The government seeks individual input; attendees/participants may provide individual advice only.

Middleware and Grid Interagency Coordination (MAGIC) Meeting Minutes[1]
March 2, 2022, 12-2 pm ET

Virtual

Participants

Audris Mockus (University of Tennessee, Knoxville)	Mallory Hinks (NCO)
David Bernholdt (ORNL)	Mike Heroux (Sandia)
Birali Runesha (U of Chicago)	Rich Carlson (DOE/SC)
Kate Evans (ORNL)	Sean Peisert (LBNL)
Kathy Austin (Texas Tech University	Stefan Robila (NSF)
Ludovico Bianchi (LBNL)	Terence Sterba (Georgia Tech)

Introductions: This meeting was chaired by Rich Carlson (DOE/SC) and Tevfik Kosar (NSF)

Trusted CI Evaluation, Guidance, and Programs for Assurance of Scientific Software

Sean Peisert, LBNL

- Trusted CI: The NSF Cybersecurity Center of Excellence
  - Mission: to lead in the development of an NSF Cybersecurity Ecosystem with the workforce, knowledge, processes, and cyberinfrastructure that enables trustworthy science and NSF's vision of a nation that is a global leader in research and innovation.
  - Sean said he has been involved for a few years now and last year he led a focused effort on the assurance of scientific software
- 2021 Software Assurance Annual Challenge
  - Involved Trusted CI members from 5 different organization
  - Goal: to broadly improve the robustness of software used in scientific computing with respect to security
  - Interviewed 6 different teams developing scientific software and compiled the findings into a report
  - Developed a guide to help scientific software developers begin bridging secure software gap
- Findings: Positive notes

[1] Any opinions, findings, conclusions, or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Networking and Information Technology Research and Development Program.

    - All projects used:
        - Code repositories
        - Software version control
        - Bug/issue tracking software (E.g., Jira, RT, and GitHub)
        - Standard libraries for cryptography
    - Most of the projects used:
        - Modern languages, eschewing older languages like C and C++
        - Dependency tracking tools
- Findings: Some concerns:
    - Challenges across the board in:
        - Software management
        - Organization/mission
        - Tools
            - Static analysis
            - Dynamic analysis
            - Dependency tools
        - Testing
        - Training
        - Code storage and management
- Management Process/Organization
    - Often a significant lack of documentation. Out of date documentation. Developers ignoring standards and requirements
    - General lack of point of contact for software security concerns
    - Many projects don't believe they would benefit from leveraging organizational IT resources
- Tools and Testing
    - Static analysis tools: use is limited
    - Dependency tools: used frequently, but not effectively
    - Wide use of manual testing, but limited automated testing
    - Most use correctness testing, but do not consider security testing
- Training Observations
    - Did not find training useful
    - Difficult to find training focused on the areas where there is a specific need
    - Importance not understood
    - Thought training is expensive
- Code Storage Observations:
    - Projects missing a formal review process for repository updates
    - Unconstrained use of third-party libraries and packages
    - Little explicit risk assessment in the use of third-party code
- Best Practices: Organization
    - Identify and appoint a cybersecurity lead
    - Involve leadership in cybersecurity decision making
    - Principle of least privilege

    - Documentation is vital and must be kept up to date
- Best practices: Training
    - Internally created: locally created courses/materials
    - Bringing in outside trainers
    - Tutorials at conferences and workshops
    - Professional tutorials
    - University classes: leverage local resources
    - Free/open software security training from Trusted CI
- Best practices: Tools
    - First step: use dependency analysis tools
    - Second step: use static analysis tools
    - More advanced: use dynamic analysis tools
- Best practices: Code Releases
    - Use centralized version control
    - Separate testing branches from release versions
    - Separate feature releases from security releases
        - Reduce "update hesitation" by assuring users that security updates won't break their setup
- 2022 Trusted CI Software Assurance Plans
    - Secure Software Guide
        - Expand and refine the content
        - Work with community to get broad adoption
        - Get feedback and contributions from the broader community
    - Teaching and Training Software Security
        - Finish first pass on online video content
        - Expand coverage
        - Continue tutorials
    - In-depth vulnerability assessment
        - Continue research into automation our methodology
    - Threat models for ransomware
        - Developing comprehensive threat model for ransomware attacks

<u>Questions</u>

- Kate Evans: You mentioned earlier that people don't realize that their software will become important later, so they haven't done a lot of these things up front. How do you, as a scientist or software developer, address that early stage when you don't know if it's a thing that will continue?
    - Sean: It's probably unrealistic to have everyone write some comments in their code, just in case. But it would be great. However, a day of reckoning needs to come before you've written 10,000 lines of code. You can get some help or advice on how to do this.
- Mike Heroux: A lot of our scientific software is executed in an environment that has limited exposure to external vulnerabilities like on a large cluster or supercomputer or

your laptop. It doesn't have a lot of interaction with the outside world. I'm curious which of the security vulnerabilities you looked at are appropriate for that.
    - Sean: I agree, if it's just a Jupiter notebook run on your laptop, it's unrealistic and not the primary focus. However, it would still be a good idea. There is a lot of software development by the community that interacts with databases. It used to be that our HPC was entirely isolated, but it's not anymore. And now we have closed loop simulations running on HPC that interact with the control devices and the fields.
    - Mike: For the pieces of software that don't have to interact with the broad internet, what are the things you should still do? What are the high priorities?
    - Sean: You should still have the appropriate access controls for your commits. You should still be ensuring that you understand something about the reliability of any third-party libraries. I'd like you to write stuff using the software language that doesn't require you to directly address memory.
        - Mike: It's not always up to us to do that.
- Stefan Robila: Related to the first part, where you engaged several teams or groups that are working on software, did you notice any differences in size? Were there any challenges if the team was more distributed? In such cases it becomes a little different in how you will try to tackle security.
    - Sean: We definitely talked to multi-institution teams. FABRIC was an example. I don't think there's a one size fits all answer. These institutions have a variety of resources that they can use.
- Birali Runesha: When you think about the graduate student, the aspect where they are trying to focus on getting the result and publishing. Every time we've approached them and try to encourage them to follow these best practices, they see it as slowing down the process. How do you approach that?
    - Sean: There's a distinction between software developed for an experiment vs something that is infrastructure. Sometimes experimental software becomes infrastructure and therein lies the problem. The quick answer is don't use grad students for infrastructure. Could also explain the importance of best practices and suggest that they build a reputation for professional quality code.

