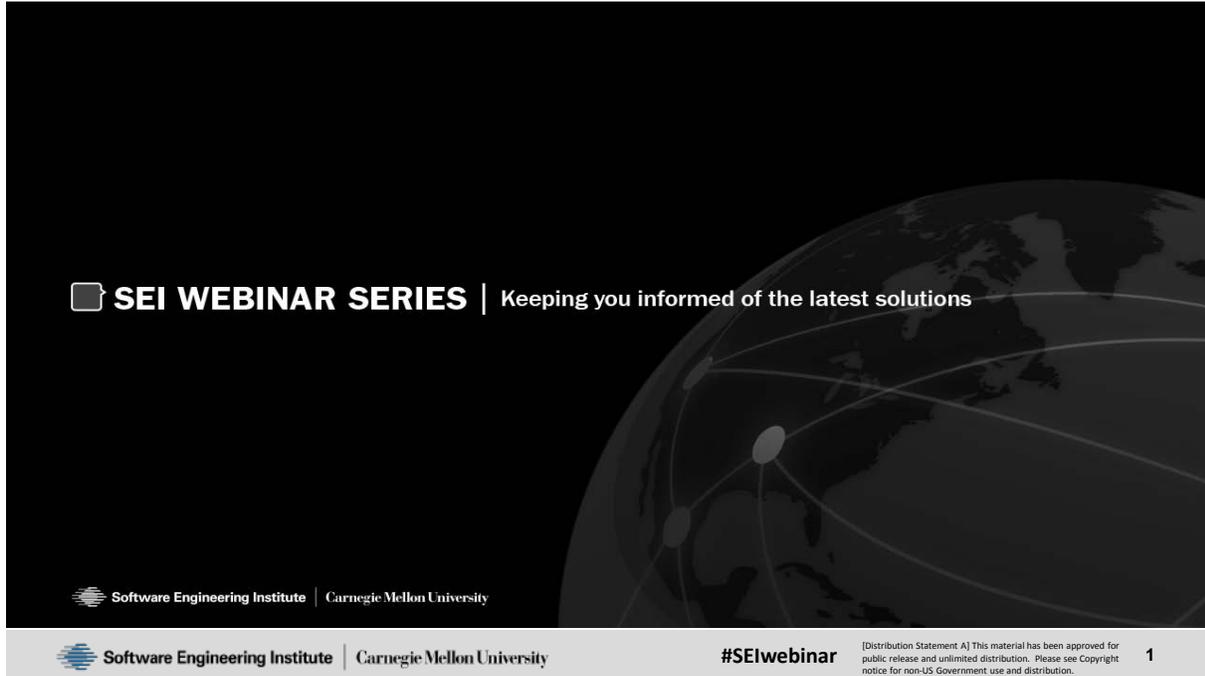


Secure Coding Best Practices Part 3

Table of Contents

SEI WEBINAR SERIES Keeping you informed of the latest solutions.....	2
Carnegie Mellon University.....	2
Copyright 2016 Carnegie Mellon University.....	3
Secure Coding Best Practices	3
Why Software Security?.....	5
Most Vulnerabilities Are Caused by Programming Errors	7
Secure Software Development	8
Sources of Software Insecurity	9
Unspecified and Undefined Behaviors	11
Polling Question	12
Adopting Secure Coding Practices	15
Tools encourage application of secure coding	20
Static Testing – Source code analysis tools.....	21
SCALE Multitool evaluation	23
Static Testing – Source code analysis tools.....	24
Conformance Testing	25
Polling Question	26
Supply Chain Software	29
SEI WEBINAR SERIES Keeping you informed of the latest solutions.....	31



Carnegie Mellon University

Carnegie Mellon University

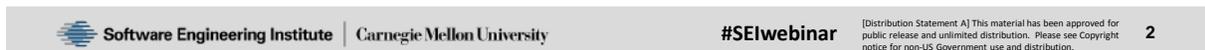
This video and all related information and materials (“materials”) are owned by Carnegie Mellon University. These materials are provided on an “as-is” “as available” basis without any warranties and solely for your personal viewing and use.

