then, you can access the library's functions through the Java Native Interface (JNI).
Python: Use the ctypes module to load the library and access its functions."

Lab for Unit 3

After integrating the third-party or open-source components to a software, the user needs to verify their functionalities and security. This is the main content of Unit 3 (Figure 6). Besides the ordinary software testing, we designed a lab requiring the students to do research on static scanning tools and choose one to scan an executable and demo the buffer overflow vulnerability that can be detected by the tool.

Lab for Unit 4

Verifying the seamless integration of the third-party component is easily blocked by the lament bug or incompatibility embedded in the third-party component. Waiting for the technical support from the vendor is not a realistic solution in most cases and very time-consuming. Therefore, the user of the third-party component should prepare for independent debugging. In Unit 4, the instructor demonstrated Linux OS debugging, which is a good example of third-party component debugging (Figure 7). We designed 3 labs around the built-in Linux *eBPF* virtual machine focusing on the user interfaces but ignoring the implementation details.

Lab for Unit 5

During the software development, often the developers will encounter a situation where some relied-on third-party component got a new version released with critical security issues being fixed. To fix the security issue inherited from the old version of the third-party component, update/patch/upgrade is needed. Unit 5 talked about these scenarios (Figure 8). We designed one lab for Unit 5. After completing it, the students should know how to handle these scenarios.

Lab	R	U	A1	A2	E	C	T	Notes
1. File size increased after compilation	X	X	X	X				

R: Remember, U: Understand, A1: Apply, A2: Analyze, E: Evaluate, C: Create, T: Teamwork

ISSN: 2473-4901
v10 n6192

Table 1: Lab for Unit 1 Mapping to the Bloom's Taxonomy.

Lab	R	U	A1	A2	E	C	T	Notes
2. Derive its sections information from an executable	X	X	X					

Table 2: Lab for Unit 2 Mapping to the Bloom's Taxonomy.

Lab	R	U	A1	A2	E	C	T	Notes
4. Static Scanning				X	X			
R: Remember, U: Understand, A1: Apply, A2: Analyze, E: Evaluate, C: Create, T: Teamwork								

Table 3: Lab for Unit 3 Mapping to the Bloom's Taxonomy.

Lab	R	U	A1	A2	E	C	T	Notes
5. Try another BCC example	X	X	X			X	X	Try BCC python coding to customize a BPF trace
6. Try bpftrace tool	X	X	X				X	
7. Other BCC tools	X	X	X				X	Try 'biolatency' and 'cpudist' BCC tools
R: Remember, U: Understand, A1: Apply, A2: Analyze, E: Evaluate, C: Create, T: Teamwork								

Table 4: Lab for Unit 4 Mapping to the Bloom's Taxonomy.

Lab	R	U	A1	A2	E	C	T	Notes
8. Temporarily replace a library by utilizing the LD_LIBRARY_PATH environment variable	X	X	X		X	X	X	
R: Remember, U: Understand, A1: Apply, A2: Analyze, E: Evaluate, C: Create, T: Teamwork								

Table 5: Lab for Unit 5 Mapping to the Bloom's Taxonomy.

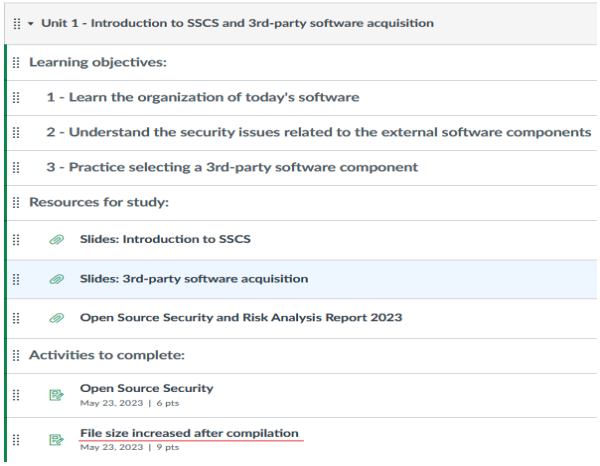


Figure 4: Content of Unit 1.

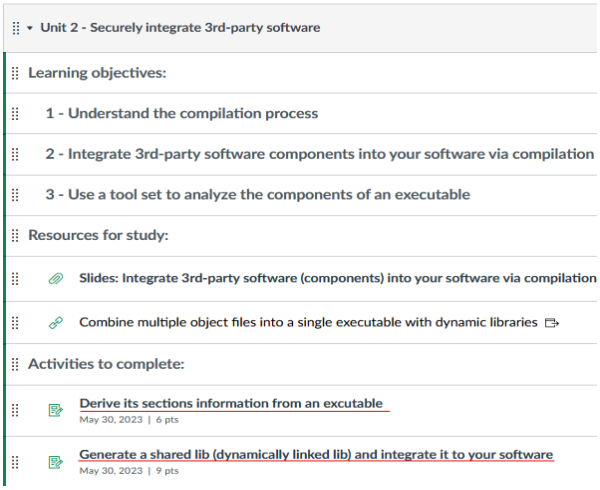


Figure 5: Content of Unit 2.

