Developer Perceptions on Trusting Software

Ryan ElKochta
University of Maryland
relkocht@umd.edu

Anders Spear
University of Maryland
aspr@umd.edu

Michael Suehle
University of Maryland
msuehle@umd.edu

Tien Vu
University of Maryland
vut@umd.edu

Abstract

How do software developers decide to trust third-party code? To investigate the level of awareness software developers have regarding supply chain vulnerabilities, we conducted seven semi-structured interviews with University of Maryland students who had experience writing code in industry through internships and full time positions. We performed inductive coding on the interviews and found that popularity, usage by personally known people, and verifiability of source were major factors that the participants looked for to determine the trustworthiness of dependencies. We also found that industries tend to assign dedicated groups to ensure security and that participants care roughly the same about security when developing personal projects versus projects for industry.

1 Introduction

Every developer that plays a role in the development and distribution of software introduces the potential of a supply chain attack as every developer must choose to trust the software they run on their systems and the software they pull in as dependencies. If a developer trusts malicious or vulnerable software, then the software they produce can itself be malicious or vulnerable. Dependent software can then inherit the malicious or vulnerable code via the supply chain of software. As a result, it is important for developers to know who they are trusting, whether it is a package maintainer or another source for software, such as the developer themselves. Moreover, programming languages tend to have their own package environment. Python might use pip, Rust uses cargo, and C compilers typically use libraries from the system itself. Both the system software and the dependency software can impact the software developed and distributed from the developer's system and thus are important factors in the security of the supply chain.

There have been an increasing number of supply chain attacks performed in the wild. For instance, XZ Utils, a compression library included on most Linux systems, was backdoored by one of its maintainers in February 2024; a remote SSH backdoor consequently came very close to making it into popular Linux distributions [2]. In another instance, following a dispute with Canadian messaging app Kik over ownership of an NPM package name, a developer broke the build systems of countless JavaScript applications, including those from Meta, Netflix, PayPal, and Spotify, by simply removing a popular package of his [4]. This incident demonstrated the wide breadth of impact a software supply chain disruption incurs.

Our primary research question is "how do software developers decide to trust the software they use?" To answer this question, we conducted semi-structured interviews asking participants about their experiences developing software professionally and for personal projects. The interviews used structured questions to target key questions about the developer's practices for downloading and installing software and packages and how they decide to trust the software and packages. Making the interviews semi-structured allowed the interviewers to follow-up on participant statements more dynamically, especially in regards to ideas and solutions not previously considered. Ultimately, this approach was able to obtain a broad sense of the user's perceptions and trust before allowing the interviewer to dig deeper into what their behaviors and thoughts are.

2 Related Work

2.1 Types of Supply Chain Attacks

Ladisa et al. [11] develop a taxonomy to describe mechanisms for performing a supply chain attack against an open-source project. They describe four systems that can be leveraged: version control systems, build systems, distribution platforms, and open-source developers' workstations. This taxonomy is useful to our work, although we are not limiting ourselves to the open-source supply chain but investigating the supply chain of even proprietary code. Our interview study will primarily focus on the build system (dependency managers and package repositories) and developer workstation aspects.

Ohm et al. [13] name some specific issues that we aim to investigate. They note that developers often place a lot of trust in the authenticity of packages that they include as dependencies. Even a careful developer might make mistakes; Bagmar et al. [1] describe package impersonation attacks, where a malicious actor publishes a package with a confusingly similar name to a popular dependency. During our semi-structured interviews we asked developers how they added dependencies to their projects and try to understand their thought process for why they thought those packages were safe (if they were even considering this). Additionally, we investigated corporate policies for vetting project dependencies.

Another issue is mentioned by Chandramouli et al., [3] who write that developers' workstations "present a fundamental risk" to software supply chain security. Developers may install third-party software on their workstations, which dramatically increases the supply chain attack surface. During our semi-structured interviews we asked developers how they reason about the trustworthiness of the software they installed on their workstations and how this might have affected the security of software they developed.

2.2 Reflections on Trusting Trust

When discussing supply chain attacks, the 1984 article "Reflections on Trusting Trust" by Ken Thompson [17] cannot go unmentioned. Although it does not discuss supply chain attacks specifically, it discusses the underlying assumptions that allow attacks on the software supply chain to occur. Thompson summarizes this well in the last section: "The moral [of this article] is obvious. You can't trust code that you did not totally create yourself."

He demonstrates this with a toy example in which he "attacks" a C compiler by compiling a malicious version of the compiler. The malicious compiler could then compile other code with a Trojan horse included. Because the compiler itself was malicious, no inspection of the compiled program's source code would turn up anything malicious. In this case, all that was needed to propogate malicious code was a malicious compiler on a developer's machine. Although this toy example of an extremely potent supply chain attack was written and published in 1984, this kind of supply chain vulnerability would continue to exist today.

Our research thus seeks to understand the ways in which up-and-coming developers at the University of Maryland trust code they did not create themselves and the actions they may or make not take to avoid becoming or creating a supply chain attack vector.

2.3 Prevention and Mitigation Techniques

One mitigation technique for software supply chain attacks that has gained popularity in recent years is the creation of a Software Bill of Materials (SBOM). An SBOM is a formal, machine-readable file that documents all components and dependencies of software and their relationships [14]. Their popularity likely grew due to their mention in a White House executive order that identified them as a critical measure for enhancing supply chain security [9].