You agree that Carnegie Mellon is not liable with respect to any materials received by you as a result of viewing the video, or using referenced websites, and/or for any consequences or the use by you of such materials.

By viewing, downloading, and/or using this video and related materials, you agree that you have read and agree to our terms of use (www.sei.cmu.edu/legal/).

Distribution Statement A: Approved for Public Release; Distribution is Unlimited

© 2016 Carnegie Mellon University.



Copyright 2016 Carnegie Mellon University

Copyright 2016 Carnegie Mellon University

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Department of Defense.

References herein to any specific commercial product, process, or service by trade name, trade mark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by Carnegie Mellon University or its Software Engineering Institute.

NO WARRANTY. THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN "AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

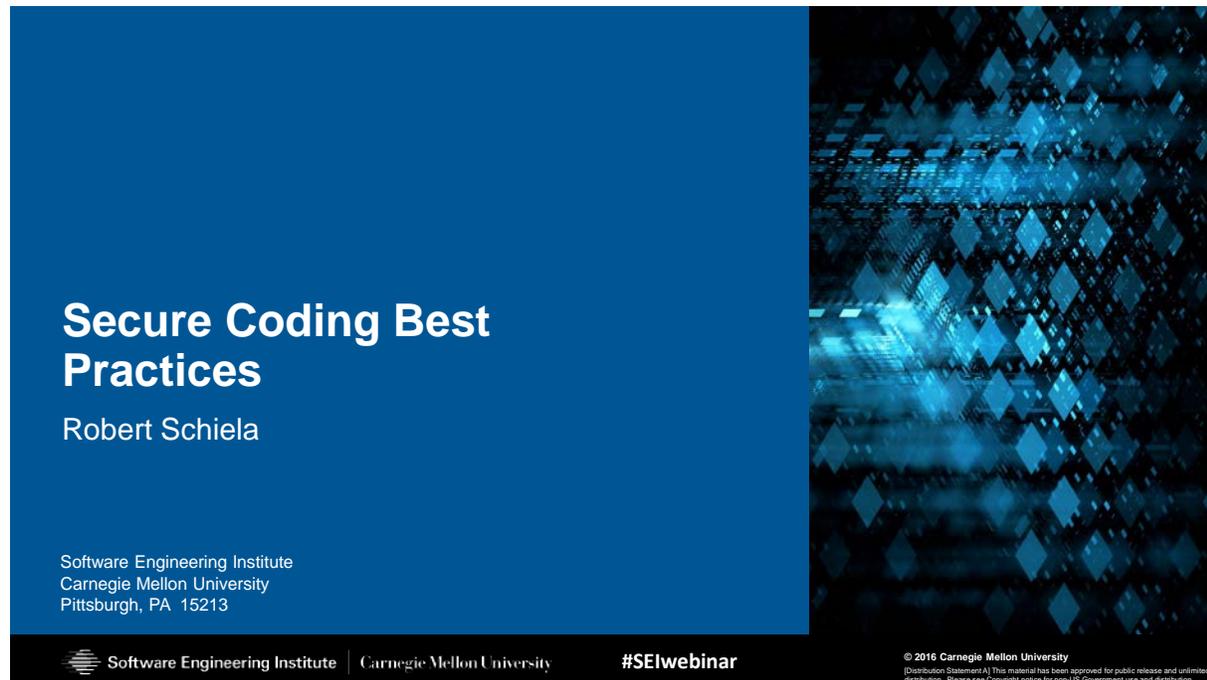
[Distribution Statement A] This material has been approved for public release and unlimited distribution. Please see Copyright notice for non-US Government use and distribution.

This material may be reproduced in its entirety, without modification, and freely distributed in written or electronic form without requesting formal permission. Permission is required for any other use. Requests for permission should be directed to the Software Engineering Institute at permission@sei.cmu.edu.

Carnegie Mellon® is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

DM-0003493

Secure Coding Best Practices



**065 And welcome back to the SEI virtual event, Lessons Learned from the Jeep Hack, How to Reduce Vulnerabilities in Cyber-physical

Systems. Once again, for anyone new joining us today, the presentation slides are available in the download materials tab at the bottom of the console. For those of you using Twitter be sure to follow @CERT_division and use the hashtag SEIwebinar. And there's a survey tab there as well as we want you to fill out that survey as your feedback is always greatly appreciated.

