Presenting the Results of the Trusted CI Annual Challenge on Software Assurance

Elisa Heymann, Barton Miller, and Sean Peisert
Trusted CI Webinar
February 28, 2022
The Trusted CI Annual Challenge on Software Assurance was a year-long project in 2021. The Challenge involved Trusted CI members from five different organizations.

The goals of the challenge were to *broadly improve the robustness of software used in scientific computing with respect to security*.

In the first part of the year, the Trusted CI team *interviewed six different teams* developing scientific software, and compiled the findings from those interviews into a report.

In the second part of the year, the Trusted CI team *developed a guide* to help scientific software developers begin bridging secure software gap.

The Trusted CI Guide to Securing Scientific Software is intended to be a *living document* that evolves as the field evolves and expands to cover more topics as time permits.
The State of the Scientific Software World: Findings of the 2021 Trusted CI Software Assurance Annual Challenge Interviews

September 29, 2021
Status: Final Report v1
Distribution: Public

Andrew Adams, Kay Avila, Elisa Heymann, Mark Krenz, Jason R. Lee,
Barton Miller, and Sean Peisert

Guide to Securing Scientific Software

December 14, 2021
Status: Final Report v1
Distribution: Public

Andrew Adams, Kay Avila, Elisa Heymann, Mark Krenz, Jason R. Lee,
Barton P. Miller, and Sean Peisert

https://www.trustedci.org/software-assurance
Findings
First, Some Positive Notes

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

- Modern languages (e.g., Java or Python) eschewing older languages like C and C++
- Dependency tracking tools
And now, some concerns ...

What were the organizational postures and directions on the use of:

- Software management process
- Organization/mission
- Tools
- Use of static analysis tools
- Use of dependency Tools
- Testing
- Training
- Code storage
- Newer cybersecurity techniques
Threat Modeling

A structured approach to conceptualize and consider security threats to the software before coding.

Observations: In most of the scientific software projects that we evaluated, we did not observe threat modeling being performed, either formally or informally.

Documentation

Documentation for: process and policy (e.g., onboarding and offboarding of developers); who is allowed to submit and approve commits; code and development standards; and design documents.

Language Choice

Avoid use of legacy languages, especially C, C++ and PHP. There are just too many pitfalls to avoid.

Observations: While there is widespread use of better languages, there is still too much risky code in use.

Documentation

Documentation for: process and policy (e.g., onboarding and offboarding of developers); who is allowed to submit and approve commits; code and development standards; and design documents.

Organization and Mission

Point of contact for software security issues

Need someone responsible for managing and-or coordinating security matters. Additionally, can benefit from project management support.

Observations: General lack of such personnel.

Repository management

Avoid the use of multiple independent repositories on a single project and have a policy for coordinating management and control of the single repository.

Observations: Multiple repositories existing and, even when they do not, there are often too many people exerting control over the single repository.
Leveraging organizational IT resources

Your organization may have audit and review resources that can help improve your project’s security.

Observations: Many projects believed that they would not benefit from additional resources and chose to go it alone.
Static analysis tools

Given the large variety of tools available, for almost any programming language, these are an essential component of any software development life cycle.

Observations: Use of tools is limited and ability to interpret what the tools are reporting is weak. There are limited resources available to help with this issue.

Dependency tools

Dependency tools are essential for evaluating software supply chain issues and developing a SBOM. There is a growing number of effective tools.

Observations: There is wider use of these tools than static analysis tools, but often a misunderstanding of how these tools work and how to use them effectively.
Test suites and automated testing

Test suites and automated testing (regression testing) is crucial to the stability of your code base.

Observations: Wide use of manual testing techniques but limited automated testing.

Security vs. functional testing

Testing should include both functionality (correctness) and security characteristics.

Observations: Most teams understand correctness testing but do not consider security testing.
Testing

Independent testing

Testing is most effective when it is done by a group independent of the software developers. This avoids blind spots and conflicts of interest.

Observations: Few organizations have independent testing groups or cross testing between different development teams.
Training is key to building a team that will produce secure software.

Observations: Too many obstacles to wide adoption of a software security training program.

• Did not find training useful at all, or are unable to find appropriate training.
• Thought training is expensive because
  1. too costly to have developers offline attending several-day long training sessions and
  2. companies offering training charge several thousand dollars for a few days of training.
• Difficult to find training focused on the areas where they have a need.
• The importance of training is not fully understood.
Code repositories are an key resource for any development project. For these repositories to be effective, there must be well established procedures in its use and disciplined adherence to these procedures. Separate branches (and releases) for security updates are essential.