There are also scanning tools, such as Eclipse Steady [15], that scan software projects for dependency vulnerabilities, package analysis techniques for "sanitizing" package manager repositories [7], and potential metrics that software developers use for determining the security of packages [6]. We hypothesized that the majority of our participants rely on a combination of these techniques when doing software development in industry with the consideration potential metrics from Lopez de la Mora and Nadi [6] such as popularity and release frequency to decide whether or not to trust a package being the most common.

2.4 Supply Chain Interview Studies

A common approach for qualitative research in supply chain usage are interview studies. Fourné et al. [8] interviewed 24 participants from the Reproducible-Builds.org project, identifying experiences that help and hinder adoption. They identify that communication is crucial for disseminating R-B benefits to the community. We were aiming to understand the awareness that newer software developers may have about R-B and other tools for validating the integrity of dependencies, seeing if developers have been exposed during school or early in their career.

Wermke et al. [19] interviewed 27 open source maintainers about security and trust considerations in their projects, finding that the projects were highly diverse both in deployed security measures and trust processes. They argued that the community cannot treat Open Source Code (OSC) as a black box, and needs to put their support back into the projects. The authors focused on the processes relating to security within OSC itself. Conversely, we wanted to focus on the end users of the OSC, hoping to identify their level of support and understanding of the security related to OSC.

Larios Vargas et al. [12] identified 26 technical, human, and economic factors that developers consider in their dependency selection processes based on 16 interviews and a survey with 115 developers. We used the factors they identified to focus the interviews we conduct. We wondered about the transferability of their findings toward the student developer demographic, and if there are particular groups of considerations that our studied users prefer.

Wermke et al. [18] interviewed 25 software developers, architects, and engineers about the role/prevalence of OSC, company attitudes towards the security of OSC, and stakeholder considerations regarding OSC. They found that although OSC adoption is widespread, users have ambivalent attitudes towards security. Many users mentioned only planning to address incidents if/when they happen, while still wishing for

large scale audits of said OSC dependencies. Their study focused heavily on the frameworks that companies use to address the problem, we focused our research on the individual developer. Students likely have experience with much smaller-scale projects with less rigorous pipelines, so we expected to gain insight into whether they still consider supply chain vulnerabilities in their development process.

3 Methods

3.1 Participant Selection

We posted a 1-2 minute eligibility survey (hosted using Qualtrics) on the UMD Discord channel, UMD CMSC Discord channel, UMD CS graduate student Slack channel, and UMD Reddit page.

This survey captured basic information for determining software development experience level such as coursework level, major, and experience writing software as well as three knowledge check questions from Davinlova et al. [5] to check if the respondents actually have programming experience. See Appendix A for the eligibility survey. For a respondent to be eligible, they must have experience writing software in industry, experience in writing personal software projects, and they must correctly answer the three experience check questions. We selected seven of the eligible participants to participate in the interview, prioritizing more experienced respondents.

3.2 Semi-structured Interview

We conducted 20 to 30-minute semi-structured interviews with each of the seven participants who were compensated $25 for their time. The interviews followed an outline allowing for additional follow-up questions as needed. The outline is in Appendix B. The information divulged in the interview was collected via audio recording and transcribed with the aid of OpenAI Whisper. The transcripts were then imported into NVivo for analysis and the audio recordings deleted.

We began our semi-structured interviews with a discussion of the participant's professional development experience. We asked what languages were used and what third-party packages (if any) were imported into each project. Next, we had the participant discuss their thought process in choosing specific packages, in addition to corporate procedures for pulling in new dependencies. During this time, we were careful not to explicitly mention security to avoid cluing the participant to our research question[1]. We next were more explicit in asking the participant whether they took any steps to ensure dependency trustworthiness, and what those were.

We followed this with a similar discussion of what third-party software participants had installed on their development machine(s) and where they obtained this software, with the goal of ascertaining whether their workstation was trustworthy. In addition, we discussed all of the above for personal projects, with the aim of learning whether participants' practices differed in a professional or personal environment.

Finally, we asked participants general, broad questions about their views on what makes a third-party piece of code trustworthy and the effects of having malicious or vulnerable software on their workstations.

3.3 Ethical Considerations

Our study was approved by the UMD Institutional Review Board as exempt, and as anticipated, we did not encounter any major ethical issues. To minimize potential for ethical issues, we took several steps:

- Our eligibility survey did not ask participants for any personal information aside from email for the purposes of contacting them for an interview.
- Our interviews began with a reminder to participants not to disclose anything sensitive about their company if possible.
- Our interviews began with a statement informing participants that they can later redact information they felt was too revealing.
- Our transcripts of the audio recordings redacts any and all names, whether they were of companies or people.
- Our audio recordings were deleted after transcription, such that there is no link between the participant (whose name was not recorded) and their company (whose name was redacted).

These steps were taken for several reasons. The first is that reducing the amount of information we retain about the participant and their company minimizes potential for any ethical issues right off the bat. The second is that our questions may cause the participant to accidentally share sensitive, NDA-restricted information. Redacting names, especially of companies, helps to generalize the information gathered and prevents the participant from being at risk of legal action from their employer.

3.4 Coding and Analysis

Our data analysis involved qualitative coding. We assigned the interviews to each other such that each interview would be individually coded by two researchers. After every interview was coded twice, we met and consolidated the codes into final, high level themes that we analyze in 4 Results and Discussion. We chose to limit ourselves to qualitative, rather than quantitative, analysis because we had a small sample

[1] As we will discuss, this was not entirely successful.

size and our sample was not very representative of software developers in industry.