Our next talk is going to be secure coding best practices by Bob Schiela. And Bob's the technical manager leading the CERT secure coding team in the Cyber Securities Foundation Directorate within CERT. Bob has been working in the field of information technology, software development, and software development education for almost twenty years. He's been helping to lead research teams and projects in cybersecurity foundations for four years, primarily with the Science of Cyber Security Group. Prior to joining the Cyber Security Foundations team, Bob was a technical advisor of the director at the SEI. So, Bob you're doing to talk about what can be done during development to the security of software. So, what can be done?

Presenter: Thank you, Shane.

Presenter: Sure.

Presenter: Good afternoon, Mark and good afternoon to everyone that's here with us. Thank you for your time. At the risk of preaching to the choir, what I'd like to do is set a

little context. And I think this is a lot of what we've already heard today. But as it turns out, context is very important to secure software. And as it is with solving mazes, sometimes it's best to start at the end. So, what I'd like to start off with is thinking about what are we trying to accomplish and what are our goals. Largely, we're trying to avoid vulnerable software. We're trying to avoid hackers taking control of our software, hackers or unintended, or unauthorized access of data. We're trying to avoid physical damage in some cases. And--

Why Software Security?

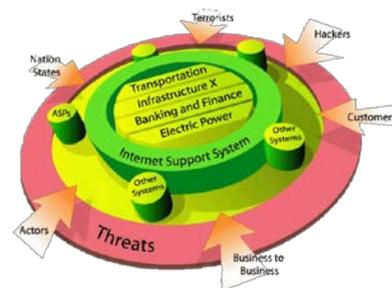
Why Software Security?

Developed nations' economies and defense depend, in large part, on the reliable execution of software

Software is ubiquitous, affecting all aspects of our personal and professional lives.

Software vulnerabilities are equally ubiquitous, jeopardizing:

- personal identities
- intellectual property
- consumer trust
- business services, operations, and continuity
- critical infrastructures & government



**066 So, why is this important? As I said, it's probably obvious to a lot of the audience here today. But software is everywhere. It's now more than ever part of our lives, part of our economy, part of the way we

live. And we rely, our businesses rely on software more than ever.

Additionally, software and the risk of attacks on our software and systems is ever present. And so, our systems are now more important than ever and yet, because of connectivity, they're able to be attacked. Or, at least the risk of attack is always constant.

Additionally, our software is vulnerable. The software that is in the field today, and unfortunately often software that we're developing today, has vulnerabilities that can be exploited, again, as unintended use. Systems that have been built are being put in environments that were never expected by the original designers and developers. Systems, cyber-physical systems for example, and controllers are just being bolted into or bolted on to networks that provide access that were never part of the original threat analysis if a threat analysis was done to the system, or a risk assessment, because there is never this-- or there was not an anticipated mechanism of getting onto the system through a network when it was originally developed.

Most Vulnerabilities Are Caused by Programming Errors

Most Vulnerabilities Are Caused by Programming Errors

64% of the vulnerabilities in the NIST National Vulnerability Database due to programming errors

- 51% of those were due to classic errors like buffer overflows, cross-site scripting, injection flaws

Top vulnerabilities include

- Integer overflow
- Buffer overflow
- Uncontrolled Format String
- Missing authentication
- Missing or incorrect authorization
- Reliance on untrusted inputs (aka tainted inputs)

Sources: Heffley/Meunier: Can Source Code Auditing Software Identify Common Vulnerabilities and Be Used to Evaluate Software Security?; cwe.mitre.org/top25
Jan 6, 2015

**067 Additionally, studies have shown that a substantial number of vulnerabilities come from programming errors. These are programming errors that have been well known for years, like integer overflow, buffer overflow, and missing authentication. Some of these vulnerabilities lead to system shut down. Some of them may lead to unauthorized access of data. And some of them unfortunately may lead to unanticipated control of your software end system.

Secure Software Development

Secure software development starts with understanding insecure coding practices, and how these may be exploited.