<u>World of Code: Enabling a Research Workflow for Mining and Analyzing the Open-Source Supply Chains</u>

Audris Mockus, University of Tennessee

- Audris mentioned that this is a short version of the talk, there is a longer version.
- World of Code: Enabling Research on OSS Supply Chains
    - Audris said this slide was about trying to find collaborations or somebody who might actually be interested in using this resource.
    - One of the useful concepts is called software supply chains. Different from regular supply chains. Not well defined.
- SSC of the first kind: "technical"

    - Technical dependencies among projects with change effort as product flow: upstream effort either reducing or increasing the need for downstream effort
    - Risks: unknown vulnerabilities, breaking changes, lack of maintenance, lack of popularity
    - Measure: no way to tell how many projects use a package/library
    - Examples
        - Python: import re
        - Java: import java.util.collection
        - JavaScript: package.json
- SCC of the second kind: “copy”
    - Copying of the source code from project to project flow (22% of files copied to, on average, 14 other repositories)
    - Risks: license, unfixed bugs, no new functionality
    - Measure: no way to identify what has been copied
    - Examples
        - Implementation of a complex algorithm
        - Useful template
        - Build configuration
- SSC of the third kind: ‘knowledge”
    - Knowledge (product) flow through code changes as developers learn from and impart their knowledge to the source code
    - Risks: developers may leave, companies my discontinue support, bad practices my propagate
    - Measure: no way to know what projects a developer worked on
    - Examples:
        - Developers gaining skills with tools/packages/practices
        - Developers spreading practices, e.g., testing frameworks
- How to measure Software Supply Chains
    - Need all versions of all source code
        - Discover all git repositories
        - Clone/download these project repositories
        - Extract/correct necessary data
        - Construct SSCs
    - Noted that the amount of open source code grows about 20% in 6 months
- Discover
    - Forge discovery
    - Within-forge discovery
    - Compare to other efforts
- Storage/retrieval tricks
    - Impossible to store all 50 PB
        - De-duplicated: 250 TB
    - Cannot transfer 50 PB
        - Do only updates: use last commit

        - Approximately 200 TB/quarter
- Audris summarized World of Code status.
- Apart from SSCs
    - Improve research quality e.g.,
        - Avoid convenience sampling
        - Account for network effects
        - Conduct natural experiments
        - Increase productivity by sharing curated data
    - Build tools for FLOSS developers
    - Inform Enterprises
- VDiOS (Vulnerability Detection in Open Source)
    - "Orphan vulnerabilities"
        - Vulnerabilities in copied code that still exist in a project after they are discovered and fixed in another project
        - Code is copied: no package manager
        - Link to the original code no longer available
        - An overlooked part of the software supply chain
    - Orphan vulnerability risks
        - An exploit may be available
        - Projects may not be aware of the vulnerabilities
        - Applications may be built by unsuspecting users
        - OSS may suffer reputational damage as a dump low quality code
    - Showed schematic of VDiOS architecture
- VDiOS operation
    - Given a vulnerability fix in on project, using WoC identifies all other repositories that
        - Still contain the vulnerable code
        - Used to contain the vulnerable code, but have now been fixed
        - Used to contain the vulnerable code, have been changed, but we do not know if the change fixed the vulnerability
    - Key Benefits
        - Inform maintainers and users of still vulnerable projects about the risks of the vulnerability in their code
        - Warn users that contemplate reusing such code about the unpatched vulnerabilities

Questions

- Rich Carlson: How extensive is your World of Code? You said you were trying to get everything out of GitHub?
    - Audris: Oh, it's much more than that. It's about 54 Gits.
    - Rich: And you're cloning those and looking through those for vulnerabilities of style?
    - Audris: Right. The first stage is becoming aware of the code. Second is getting the data for these repositories. And then there's the next stage where we build

various applications, including curation. So, extract which APIs it uses, who are the authors of the code, how the copying between projects happens, linking it with vulnerability data, linking it with bug data.
    - Rich: And then this information is fed back to the code developers?
    - Audris: Well, it depends. There could be tools of submitting batches. There are some other tools to measure the risk of trusting the pull request. You can use the data to establish reputation or experience of the commit of that pull request author to increase trust.
- Rich: Where would you find more information about this work?
    - Worldofcode.org
    - Bitbucket.org/swsc/overview
    - Next hackathon
        - woc-hack.slack.com
        - Sign up at http://bit.ly/WoC-Hack
        - github.com/woc-hack/tutorial
        - Signup form http://bit.ly/WoCSignup
- Ludovico Biachi: I just wanted to say that it looks super cool. I'm going through the paper on the website. It look like a very nice source of data.

Roundtable

- Mallory announced that she will be having a baby at the end of May and there would be a temporary coordinator in place for while she is on maternity leave.

Next Meeting

April 6 (12 pm ET)