4 Results and Discussion

4.1 Participant Demographics

We received a total of 33 responses in our eligibility survey. 14 (42%) respondents were undergraduate students, 18 (55%) were graduate students, and 1 (3%) was not enrolled as a student. We chose to prioritize interviewing the graduate students because they were likely to have the most experience developing software both professionally and personally. Outside of the graduate students, we also invited a few undergraduate students with some professional experience. In total, we invited 13 participants to be interviewed and were able to interview 7. See Table 1 for more details.

Table 1: Demographics of selected participants.

ID	Program	Professional Experience
P1	Graduate	Full-time
P2	Graduate	Full-time, Part-time, Internship
P3	Undergraduate	Internship
P4	Graduate	Full-time
P5	Undergraduate	Internship
P6	Graduate	Full-time, Part-time, Internship
P7	Graduate	Full-time, Internship

4.2 Selection of third-party dependencies

All participants mentioned popularity as a significant factor in which packages they trusted. In particular, participants evaluated popularity based on mentions on Stack Overflow, Reddit, Hacker News, and GitHub. Two mentioned the download count on their package manager as well. A couple of participants mentioned looking at which dependencies similar open-source projects used, and pulling those in themselves. An interesting to note is that mentions on these websites do not necessarily correlate with popularity; a concern here is that if a malicious actor published fake posts on these websites recommending a malicious dependency, many of the participants would not have caught this. Additionally, a supply chain attack could absolutely still be mounted against a popular dependency.

Interestingly, multiple participants mentioned that they had to pull older, potentially unmaintained versions of packages due to dependency and/or toolchain conflicts. In particular, P7 said that they usually used a few-months-old versions of Python packages “because that seems to be very compatible with the deep learning models we are using right now.” This was surprising to us; we expect that some companies are not diligent about keeping dependencies up-to-date, but not *intentionally* keeping old versions of dependencies. We suspect that a solution to this would be for packages to more rigidly adhere to SemVer [16], and keep older minor versions maintained with security patches. It is also possible that developers pin old versions for non-security-critical packages; the issue is that a vulnerability in one package may be exploited to take over another, potentially security-critical, piece of installed software.

Another factor some participants mentioned was the source of the package. One participant mentioned trusting packages affiliated with large institutions (such as research code); another mentioned checking the name of the package to verify that it matches expectation. Finally, two participants mentioned source code availability being a factor that makes a software package more trustworthy.

Particularly in a professional context, many participants trusted others to decide on which packages to use. Some mentioned being told by people they know about relevant packages, and a few mentioned working within the constraints of corporate policy. (We will elaborate on our findings about corporate policies in section 4.3).

We also asked participants about how they obtained the software packages. The majority used an official or semi-official package manager of the language they were developing in; in a professional context this was often NPM, pip, or Conan. Only two participants mentioned having an internal artifactory with known-good packages. Most participants installed software themselves, and the vast majority of those who did mentioned having little oversight in how they did so.

Finally, most participants indicated that at least on occasion, they took no steps to ensure third-party dependencies were secure.

The fact that the vast majority of participants often did not consciously think about what repository the package came from, or what organization(s) are responsible for maintaining it, is fairly alarming. Educating developers on security is not likely to be sufficient, since developers’ primary aim is to *solve the problem at hand,* rather than to do so securely. Technical solutions, such as SBOMs [14], offer one possible path forward.

4.3 Removing the Developer From the Loop

When discussing how dependencies are included in the software projects our participants were developing, many noted some interesting facets of corporate software development practices. These facets could be summarized cleanly as processes designed to remove the developer from having to make security considerations during software development.

One such process is a dedicated review team. Many participants, such as P1, noted that "there is a specific team which is set up, which will review the software, check, are there any vulnerabilities or something." An interesting aspect of this response is the "or something" appended to their statement. This demonstrates a lack of clarity into the processes of this

team which could be a successful sign of removing the developer from the loop. The developer no longer has to think about security. Instead all of those thoughts and concerns can be handled by another party. As a result, the developer can focus on the job at hand: developing, not security.

Another such process is an artifact repository. These artifact repositories can do a lot of heavy lifting in terms of making sure developers and build systems are pulling known safe and secure copies of dependencies. P4 stated that "[they] typically would end up adding whatever third party dependency... to some sort of like a software supply chain monitoring like system that they had." These systems can run vulnerability scanners and automate the process of verifying the safety of these dependencies. Then, the dependency would end up in a "secondary repository" that would cache the package for other developers in the company. This also has the added benefit of caching the work of a review team or review system such that a large body of vetted software can be pulled by any developer at any time without engaging the review team and having to wait for approval.

One last process is the pre-installation or provision of vetted software relevant to the developer's work. This includes the pre-installation or provision of software such as IDEs, browsers, and word editors. By supplying the developer with most or all of the software they will need to accomplish their tasks, the company avoids having the developer seek out software themselves, software which either is itself not secure or which comes from an insecure source. To this end, P4 noted that their corporate MacBook had an app store that "only [has] like 20 apps that the company approves" containing software such as a "modified vs code" or "microsoft word".

These processes serve a common goal: alleviate security considerations from the developer. Instead of making the developer examine software for security vulnerabilities, a team dedicated to that task accomplishes the examination for them. Instead of making the developer check to make sure they are pulling dependencies from the correct source (and that the source has not been tampered with), an artifact repository caches known safe copies of the dependency. Instead of making the developer install software they need from unknown sources, either provide a source or pre-install the software for them.