Observations:

• Many projects missing a formal review process for repository updates. Such a review should include style, functionality, and security checkoffs.
• “Dependency hell”: unconstrained use of third party libraries and packages.
• Little explicit risk assessment in the use of third-party code. (Though good use of dependency tools.)
Adoption of Modern Cybersecurity Practices

Recent advances in security can make substantial improvements to a project’s security.

Observations:

- Two factor authentication was used when required (such as github) but avoided when not required (such as for local VPNs).
- Aware of the need to securely store passwords and keys but generally avoided password managers.
Threats
Trust Boundaries and Attack Surfaces

What do you control?  What do you not control?  
What do you trust?  What should you not trust?

These are the things that you do control

Trust: The code that runs on your Web server

These are the things that you don’t control

Don't trust: The Javascript that runs in your browser
Trust Boundaries and Attack Surfaces

What do you control? What do you not control?
What do you trust? What should you not trust?

Web Browser

Web Server

Business Logic

Database Server

Attack Surface
Categorizing the Threats

**Threat:** “Any circumstance or event with the potential to adversely impact . . . via unauthorized access, destruction, disclosure, modification of information, and/or denial of service.” (NIST)

<table>
<thead>
<tr>
<th>Exploiting Humans</th>
<th>Exploiting Software</th>
<th>Exploiting Protocols</th>
<th>Exploiting Supply Chain</th>
<th>Insecure Design</th>
</tr>
</thead>
<tbody>
<tr>
<td>Phishing, spear phishing</td>
<td>Injections, Buffer overflows, permissions, web (big topic), guessing &amp; brute force.</td>
<td>Replay, password attacks, sniffing.</td>
<td>Attack on unidentified or unassessed software dependencies.</td>
<td>Design flaws vs. implementation flaws, threat modeling.</td>
</tr>
</tbody>
</table>
Best Practices
Organization

Identify and appoint a cybersecurity lead (Framework Must 7)

Organizations must establish a lead role with responsibility for securing software design and development process, to advise leadership and stakeholders of potential risk at any point in the software’s life-cycles.

Involve leadership in cybersecurity decision making (Framework Must 5)

Dialogue between a security lead and leadership is essential for projects as a whole. Leadership should not be surprised about risks.

Project managers are vital help for managing secure software development

Developers can get caught up in the coding process and forget about deadlines. Adhering to a plan in the software’s life-cycle reduces the chance of pitfalls. A project manager can help.

Principle of least privilege - limit access and rights within the project according to the needs and responsibilities of individuals’ positions

Access and rights should be set according to the needs and responsibilities of individuals’. Protect the service and dependent systems. Protect individuals from inadvertently causing damage to the system.
Leadership must instill a *culture of security* in a way that it is not seen as a burden

Lack of a security culture can come from:

• Not having needed security controls in the past, but being successfully despite that.
• Having a negative experience with a security measure or policy.

Documentation is vital, and must be kept up to date

Process and policy documentation, such as onboarding and offboarding of developers.
Policies regarding who is allowed to submit and approve commits

What process must be gone through before approval is granted.
Code and development standards, and design documents.
Training

Training is essential and do not leave it to chance. The goal is to find a strategy that works for your organization.

Set expectations and policy for training.

There are lots of options.

• Internally created: Locally created courses and online materials.  
  Best suited for larger organizations.

• In house: Bringing outside trainers into your organization.  
  Can be good but expensive but can be tailored to your needed. (Talk with Trusted CI)

• Conferences & workshops: Many conferences and workshops offer tutorial programs.  
  Can require travel and quality varies. Some conferences (e.g., SC) maintain high standards.

• Professional tutorials: Similar to in house.

• University classes: Leverage your local resources.  
  Requires a larger time commitment but can be comprehensive.
Free and open software security training resources, sponsored by Trusted CI:

https://research.cs.wisc.edu/mist/SoftwareSecurityCourse/

Also join us for in-person tutorials at meetings or in-house.
Training

How do you proceed?
1. Perform a self-assessment to understand the needs of your organization.
2. Have a plan to overcome the different hurdles.
3. Set realistic expectations.
4. Take action.
Software analysis tools find flaws in a program and increase the security of your software.

**First step: Use Dependency analysis tools**

Detect publicly disclosed vulnerabilities present in a project’s dependencies.

Examples:

- **Snyk**: Java, .NET, Python, Ruby, PHP, Node.js, C, C++, Go, Swift
- **Dependabot**: Ruby, JavaScript, Python, PHP, Elixir, Elm, Go, Rust, Java and .NET
- **OWASP Dependency Check**: Java, .NET, Python, Ruby, PHP and Node.js
Tools

Second step: Use Static analysis tools

Scan a program’s source, byte, or binary code in the search of problems. Can find buffer overruns, injections, cross site scripting, cross site request forgery, improper input validation, path traversal errors, hard coded credentials, serialization errors, ...