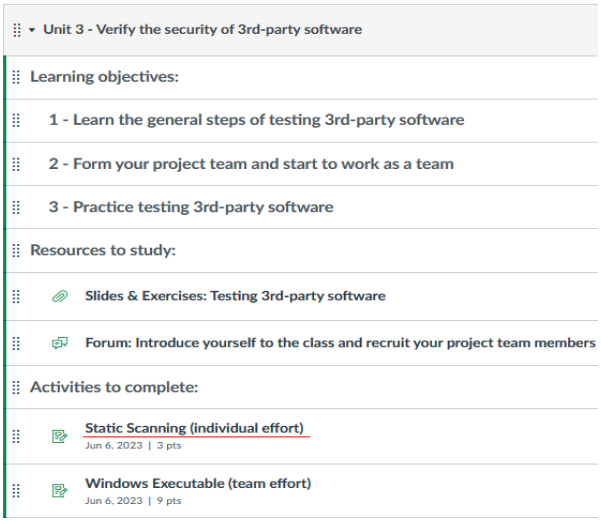


Figure 6: Content of Unit 3.

ISSN: 2473-4901
v10 n6192

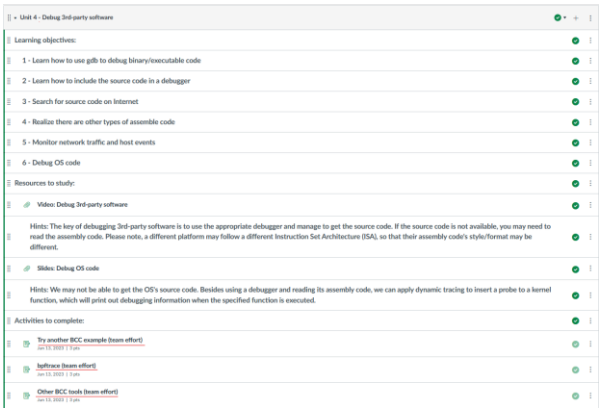


Figure 7: Content of Unit 4.

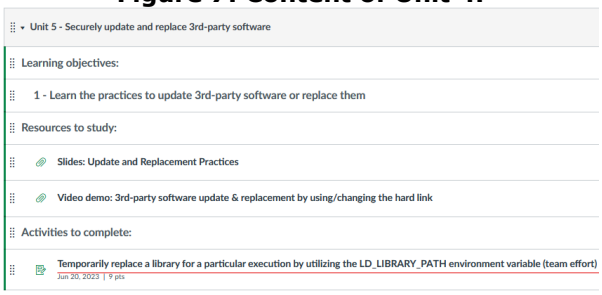


Figure 8: Content of Unit 5.

3. TEACHING EFFECTIVENESS ANALYSIS

	Question Text	N	RR	Avg	CY Avg	Div Avg	Sch Avg					
Grp	Required Questions (Online)			4.93	4.49	4.36	4.31					
								Required	Option	GenEd	Elective	
1	Course is for major or minor	7	58%					57%	0%	0%	43%	
								Always	Freq - ently	Half	Rarely	Never
2	Percent of time student prepared for class	7	58%					71%	29%	0%	0%	0%
								Str Agree	Agree	Neutral	Disagree	Str Disagree
3	Expectations clearly communicated at semester start	7	58%	4.86	4.60	4.54	4.56	86%	14%	0%	0%	0%
4	Required materials used effectively	7	58%	5	4.57	4.47	4.44	100%	0%	0%	0%	0%
5	Assessments reflected course content	7	58%	5	4.56	4.49	4.50	100%	0%	0%	0%	0%
6	Assignments helped understanding	7	58%	4.86	4.59	4.47	4.48	86%	14%	0%	0%	0%
7	Timely feedback on academic performance	7	58%	4.86	4.52	4.35	4.36	86%	14%	0%	0%	0%
8	Instructor interested in teaching class	7	58%	5	4.54	4.47	4.56	100%	0%	0%	0%	0%
9	Subject matter communicated clearly	7	58%	4.86	4.48	4.36	4.41	86%	14%	0%	0%	0%
10	Instructor well prepared/used class time effectively	7	58%	5	4.49	4.42	4.46	100%	0%	0%	0%	0%
11	Instructor responsive to student questions/ideas	7	58%	4.86	4.52	4.42	4.49	86%	14%	0%	0%	0%
12	Knowledge/abilities in subject increased	7	58%	4.86	4.55	4.44	4.45	86%	14%	0%	0%	0%
13	Instructor engaged students via discussion activities	7	58%	5	4.42	4.28	4.32	100%	0%	0%	0%	0%
14	Online course organized logically/conducive to learning	7	58%	4.86	4.57	4.44	4.43	86%	14%	0%	0%	0%

Figure 9: Student Evaluation on Teaching Effectiveness.