One interesting contrast between companies was found in this respect: the focus of the company. P7 noted that their employer was not technology-focused and lacked a body of expertise that a technology-focused company might have about supply chain attack vectors. As a result, P7 did not have to submit dependencies to a review team or reap the benefits of an artifact repository. P7 did, however, have to discuss changes in dependencies in weekly group meetings which moreso addressed the purpose and necessity of adding or upgrading dependencies as opposed to a security review.

This contrast highlights a gap where software supply chain security processes and software solutions require more technical expertise than is available in many non-technology-focused companies. As a result, we find a need for low-overhead software supply chain security processes and solutions that enable these companies to secure their software supply chain.

4.4 Professional vs. Personal Development

Participants have roughly the same attitude towards security both when developing personal projects and when developing for industry. We saw two schools of thought for security behavior in work environments. Some participants exercised increased caution when performing actions such as installing potentially untrustworthy software, mentioning that they did not want to introduce possible security vulnerabilities. But many other participants felt the opposite; they expressed comfort taking risky actions, relying on corporate security policy such as CVE scanners, dependency graphs, and anti-malware to handle the possible consequences.

Users often forwent recommended behaviors such as verifying trust in the publisher, verifying hashes, using up-to-date software, and checking for known exploits despite mentioning later in the interview that these were indicative of software being trustworthy. In personal environments, most participants expressed similarly lax security habits. Users focused on the end result, with security at the back of their mind. "If it works, you are good to go (P1)". It is concerning how participants justified a lack of security-conciousness in the workplace by claiming others at the company handle it, but turn around and exhibit the same security weaknesses, this time without any potential security support system. Some participants were aware that they failed to follow best-practices, but as P2 said "are you really going to be like expected to check [the hash] every single time?"

One notable exception to this was P6. In 2017, 30 of their virtual and physical machines were compromised by WannaCry. In the wake of the attack P6 said they were much more careful about security, taking measures such as verifying digital signatures, checking files for rootkits, and not placing trust in just one antivirus solution. But even P6 admits that day-to-day they do not always perform the full suite of checks, reserving that energy for "something very important, like an operating system image (P6)".

Dedicated development computers more common in industry. All participants, except P7 who was a special case in which their company was unable to provide them with a dedicated work computer, mentioned that their company provided them with a dedicated work computer. Two participants, P2 and P4, further specified that all of their development was actually done on a remote server via SSH connection on their work computer.

The software permitted on the work participants' work computers varied. Roughly half of the participants were allowed to install anything on their work computers with their

company keeping track of what was installed. For example, P1 said, "you can install anything you want. If it is suspicious, they will flag it and then you have to remove it." The other half of the participants indicated that there was a white list of software that was allowed on their computers. For example, P4 said, "you have a corporate managed mac that that can only have like 20 apps that the company approves." This shows mitigation versus prevention for supply chain attacks. Recording software that gets downloaded is a mitigation technique while only allowing a specific list of software is a prevention technique.

Two participants, P3 and P6, indicated using dedicated computers for their personal projects. The rest of the participants used their personal, daily-use computers. Software that participants mentioned having on their personal, daily-use computers included software development software such as code editors and runtime environments, entertainment software such as videogames and music streaming applications, browsers, and messaging services such as Discord. Participants largely indicated that they go to the official websites for installing software both when installing software for personal use and for work (when it is permitted).

5 Challenges and Limitations

5.1 Communication

One major challenge that all of the interviewers ran into were participants that realized the intent of the interview questions during the interview. This, alongside the desirability bias, resulted in responses that answered what the participant suspected the interviewers wanted as opposed to what the interviewers were directly asking. This was an especially notable issue with P2 who began answering different questions than the ones we asked. Desirability bias was also apparent with another participant who repeatedly asked if they were giving responses we wanted. This was difficult to handle, as any response ("yes" or "no") could affect what the participant thought we wanted to hear and change their responses accordingly.

Another, less common challenge was generally communicating ideas. In an attempt to avoid leading questions, our questions tried to be vague and general to allow the participant to come up ideas on their own. This proved difficult with P7, who misinterpreted what "package" and "dependency" meant and required clarification that might have influenced their responses.

5.2 Validity

The external validity of this study is limited by the population examined. The small sample size of seven software developers combined with each participant's limited professional background means that our results may not generalize to the broader developer population. However, the benefit of our work comes from understanding ideas behind individual experiences, hence the use of qualitative methods as opposed to quantitative ones.

The construct validity of our work is limited by our experience as researchers and coders. The results and analysis of our interviews could likely be improved with a re-coding. Future work should focus on more generalizable and conceptual codes, focusing in on what participants think in a way that can apply to multiple documents. Future work should also perform more preliminary peer coding to create a higher quality codebook.

6 Future Work

From this pilot study, in addition to conducting more interviews to increase the sample size, we came up with some additional research questions that could be investigated in future work. These questions expand on our research to investigate if the perceptions we learned from our participants are actually indicative of trustworthy software.

- "Is software popularity is correlated with security?"
- "How do people determine if a website is "official"?"
- "How do developer perceptions on trustworthiness of software compare to those of security professionals?" Similar to Ion et al [10].