Insecure designs can lead to “intentional errors”, that is, the code is correctly implemented but the resulting software contains a vulnerability.

Secure designs require an understanding of functional and non-functional software requirements.

Secure coding requires an understanding of implementation specifics.

**068 And so, when we think about how to develop secure software, we have to think about and understand insecure coding practices that lead to exploitable vulnerabilities and understand them. We also have to think about the interactions of insecure designs and security built from insecure designs and how that leads to vulnerabilities. And as with secure designs needing the designers to understand both functional and nonfunctional requirements, we also have to think about, for secure coding, fully understanding the implementation specifics.

Presenter: So, Bob, what does secure coding mean? What does that really involve?

Sources of Software Insecurity

Sources of Software Insecurity

Absent or minimal consideration of security during all life cycle phases

Complexity, inadequacy, and change

Incorrect or changing assumptions

Not thinking like an attacker

Flawed specifications & designs

Poor implementation of software interfaces

Unintended, unexpected interactions

- with other components
- with the software's execution environment

Inadequate knowledge of secure coding practices



**069 Presenter: So, secure coding is a set of best practices to develop software while avoiding weaknesses that lead to vulnerabilities. So, here we're looking at the sources of insecurity. And there are many. Across the whole lifecycle, one common source is how much are you considering security through that lifecycle. And frankly, that's what much of the discussion today is about, the different phases of the lifecycle and when we can consider security, and how to. Another aspect is complexity of the software, complexity of the system, complexity of the design, and in some cases complexity of the source code itself.

There's also the issues of changes or changing assumptions. Again, as I mentioned before with regard to

what is the anticipated environment that a software is going to go into, and what's the anticipated uses. Not thinking like an attacker, I think Chris Valasek said this morning something about he and his partner have a particular way of thinking when they're working with these systems and trying to find issues that helped him and their research find issues to break into the Jeep.

Flawed specifications and designs, as we mentioned, poor implementation of software interfaces, that includes input/output to the users. But it also includes interfaces to other software components. And unintended and unexpected interactions, if there's one common statement that I'm-- common thread across my presentation, it's going to be unintended, unanticipated environments, the systems and interconnections, communication interfaces, uses and platforms or environments that the software has installed on. Finally, all of this can lead to what we would call inadequate knowledge of secure coding practices. And so, secure coding practices--

Unspecified and Undefined Behaviors

Unspecified and Undefined Behaviors

implementation-defined behavior - Unspecified behavior whereby each implementation documents how the choice is made.

unspecified behavior - Behavior for which the standard provides two or more possibilities and imposes no further requirements on which is chosen in any instance.

undefined behavior - Behavior, upon use of a nonportable or erroneous program construct or of erroneous data, for which the standard imposes no requirements. An example of undefined behavior is the behavior on integer overflow.

From ISO/IEC 9899-1999.



**070 Require the knowledge and understanding of the specifics of the language that you're using in the software, as well as the behavior and the implementation of the compilers and the hardware that's underlying it with regard to the anticipated platforms that you're installing. So, for example, the C language and the C++ language allow something called unspecified and undefined behaviors. Unspecified behaviors are when the language allows more than one activity or interpretation to be considered correct. And undefined is when the language does not define a particular interpretation because it deems whatever happened to not be correct. So, for example, an integer overflow is not something that should happen on the system. And so, the language deems that as not correct behavior. These exist to allow

flexibility in the implementations of the language, so the compilers and the hardware to try and optimize for what the systems are trying to optimize for.

However, that leads to the issue of portability or whether your code was written portable or not. Moving your software from one system to another might lead to completely unexpected behavior. And that is often a way that malicious attackers can take advantage of your software.

Polling Question

Polling Question

Does your organization use a coding standard for security?

- Yes
- No
- Maybe?



**071 Presenter: So, we're going to ask a few polling questions as we were with the earlier presentations. And the first one we're going to launch now is, "Does your organization use coding standards for security?" So, take about ten to fifteen seconds to vote. And Bob, you

can keep on presenting. And we'll circle back for the results.

Presenter: So, Bob you just mentioned some particular criteria used by C. What are the roles that language selection has in secure coding?