Use them from day one!

Examples: SpotBugs (Java), Snyk (Java, Javascript, TypeScript, Python, PHP, C# and Go), and many others.
Second step: Use Static analysis tools

Don’t worry if you didn’t see your language in the list. There are tools for almost any language you can imagine.
More Advanced: Use Dynamic analysis tools

They monitor a program’s execution to detect execution-time errors such as memory leaks and races.

Identify dynamic analysis tools for the languages that you are using in the GitHub list of curated tools.

https://github.com/analysis-tools-dev/dynamic-analysis
Code Releases

Use **centralized version control**

Git is the most widely used. GitHub and GitLab are also popular. All are good choices.

Use **two factor authentication** and maintain **access control** on code repositories

A code repository should only allow changes by authorized users,

Separate **testing** branches from **release** versions

Users need to be assured of stability of code. Developers should commit code to separate working branches until sufficient testing has been done on the changes to merge into the main code base, identified through a version change.

Separate **feature** releases from **security** release

Releases that address security issues should be separate from feature versions
Will reduce “update hesitation” by assuring users that security updates won’t break their setup
Vulnerability Management

Ensure robust and verifiable (e.g., PGP signed) community communication methods:

- Issue tracking built into GitHub
- Web site
- Mailing lists
- Social media

Common Vulnerabilities and Exposures (CVEs), captured in the National Vulnerability Database (NVD), are an excellent way of broadly distributing information about vulnerabilities.

Register your organization as a CVE Numbering Authority (CNA):

https://www.cve.org/PartnerInformation/Partner

When a vulnerability is not assigned a CVE, it may appear in other registries such as the NPM Security Advisory database.

Keep critical security issues private until the project is ready for them to be public.

Make it easy for users to stay up-to-date with security fixes through a variety of means.

Users should subscribe to a mailing list for critical and timely announcements.

New releases should also be announced in a highly visible part of the website.
Cryptography can help protect the confidentiality and integrity of data.

Nearly all programming languages and development environments provide cryptographic functions.

  Training materials, both written and video, exist to educate the programmer in their chosen language.

Know when to use cryptography.

  Cryptographic functions can be resource-intensive. In situations with very low power availability or very large datasets, encryption may require too much power, computing resources, or bandwidth.

Use widely-used cryptographic libraries — do not create your own algorithms!

  Writing cryptographic code is hard — it takes years of peer review before it can be trusted.

Errors in implementing cryptography lead to the same failures that it aims to prevent.
Keep any cryptography code up to date

Many previously-used algorithms (DES, SHA-1), are now weak due to increases in computing power.

Even current algorithms will become ineffective over time and data encrypted now may be able to be vulnerable using more computing power in the future.

Be careful where you store your keys — if an attacker finds a copy, that attacker can reveal encrypted material or alter signed material.

Avoid placing keys alongside the data the key is protecting.

Be sure not to commit keys or passwords to public resources such as github.com. if you accidentally do, make sure you remove not just the most recent commit but anywhere in the history the key appears.

Be careful how you create your keys – keys of insufficient length can be broken.
The State of the Scientific Software World: Findings of the 2021 Trusted CI Software Assurance Annual Challenge Interviews

September 29, 2021

Status: Final Report v1
Distribution: Public

Andrew Adams, Kay Avila, Elisa Heymann, Mark Krenz, Jason R. Lee, Barton Miller, and Sean Peisert

Guide to Securing Scientific Software

December 14, 2021
Status: Final Report v1
Distribution: Public

Andrew Adams, Kay Avila, Elisa Heymann, Mark Krenz, Jason R. Lee, Barton P. Miller, and Sean Peisert

https://www.trustedci.org/software-assurance
Trusted CI’s 2022 Software Assurance Plans
2022 Software Assurance Plans

Secure Software Guide:
- Expand and refine the content.
- Work with the 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 of hands-on exercises and text chapters.
- Continue to teach in-person and virtual tutorials.

In-depth vulnerability assessment:
- Continue research into automating our methodology.

Threat models for ransomware:
- Developing comprehensive threat model for ransomware attacks.
https://www.trustedci.org/software-assurance

Trusted CI, the NSF Cybersecurity Center of Excellence is supported by the National Science Foundation under Grant #1920430. The views expressed do not necessarily reflect the views of the National Science Foundation or any other organization.
About the Trusted CI Webinar series

To view presentations, join the announcements mailing list, or submit requests to present, visit: trustedci.org/webinars or email webinars@trustedci.org

Next Webinar: Monday March 28th at 11am Eastern
Topic: TBD