7 Conclusion

We conducted seven semi-structured interviews with UMD students who had experience writing code in industry through internships and full time positions. We performed inductive coding. We found that popularity, usage by personally known people, and verifiability of source were major factors that the participants looked for to determine the trustworthiness of dependencies. We also found that industries tend to assign dedicated groups to ensure security and that participants care roughly the same about security when developing personal projects versus projects for industry.

References

[1] Aadesh Bagmar, Josiah Wedgwood, Dave Levin, and Jim Purtilo. I know what you imported last summer: A study of security threats in thepython ecosystem. *arXiv preprint arXiv:2102.06301*, 2021.

[2] Andy Greenberg Burgess and Matt. The mystery of "jia tan," the xz backdoor mastermind, Apr 2024.

[3] Ramaswamy Chandramouli, Frederick Kautz, and Santiago Torres-Arias. Strategies for the integration of software supply chain security in devsecops ci/cd pipelines, 2024.

[4] Keith Collins. How one programmer broke the internet by deleting a tiny piece of code, Mar 2016.

[5] Anastasia Danilova, Alena Naiakshina, Stefan Horstmann, and Matthew Smith. Do you really code? designing and evaluating screening questions for online surveys with programmers. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pages 537–548. IEEE, 2021.

[6] Fernando López De La Mora and Sarah Nadi. Which library should i use? a metric-based comparison of software libraries. In *Proceedings of the 40th International Conference on Software Engineering: New Ideas and Emerging Results*, pages 37–40, 2018.

[7] Ruian Duan, Omar Alrawi, Ranjita Pai Kasturi, Ryan Elder, Brendan Saltaformaggio, and Wenke Lee. Towards measuring supply chain attacks on package managers for interpreted languages. *arXiv preprint arXiv:2002.01139*, 2020.

[8] Marcel Fourné, Dominik Wermke, William Enck, Sascha Fahl, and Yasemin Acar. It's like flossing your teeth: On the importance and challenges of reproducible builds for software supply chain security. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 1527–1544. IEEE, 2023.

[9] The White House. Executive order on improving the nation's cybersecurity, May 2021.

[10] Iulia Ion, Rob Reeder, and Sunny Consolvo. {"... No} one can hack my {Mind"}: Comparing expert and {Non-Expert} security practices. In *Eleventh Symposium On Usable Privacy and Security (SOUPS 2015)*, pages 327–346, 2015.

[11] Piergiorgio Ladisa, Henrik Plate, Matias Martinez, and Olivier Barais. Taxonomy of attacks on open-source software supply chains. *arXiv preprint arXiv:2204.04008*, 2022.

[12] Enrique Larios Vargas, Maurício Aniche, Christoph Treude, Magiel Bruntink, and Georgios Gousios. Selecting third-party libraries: The practitioners' perspective. In *Proceedings of the 28th ACM joint meeting on european software engineering conference and symposium on the foundations of software engineering*, pages 245–256, 2020.

[13] Marc Ohm, Henrik Plate, Arnold Sykosch, and Michael Meier. Backstabber's knife collection: A review of open source software supply chain attacks. In *Detection of Intrusions and Malware, and Vulnerability Assessment: 17th International Conference, DIMVA 2020, Lisbon, Portugal, June 24–26, 2020, Proceedings 17*, pages 23–43. Springer, 2020.

[14] NTIA Multistakeholder Process on Software Component Transparency. Sbom at a glance, April 2021.

[15] Serena Elisa Ponta, Henrik Plate, and Antonino Sabetta. Detection, assessment and mitigation of vulnerabilities in open source dependencies. *Empirical Software Engineering*, 25(5):3175–3215, 2020.

[16] Tom Preston-Werner. Semantic versioning 2.0. 0. *Web. Available: http://semver. org*, 2013.

[17] Ken Thompson. Reflections on trusting trust. *Commun. ACM*, 27(8):761–763, August 1984.

[18] Dominik Wermke, Jan H Klemmer, Noah Wöhler, Juliane Schmüser, Harshini Sri Ramulu, Yasemin Acar, and Sascha Fahl. " always contribute back": A qualitative study on security challenges of the open source supply chain. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 1545–1560. IEEE, 2023.

[19] Dominik Wermke, Noah Wöhler, Jan H Klemmer, Marcel Fourné, Yasemin Acar, and Sascha Fahl. Committed to trust: A qualitative study on security & trust in open source software projects. In *2022 IEEE symposium on Security and Privacy (SP)*, pages 1880–1896. IEEE, 2022.

A Appendix: Eligibility Survey

UNIVERSITY OF MARYLAND

Block 2

Are you willing and available during the month of November to participate in a 20-30 minute interview?

If Yes, we will contact selected participants by email with a scheduling link.

- ○ Yes
- ○ No

Please provide your email address.

What degree program are you in?

- ○ Undergraduate
- ○ Graduate
- ○ Other

What is your major?

- ○ Computer Science
- ○ Computer Engineering
- ○ Other

What is the highest level CS course you have completed?

- ○ CMSC1XX
- ○ CMSC2XX
- ○ CMSC3XX
- ○ CMSC4XX
- ○ CMSC6XX+
- ○ N/A

Do you have experience writing code in **industry**? (Multi-select)

- ☐ Yes, through internship(s)
- ☐ Yes, through full time job(s)
- ☐ Yes, through part time job(s)
- ☐ No

Which operating system(s) did you use?

- ☐ Windows
- ☐ Linux
- ☐ Mac
- ☐ Other

Have you developed **personal projects** that involve dependencies on third-party libraries or software packages?