Presenter: Sure. So first, I would state that all languages have some semantic interpretation in their implementation. And all languages can be written securely or insecurely. There is no such thing as a secure or an insecure language. It's all about how you use the language. The devil's in the detail with the language, the compiler, and the interpreter. Though there's commonalities among secure coding best practices, the best practices and the rules themselves are language dependent. And in many cases, they're also platform and implementation dependent.

That said, some languages have a primary concern of performance. C would be one of the ones that I'd think of as it's prioritized for performance. Others have security and other qualities to make software writing easier. That's often a primary tradeoff, easier for the computer to run for performance or easier to write and maintain. Your system software may have very precise timing requirements, for example. And so, writing-- using a language that has a run time environment that is not precise may not be suitable for that particular use. But with the power to

manipulate the machine comes great responsibility not to mislead the machine or to allow someone or something else to mislead the machine.

Presenter: All right, to wrap up the polling question. I know we're going to go to another one here in a second. But the question was, "Does your organization use a coding standard for security," forty-six percent yes, thirty-four percent no, and twenty percent maybe or not sure.

Presenter: So, of that-- now, I don't know how many developers-- what the profile of our audience is. But I threw the maybe question in almost as a laugh. And I'm curious, if that's the people that answered maybe are just not the developer. They're either managers, or architects, or whatnot.

Presenter: Feel free to type that in the question and window if you're one of the ones that were maybe, and with what role you have, or why you answered that way. So, the next polling question will be up on your screen now. We've got in the language there a little bit. So, the question is, "What programming language do you work with most in your organization?" And that is-- there's quite a list there. So, take about fifteen or twenty seconds to vote. We'll turn it back to Mark and Bob.

Adopting Secure Coding Practices

Adopting Secure Coding Practices

Secure Coding Infrastructure

- Defining Secure Coding Practices
- Influencing Language Standards
- Influencing Tool Vendors

People

- Training

Processes

- Coding Standards and Security Standards, Testing

Technology

- Tools: IDE's and Analyzers
- Automated transformation and remediation



**073 Presenter: Though I think someone feels that they want to put in some more information in the box, I'd be curious to know which coding standard they use for security. There's a lot of coding standards for style, and a couple for safety. But since half of our audience says they're using them for security, that would be understand as well.

Presenter: Yes. So, one of the aspects with secure coding is trying to understand how can it help you. And that's what that question sort of leans to. And the ones that said either that they don't, or are not using secure coding standards, and/or that they may or may not be, it would also be interesting to understand why. But one of the issues is barriers of adopting secure coding standards. And so, we've been

doing several things to try to improve and reduce those barriers. In particular, a lot of work goes into what I call secure coding infrastructure. This is basically developing, and understanding of the languages, and the compilers, and the specifics that we already talked about. It's also codifying best practices and defining and developing the roles and recommendations for those languages and those platforms. There's a lot of work, as with much infrastructure, that needs to continue to go into that to keep things relevant and up to date.

Additionally, the language standard committees have been adding security aspects to the language. In particular, for C for example, we the SEI and CERT have helped with the C technical specification 17-9-61, which is a C secure coding standard into the language. Now, that standard is intended to be used by vendors and to try and help with regard to conformance. But it also ends up helping developers develop more secure software.

And finally, we and others are working with tool vendors. And the vendors are themselves trying to improve their tools to improve the ease, efficiency, and effectiveness of developers to apply secure coding practices, not just in tests, but also in earlier phases in the, for example, the IDEs.

Presenter: Just to wrap up the polling question quickly Bob, the

majority was fifty percent C, twenty-four percent C ++, and then Java at twenty-eight percent were the big returnees.

Presenter: Okay. So, I think that for how to help an organization adopt secure coding practices, the most important aspect is their people. Their people have more influence into the security of their software than any other aspect. And so, training people on secure coding practices and on correct tool use is very important. Many software developers have been self-taught. Some have been taught directly. But very few have been taught secure coding practices. They generally know how to functionally code but not to code securely. And so, helping and training your workforce to understand secure coding practices is very important. Also, helping them to understand the languages and the semantic meanings between the software and the hardware often is also helpful as well because it's a detail that often the developers don't know.