Time	Course #	Title	Modality	E6	E7	E8	E10	E11	E12
Regular Semester Courses									
FA 2023	CS505	Data Mining	Online	4.93	4.71	4.71	4.79	4.64	4.64
SP 2023	CS645	Internet of Things	Face-to-Face	4.69	4.92	4.92	4.85	4.85	4.85
FA 2022	CY201	Intro to Cybersecurity	Face-to-Face	4.43	4.43	4.57	4.21	4.57	4.43
	CY610	Web Application Security (Graduate)	HyFlex	4.81	4.81	4.94	4.88	4.81	4.69
FA 2020	CY320	Info Security in Sys Admin	Face-to-Face	3.2	3.8	3.8	3.6	3.4	3.8
	CS480*	Data Comm.	Face-to-Face	3.55	4.1	3.7	3.45	3.55	3.5
SP 2020	CY410	Web Application Security	Face-to-Face	4.08	4.17	4.42	4.42	4.33	4
	CY420	Computer Forensics	Face-to-Face	3.57	4	4.29	3.86	4.07	3.5
	IS299	Security in Data Protocols	Online	3.75	3.83	3.58	3.58	3.75	3.83
FA 2019	CY501	Intro to Cybersecurity	Online	4.56	4.56	4.67	4.56	4.67	4.67
	CY510	Info Security & Assurance	Face-to-Face	4.62	4.67	4.71	4.62	4.76	4.71
	CY520	Info Security in Systems Admin	Face-to-Face	4.27	4.19	4.5	4.38	4.35	4.27
Summer Courses									
SU 2021	CS605	Research Methods	Online	4.92	4.85	4.85	4.92	4.92	5
	CY655	Research Methods in CY	Online	5	5	5	4.71	4.86	4.71

Notes:
E6: Understanding from Assignments; or Assignments helped understanding.
E7: Timely Feedback; or Timely feedback on academic performance.
E8: Genuinely Interested in Teaching Class; or Instructor interested in teaching class.
E10: Prepared, Organized, and Effective Timewise; or Instructor well prepared/used class time effectively.
E11: Responded to Student Questions and Ideas; or Instructor responsive to student questions/ideas.
E12: Instruction Increased CLOs; or Knowledge/abilities in subject increased.

Figure 10: Student Evaluation on Teaching Effectiveness – Compared to Other Classes

Student evaluation on a specific course is usually the first-hand input for teaching effectiveness analysis. Figure 9 shows the student evaluation on the teaching effectiveness of the experimental summer course. The underlined items were related to the course materials either directly or indirectly. Most of the rest items were focused on how nicely the instructor communicated with the students. They are not reflective to the quality of the course material. Therefore, we will not discuss them here.

Among the columns, the 'N' column represents the number of students provided the course evaluation. And 'RR' means Response Rate, that is how many percentages of students provided the course evaluation. Each evaluation varies between 0-5 with 5 being the highest evaluated.

Among the underlined items, "Required materials used effectively", "Assessments reflected course content", "Assignments helped understanding", "Knowledge/abilities in subject increased", and "Online course organized logically/conductive to learning" metrics were directly related to the quality of the course materials. The average evaluation in this category is equal or above 4.86, which is much higher than the other summer Cybersecurity courses as well as the average evaluation of the school and university.

"Instructor interested in teaching class", "Instructor well prepared/used class time effectively", and "Instructor engaged students via discussion activities" are indirect metrics. Usually if the course material is new, interesting, and

ISSN: 2473-4901
v10 n6192

providing insights, the student will recognize the instructor's effort by giving high evaluations on these indirect metrics. The average evaluation in this category achieves nearly the full points with all responded students strongly agreeing. If we check the average evaluation on all Cybersecurity courses, this category has relatively lower evaluations than the previous category. So, we would interpret it as a big improvement. Figure 10 presents the student evaluation data for other new courses taught by the first author. Obviously, the student's feedback to this experimental course was more positive.