- ○ Yes
- ○ No

Which operating system(s) did you use?

- ☐ Windows
- ☐ Linux
- ☐ Mac
- ☐ Other

Block 1

Which of these websites do you most frequently use as aid when programming?

- ○ LinkedIn
- ○ Wikipedia
- ○ Memory Alpha
- ○ Stack Overflow
- ○ I have not used any of the websites above for programming

Code snippet

```
main{
  print(func("hello world"))
}

String func(String in){
    int x = len(in)
    String out = ""
    for(int i = x-1; i >= 0; i--){
        out.append(in[i])
    }
    return out
}
```

What is the parameter of the function "func" above?

- ○ String out
- ○ String in
- ○ int i = x-1; i >= 0; i–
- ○ Outputting a String
- ○ int x = len(in)

Which of these values would be the most fitting for a Boolean?

- ○ Small
- ○ Solid
- ○ Quadratic
- ○ Red
- ○ True

Consent Form

Please read the consent form and fill out the acknowledgment at the bottom of the page.

CONSENT TO PARTICIPATE

Project Title

Developer Perceptions on Trusting Software

Purpose of the Study

This research is being conducted by Anders Spear, Tien Vu, Ryan ElKochta, and Michael Suehle at the University of Maryland, College Park. The purpose of this research project is to understand student developer perceptions on the sourcing, installation, and security of online software packages. We want to gain insight into the level of awareness developers may have about supply chain attack vectors.

Procedures

As a participant in this study, you will first be asked to fill out this consent form. Following that, you will complete a short pre-interview survey, which will take approximately 5 minutes. The survey will ask for information about your background with software development.

At the scheduled time, we will conduct a video call or in-person interview which will focus on your experiences and processes trusting and installing online software packages. The interview will take approximately 20-30 minutes and will be audio-recorded (sound only, no video). Audio recording is required to ensure accurate transcription and analysis of your responses. If you do not consent to being recorded, you are welcome to stop participating in the study, as the recording is necessary for further analysis.

During the interview, you will be asked about software projects you have worked on in the past, from internships and personal projects. The questions will focus on your use of third party packages and your perspectives and processes you employ during installation of these packages.

Potential Risks and Discomforts

There are no known risks associated with participating in this study. However, you may feel uncomfortable sharing certain details about your work or personal experiences. You are not obligated to answer any questions that make you uncomfortable, and you are free to skip any questions or withdraw from the study at any time without penalty.

Potential Benefits

While there may not be direct benefits to you personally for participating in this study, your insights will contribute to a broader understanding of the role of third party software during development, benefiting the wider software development and security communities. You may also gain some personal insight into your software installation process through the reflective nature of the interview process.

Confidentiality

We take your confidentiality very seriously and will implement several measures to protect your privacy. We will not publish or share your name, organization, or any other identifying information. All personal information and data collected during the interview will be anonymized, and your identity will not be linked to any specific responses.

The interview will be conducted via Zoom or in person at UMD, and audio will only be recorded for transcription purposes—no video will be recorded. Audio recordings will be stored locally in a password-protected folder and will be deleted immediately after they are transcribed. The audio recordings will be transcribed using OpenAI Whisper. Transcriptions will be securely stored in Google Drive, an account-protected cloud storage service.

Pseudonyms will be used in place of names and other direct identifiers. A key linking pseudonym to contact information will be stored separately from the anonymized data and will be destroyed at the conclusion of the project. All recordings and identifying data will be permanently deleted at the end of the study. Only authorized members of the research team will have access to the raw data during the study.

If we write a report or article about this research project, your identity will be protected to the maximum extent possible. Your information may be shared with representatives of the University of Maryland, College Park, or governmental authorities if you or someone else is in danger or if we are required to do so by law.

Compensation

You will receive $25 in cash from one of the Principal Investigators following the interview. You will be responsible for any taxes assessed on the compensation.

For remote participants, you will need to meet with one of the investigators at the University of Maryland, College Park campus to receive this compensation.

Right to Withdraw and Questions

Your participation in this research is completely voluntary. You may choose not to take part at all. If you decide to participate in this research, you may stop participating at any time. If you decide not to participate in this study or if you stop participating at any time, you will not be penalized or lose any benefits to which you otherwise qualify.

If you decide to stop taking part in the study, if you have questions, concerns, or complaints, or if you need to report an injury related to the research, please contact the investigator:

Anders Spear
Iribe Center, 8125 Paint Branch Dr
College Park, MD 20742
aspr@umd.edu

Participant Rights

If you have questions about your rights as a research participant or wish to report a research-related injury, please contact:

University of Maryland College Park
Institutional Review Board Office
1204 Marie Mount Hall
College Park, Maryland, 20742
E-mail: irb@umd.edu
Telephone: 301-405-0678

For more information regarding participant rights, please visit: https://research.umd.edu/research-resources/research-compliance/institutional-review-board-irb/research-participants

This research has been reviewed according to the University of Maryland, College Park IRB procedures for research involving human subjects.

Statement of Consent

By checking the boxes below, you indicate that you are:

- At least 18 years of age
- You have read this consent form or have had it read to you
- Your questions have been answered to your satisfaction and you voluntarily agree to participate in this research study.

You may download the consent form for your records.

Consent form.docx

If you agree to participate, please check all of the boxes below.

- ☐ I am 18 years or older.
- ☐ I have read the above consent form.
- ☐ I voluntarily agree to participate in this study.
- ☐ I agree to the use of audio-recorded interview information.