Presenter: Have you seen organizational barriers to adopting secure coding? These look like sort of some of the technical elements that have to be met in order to adopt it.

Presenter: So, certainly from a marketing perspective, it seems that there is a priority now for functional release, for speed to market with functionality. I think that as aspects, in particular as the risks keep

growing-- as you mentioned earlier, there are several now cyber-physical attacks that have caused physical damage, and the Sony hack, and OPM. As more and more attacks actually affect more and more people, I think adding aspects to security of the software is going to become more important as the companies realize the risk that they're under.

Right now, there is-- sorry, there has not been a lot of precedence for companies to pay a lot of risk compared to the amount of economic damage that has happened, for example Sony. There's been studies on the economic damage compared to what Sony paid its users. Now, leaders of companies sometimes have lost their jobs. But there hasn't been a lot of precedence for risk to actually be realized to companies. As that happens, I think security will become a higher quality attribute compared to the functionality and the speed to release.

So, additionally to improving the-- or reducing the barriers for the adoption of people is tools and automation. And this also I think goes to that question of why are organizations not currently worried about security as much as functionality. It's hard. It's not easy. The tools don't currently make it easy for developers or testers. And a lot of the tools right now have a low signal to noise ratio. And the tools need to be improved. That's something that I think tool vendors are currently working on.

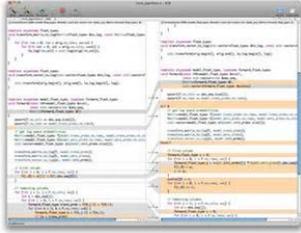
Additionally, organizations should still be adding these processes to their repertoire. So, they should be considering coding standards and coding reviews to try and improve the quality of the code because quality of code leads to less weaknesses, which leads to less vulnerabilities.

Presenter: What did you mean when you commented on you need to improve the signal to noise ratio?

Presenter: Ah, so a lot of tools-- actually, can we hold that question? And I will answer it in a moment. But that's a great question. And I already have an answer kind of queued up. What I wanted to do just to finish this off is to kind of lead into a further discussion of tools and just mention that using tools such as integrated development environments that provide alerts to users and paying attention to those alerts to try and improve the quality of your software is important as well as using static analyzers or dynamic analyzers to improve the security. We're also working on tools that try to automate changing of code, transforming of code, and remediation of errors with regard to error handling.

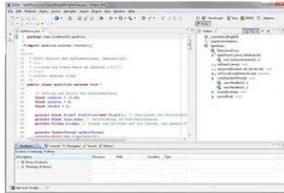
Tools encourage application of secure coding

Tools encourage application of secure coding



Moving rules into IDEs improves application of secure coding:

- Early feedback corrects errors on introduction.
- Exceptions are understood in context.



Adoption of secure coding IDEs

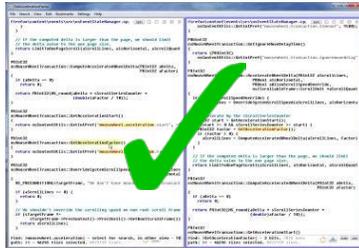
- help deploy tools
- training on tools
- extend tools to meet targeted needs

**074 But what I wanted to do was dig a little deeper with regard to tool use, which leads to that question of currently a lot of the alerts that tools provide to developers and testers are noisy. What I mean by noisy is they're false positives. They tell you something might be wrong when in fact there is nothing wrong. Or, whatever might be wrong is not going to lead to a vulnerability. And a lot of the tools simply aren't precise enough to give alerts that-- or at least a high proportion of alerts that actually are issues. There's certain identifiers in code that might suggest a particular issue. But in fact, there's also perfectly good reasons to do things in those ways if you're doing them the correct way. And often software analyzers can't tell the difference. So, they give you an abundance of alerts.