Overall, we can tell this is positive feedback to the course materials. The strength of the course materials is their technical quality: easy to understand and follow. The weakness of the course materials is its coverage and depth (these can only be revealed by students' text remarks). Only two student remarks were recorded: "This course content was up to date," and "very cooperative and way of explanation is very nice".

Figure 11 illustrates the score distribution of the student evaluation.

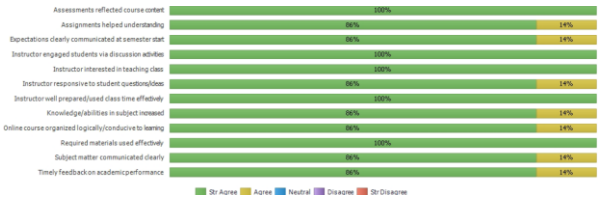


Figure 11: Score Distribution of Student Evaluation

While the course emphasizes practical technical skills, it would benefit from a strong integration of soft skills, which are essential for cybersecurity professionals who must often collaborate with diverse teams and communicate complex technical issues to non-technical stakeholders. Regarding students' soft skills development through this course study, we only focused on collecting data for teamwork and professional communication.

Of the 10 assignments/labs, 5 require team effort. Students are encouraged to take on different roles within their teams - some involving leadership, others followership. A student may lead one team in one assignment while acting as a follower in another. The key lesson to impart is how to contribute effectively to the team to achieve a 1+1>2 level of efficiency.

Of the 10 assignments/labs, 2 require technical communication:

1. Open Source Security
   Read the Open Source Security and Risk Analysis Report 2023 and provide your summary and insight

2. File size increased after compilation
   Explain why the compiled program vuln1 has a much larger size than the size of the source program vuln1.c.

From the students' grades, these labs/assignments well developed their soft skill.

4. FUTURE WORK

Currently we have eight labs designed for the course, which are not sufficient for a regular semester course. We shall add 4~6 more labs to the course in the future. Case studies will be good fits. The instructor can ask students to study the SolarWinds incident, Log4j exploitation, and the outages caused by CrowdStrike updates. These cases were similar but had different root causes. Analyzing them will help students better understand the dangers of the real cyber world and establish their best practices to protect software from supply chain attacks. Another promising research direction will be the problem of inter-vendor compatibility. A software product may integrate multiple 3rd-party libraries. However, if they have potential conflicts with each other, this will be a big problem to challenge the security professionals. One more regular semester at least is required for us to prepare and add manuals for the new labs. A minimum of one faculty member and one student worker will be required.

As aforementioned, the experimental lecturing and lab practices were based on Linux environment. Given the big differences between Windows, iOS, Android, and Linux operating systems, it is necessary to develop similar modules for the other operating systems than only for the Linux environment. Because iOS and Android somehow can be treated as variants of Linux OS, firstly developing iOS and Android specific modules seems a good choice because more course materials can be reused. We estimate that one or two regular semesters will be needed before we can develop the course materials for iOS, Android, and Windows platforms. A minimum of two faculty members and two student workers will be required.

Besides following academic standards, we can also integrate industry standards in the course

ISSN: 2473-4901
v10 n6192

development. The following ideas seam promising:

Align Course Objectives with National Institute of Standards and Technology (NIST) Standards. Specifically, the five core functions (Identify, Protect, Detect, Respond, Recover) of the NIST Cybersecurity Framework (CSF) can be used to guide students on protecting software supply chain.

Use the NICE Framework for Competency-based Learning. Specifically, we can map the course content to the work roles defined in NICE Framework. Or in a fine granularity, map the course content to the tasks and KSAs (Knowledge, Skills, and Abilities) defined in NICE Framework.

Use real-world case studies where NIST guidelines and the NICE framework were applied in software supply chain security breaches.

We plan to integrate these industry standard - based ideas into a new research paper.

5. CONCLUSION

Developing Software Supply Chain Security courses is urgent and demanding in the Cybersecurity program of NCAE-CD institutions. In this article, we present our Linux-environment-based, experimental course materials including the outline, lab practices, as well as the idea. Our major contributions in this paper include timely response to the urgent call for the course materials of Software Supply Chain Security; the easy-to-follow methodology for developing the Software Supply Chain Security course as well as other emerging cybersecurity courses; the hands-on lab development framework; and preliminary learning effectiveness analysis. The preliminary feedback from the students, who attended the experimental summer course, indicates that the organization of the course modules seemed effective.

6. REFERENCES

DEF. (2024). Retrieved from https://binmile.com/blog/build-secure-scm-software/.

KU. (2024). Retrieved from https://dl.dod.cyber.mil/wp-content/uploads/cae/pdf/unclass-cae-cd_ku.pdf.

TAX. (2024). Retrieved from https://www.flickr.com/photos/vandycft/29428436431