B Appendix: Semi-structured Interview Outline

Note: The following questions and messages represent the general structure and content of the interview, but may not be asked verbatim. The phrasing or order may be adjusted to fit the flow of the conversation or specific context.

- Thank you for taking the time to participate in our research project!
- Inform participant about study and get consent
  - We are conducting interviews for Michelle Mazurek's CMSC732 Human Factors in Security and Privacy Course and are looking to investigate how software developers use open source software.
  - Do you consent to your responses in this interview being recorded and used for this project? We will delete the recordings after we transcribe them.
- In the eligibility survey, you noted that you developed code in industry.
  - Can you talk me through what languages you used?
  - Did you use third party packages? Can you name a couple or state their purpose?
  - Did you install them? If so, how did you install them?
    - Was there a corporate procedure for adding third-party packages to projects? How does it work? Take care not to divulge anything too specific, though we can always redact that information later if necessary.
  - Can you walk us through the process in detail? What do you think about when choosing which packages to download and how to download them?
  - Did you take any steps to verify that the packages were safe to use? Safe to use being non-malicious or non-compromised.
    - If so, what?
  - Did you have any challenges installing them?
  - Was development performed on a personal computer or a dedicated work computer?
  - On said computer, what software or what types of software did you install? Disregard software already installed (e.g. Teams, Outlook). This can include browsers, IDEs, or personal entertainment (i.e. Spotify).
  - How did you install said software?
  - Can you walk us through the process in a bit more detail? What do you think about when choosing what to download and how to download them?
  - Did you take any steps to verify that software was safe to download and use? Safe to use being non-malicious or non-compromised.
- In the eligibility survey, you noted that you developed code for personal projects. For your most recent personal project:
  - Can you talk me through what languages you used for that?
  - Did you use any packages? Can you name a couple or state their purpose?
  - Did you install them? If so, how did you install them?
  - Can you walk us through the process in detail? What do you think about when choosing which packages to download and how to download them?
  - Did you take any steps to verify that the packages were safe to use? Safe to use being non-malicious or non-compromised.
    - If so, what?
  - Did you have any challenges installing them?
- For the following questions, answer them for your most recent personal project:
  - Was development performed on a personal computer or a dedicated development computer? On either, was Docker used?

  - What software or what types of software did you install? This can include software used for other activities on the device, such as school, work, or personal entertainment.
  - How did you install said software?
  - Did you take any steps to verify that software was safe to download and use? Safe to use being non-malicious or non-compromised.
- What makes a software package trustworthy?
- Do you think that the security of one software can impact other software on the device? Why?
- Do you think that the security of one software can impact software developed on the device? Why?
- Thank you for your time!
- Coordinate payment with them.

C Appendix: Codebook

Category	Secondary Category	Final Code	Primary Code	Files	References
Perceptions	Can security of software impact security of other software on the same device	not thought about	I assume corporations have some mechanism of preventing it.	2	2
		Yes: provided an explaination of an attack that could happen	explaination: can cause data breaches	1	1
			explaination: package compromizing system and creating a back door	1	1
			explaination: modify a tool in the chain of development	1	1
			varying difficulty	1	3
			explaination provided: worried about files being transferred to server	2	2
		Yes: provided specific example of an attack	explaination: example: CrowdStrike.	1	1
			explaination: reading about virus	1	1
			explaination: provided example of XZ	1	2
			explaination: was a victim of a ransomware attack	2	2
	Potential attacks mentioned	Impersonating legitimate package		1	1
	What makes software trustworthy	Reputable Source	affiliated with large institution (such as university)	1	1
			Comes from correct source	2	2
			digital signature	1	1
			find the official source	2	2
			If the developers are trustworthy	1	1
			Microsoft Certificate	1	1
			name difference	1	1
			Open source	2	2
		Heuristic Checks	checked by antivirus	1	1
			asks for permissions	1	1
			download size	1	1
			file extension	2	2
		Approved by others	people participant knows deem it trustworthy	1	1
			audits from website	1	3
			approved by package manager repo	1	1
			check for negative reviews online	1	2
		Popularity	mentioned in forums: Hacker news	1	1
			mentioned in forums: Reddit	1	1
			number of downloads	2	2
			github stars	1	1
		Well-Maintained		2	3
			would check issues	1	1
Personal Projects	Issues when installing packages	Issues when installing packages\Dependency version conflicts		1	1
	Languages	C++		2	3
		OCaml		1	1
		Python		4	5
		Rust		1	1
		TypeScript		1	1
	OS	Linux	arch	3	5
			fedora	1	1
			NixOS	1	1
			Ubuntu	1	1
		Mac OS		2	2
		Windows	Windows 10+	2	4
			Windows XP	1	3
	Package Related Information	Package Managers	BUN	1	1
			cargo	1	1
			Conan	1	2
			Conda	2	3
			Dune	1	1
			NPM	1	1
			pip	3	4
		Package Sources	conda-forge	2	2
			GitHub	1	1
			OCaml official repos	1	1
			PyPi	3	3
			user repository helper	1	1