Now, one thing that we suggest is that static analyzers in particular are often particularly optimized for specific types of errors. And so, they find types of errors very well and try to reduce the number of false positives of certain errors. But at the same time, they might miss errors that they should catch. And so we recommend using multiple static analyzers to get better coverage of different types of errors. When doing that--

Static Testing – Source code analysis tools

Static Testing – Source code analysis tools



Secure Code Analysis Laboratory (SCALE)

- C, C++, Java, PERL, Python, Android rule conformance checking
- Thread safety analysis
- Information flows across Android applications
- Operating system call flows

**076 Presenter: Do you have any feeling for what a typical coverage is for a tool?

Presenter: I'm not sure I understand your question. What do you mean by that?

Presenter: So, I'll pick a number out of the air. Let's pretend that there's a hundred different kinds of problems that one might find.

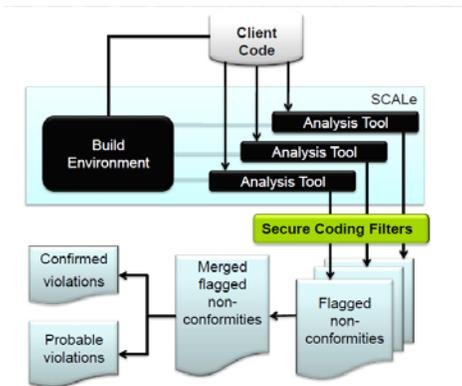
Presenter: Sure.

Presenter: Will the typical tool find five, ten, fifty of those kinds?

Presenter: Sure. Sure. So, my experience is that the tools-- there's a very wide range of tools. Many of the tools will find-- so, like you said, if we categorized to a hundred different types, many of the tools will find roughly twenty-five to fifty, the better tools. There aren't too many, if any tools that will find more than fifty percent of the types with one particular tool. And so, adding multiple tools might get you to a sixty or seventy percent coverage of the types of issues. Some of the issues simply cannot be found with tools. What I mean by that is there is no way for the computer to distinguish proper use versus improper use. But that's about the rough percentage I would say with regard to the tool use.

SCALe Multitool evaluation

SCALe Multitool evaluation



Improve expert review productivity by focusing on high priority violations

Filter select secure coding rule violations

- Eliminate irrelevant diagnostics
- Convert to common CERT Secure Coding rule labeling

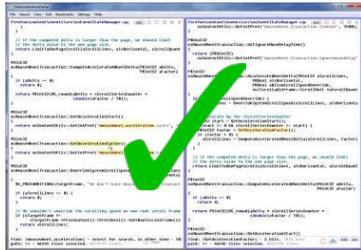
Single view into code and all diagnostics

Maintain record of decisions

**077 And so, one of the things that using multiple analyzers, a problem that that creates is, as I said, there's this false positive issue, this noise to signal ratio issue. Using multiple analyzers is just going to create more noise. And so, what we recommend, and we have a tool that can help with this, is aggregating the output of the tools, filtering and prioritizing the tools based on previous data and information that we know about the different tools and what rules, what issues they're better at finding, and paying attention to the ones that are most important as well as paying attention to the ones that have the highest risk with regard to what will happen if a weakness of this type gets through the software.

Static Testing – Source code analysis tools

Static Testing – Source code analysis tools



Secure Code Analysis Laboratory (SCALE)

- C, C++, Java, PERL, Python, Android rule conformance checking
- Thread safety analysis
- Information flows across Android applications
- Operating system call flows

**076 We also have a tool that helps with regard to our coverage of analyzing software using and aggregating all of the output of these static analyzers as I mentioned that helps with several languages including C, C++, Java, and Python.

Conformance Testing

Conformance Testing

The use of secure coding standards defines a proscriptive set of rules and recommendations to which the source code can be evaluated for compliance.

For each secure coding standard, the source code is certified as provably nonconforming, conforming, or provably conforming against each guideline in the standard:

Provably nonconforming	The code is provably nonconforming if one or more violations of a rule are discovered for which no deviation has been allowed.
Conforming	The code is conforming if no violations of a rule can be identified.
Provably conforming	Finally, the code is provably conforming if the code has been verified to adhere to the rule in all possible cases.

Evaluation violations of a particular rule ends when a “provably nonconforming” violation is discovered.

**075 The tool actually works with other languages. The issue is that we don't currently have rules for other languages. And as I mentioned with the software infrastructure issue is the care and feeding in development of new roles for new languages, or updating roles as languages change over time.