	Packages	arkworks	1	1
		Beautiful Soup	1	1
		crow	1	1
		Flask	1	1
		Flutter	1	1
		Jinja	1	2
		Mongo	1	1
		Next.js	1	1
		pytorch	1	1
		react	1	1
		Tailwind	1	1
		Tailwind CSS	1	1
		Webpack	1	1
Procedures for installation	curl bash		1	1
	look at GitHub source code and download		1	1
	Not reading the PKDBUILDs		1	1
	not using official OS app store		1	1
	Official website		3	4
	software facilitates installation	Installing games through Steam	1	1
		repository helper	1	1
	Told how to do it	Told what to install (for e.g. class)		
		Google	1	1
	uses a backup environment		1	1
	look up in a binary cache		1	1
Software Installed	Adobe Acrobat		1	2
	alt tab		1	1
	browser		4	4
	Cisco VPN		1	1
	For code development	Android Studio	1	1
		Eclipse	1	1
		IntelliJ	1	1
		Neo vim	1	1
		VS	1	1
		vs code	1	1
		Zed	1	1
		docker	1	1
		cargo	1	1
		run times	1	2
		Shadcn	1	2
		sp1	1	1
		Hyper-V	1	1
		Homebrew	1	1
	Entertainment	Cracked Video Games	1	1
		Games (on a different PC)	1	1
		Spotify	1	1
		Steam	2	3
		Torrenting	1	1
	Discord		1	1
	Microsoft Office		1	2
Software Sources	Distribution Repository		2	2
	Docker Hub		1	1
	Official Website		1	1
	Torrents		1	1
Verification that software is safe	Source Verification	digital signature\check the hash	2	3
		official website	1	1
		scanning website	2	5
		manually confirming the SSH	1	1
		use official repositories	2	2
	Software Verification	antivirus	1	1
		Ensures the software doesnt ask for extra permissions	1	1
		make sure the developer actively fixes issues or bugs	1	1
		Manually verify that the files look normal (ex. not a rootkit)	1	1

			tries in a dev or backup environment	1	1
			recent	1	1
		None		5	6
		popularity		2	3
			Stack Overflow	1	1
			standard packages	2	2
			used tutorials\early in google search list	1	1
			used in other project	2	2
			well-known packages	3	4
		what they said they did NOT do	did not check new hashes for updates	1	1
			didnt read PKGBLDs	1	1
	Where development was done	Dedicated Development computer		2	3
		Personal Computer		5	5
		VM on personal computer		1	2
Professional	Build Process	Docker		0	0
		Manually from source		1	1
		podman		1	1
	Issues while installing packages	Company infrequently updated dependencies		1	1
		Compatibility between dependency versions		3	5
		Toolchain version(s)		1	1
	OS	Linux	Unspecified	2	4
			Debian	1	2
		Mac OS		3	3
		Windows		1	1
	Package Related Information	Package Managers	Brazil	0	0
			built-in Go package manager	1	2
			Cargo	1	1
			Conan (C++)	1	3
			Conda	1	1
			NPM	2	3
			PIP	2	4
			pypy	1	1
			Redux	1	1
			System package manager	1	2
		Package Sources	internal artifactories	2	5
		Third Party Packages	Catch2	1	1
			Crow	1	2
			doctest	1	1
			Emphasis on internal dependencies	1	1
			FastAPI	1	2
			FFMPEG	1	1
			Flask	1	2
			Go-Chi,	1	1
			Internal Packages	1	1
			lib sodi	1	1
			Librosa	1	1
			Math	1	7
			Mocking Libraries	1	1
			OCR libraries	1	1
			OpenCV	1	2
			OpenSSL	1	1
			Pandas	1	2
			property based testing libraruies	1	1
			PyTest	1	2
			range v3	1	1
			React	2	3
			Redux (JS)	1	1
			SaltStack,	1	2
			Soundfile	1	1
			spdlog	1	2
			WebRTC	1	1
	Procedure for installation	White List	Allowlist	1	1
			bootstrapper	2	2
			coporate managed app store	1	1
			They were already decided.	3	6

	Participant Installs themself		5	8
		loose to no rules	4	5
		official website	4	5
		Bash Curl	1	2
		downloads recorded	1	1
		Online tutorial	1	3
	corporate antivirus or precautions		1	1
		discouraged to install on your own	1	1
		must be open source	1	2
		Some secure environments, some less restrictive environments	1	2
	Approval or Security Team		1	2
		platform in the company for requesting	1	1
		secondary repository that you would check in your third party dependencies	1	2
Programming Languages	C		2	3
	C#		1	2
	C++		4	6
	Go		2	3
	Java		2	2
	JavaScript		1	2
	MATLAB		1	2
	Python		6	10
	Rust		1	1
	Scala		1	1
	Spark		1	1
	TypeScript		3	5
Software Installed	For development	APT GAD	1	2
		clan	1	1
		Docker	1	1
		G++	1	1
		GCC	1	1
		Intellij	1	1
		Neovim	1	1
		Visual Studio Code	1	1
		nvidia drivers	1	1
	Pre-installed software by company		2	6
		bootstrapper	1	1
	browser		2	3
	entertainment software		2	2
		Spotify	1	2
	Misc	Torrenting	1	1
		Ink Space	1	1
		VPNs	1	1
		Wine	1	2
		Word	2	2
	No Personal Software Allowed		1	1
Software Sources	Distribution Repositories		1	1
	Github		1	4
	Homebrew		2	4
	Official Website		2	4
	University		1	1
Where was development done	Dedicated work laptop	Dedicated work laptop	6	10
		Docker	1	1
		SSH or Remote into other machine	2	3
		WSL	1	1
	Personal computer	Personal Computer	1	2
		Remote into server from personal computer	1	1