Another aspect or process that we recommend-- well, as we said earlier, it was having and using a set of secure coding rules, your own tailored version, but also having a security policy, and then using those rules for a conformance testing. So, at some point you want to test your software to make sure that it's actually complying with the security rules that you have defined, else you won't know if you're software's actually following, and you're

mitigating the risks like you think that you will be.

Polling Question

Polling Question

What testing does your organization perform on your software?

- Static Analysis
- Dynamic Analysis
- Both
- None

**078 Presenter: Okay, that's going to lead us to our next polling question, which is going to be posed now. And we'd like to know, "What testing does your organization perform on your software, static analysis, dynamic analysis, both, or none?" While we give you a little time to vote on that, seems like an ideal time for some audience questions. So, we'll turn to Bob and Mark for Brandon wants to know, "Are there methods to prove that software has secure code and lacks common vulnerabilities?"

Presenter: Prove is a hard word. So, back to what I was just mentioning about conformance testing, there's certainly tools that allow you to try

and find these issues. And if they find them and alert you to them, then you can go and mitigate them and remove them or change them. I don't know. I guess I'm afraid of the word prove. I don't know how you feel about that, Mark. But--

Presenter: So, I'm actually not afraid of the word prove. But the reason I'm not afraid is because it's not helpful. I could write a piece of code, "If Turing test, then secure coding violation," in which case, I promise you I can't write an analyzer that will be able to figure out one way or the other what the answer is. But as a practical matter, we've seen them getting better and with programmer assistance in various kinds of annotations, and we've seen this in certain classes of language features. For example, I think in the C++ threading area, these annotations are now being put into the standards and have shown some practical value. One can get better results as you move more and more into conventional theorem proving. Frankly, it requires more and more skill on the part of the programmer in order to specify what they're looking for even with the tool support. And it's become even more difficult and more training in practice is what we've seen. But it is getting better.

Presenter: And I think that, in practice, the big issue ends up being the complexity of the software. As your software becomes more complex, provability becomes really, really difficult and requires a lot of

resources that often become impractical.

Presenter: Okay, this is a question from Martha from one of our earlier polling questions. She's saying, "Your question did not have the right query. Some of our projects use coding standards, but only for normal coding practices. Others do use secure coding standards. And some do not use either. We are working on getting everyone to use them, not just there yet."

Presenter: Sure. So, that brings up an interesting aspect, which is understanding the criticality of your software and what you can and cannot test with the software. So, you may not be able to apply this same levels of security activities to all software that you're developing. And so that kind of leads us--

Supply Chain Software

Install Security Updates.

Test Source Code.

Review vendors' security and software assurance practices and results.

Request reports from their own testing or request independent security reviews and testing.

Test Binaries.

Apply Defense in Depth – only enable features that are required, and protect them.

Isolate critical and non-critical services and data.

Perform penetration testing.

Install an independent monitoring system.

**079 To probably our final slide here, which is what happens when you don't have the source code, or you're not developing the software yourself. And so, you're limited with regard to what you can do. As I said, you can't always apply the same practices. So, what I suggest with this is a layered security approach, basically doing whatever it is you can do. And that may depend on the leverage that you have on your lender as a buyer. And so, certainly as security updates become available, you should install the security updates on software packages that you are buying. If available, if you can get access to source code, in particular if it's critical software, you may want to go through the activity of testing it yourself.

Presenter: Okay, just to wrap up the polling question, which was, "What testing does your organization perform on your software," twenty-seven percent static analysis, one percent dynamic analysis, sixty percent both, and twelve percent none. So, we've got about thirty seconds, Bob. So, we're going to get one more question here before we go into our next talk.

Presenter: That's interesting as sort of the flip of what some of the other studies that we've seen.

Presenter: And this is from Ted asking, "It seems that the secure coding standards that we are aware of are more oriented toward business applications that our clients/server are highly networked. Are there coding standards tailored to embedded systems that are not widely networked or self-contained?" Do we have a quick answer for that one?

Presenter: I would say that a lot of popular languages, now there is a market for tools that help with the security of those languages. There may or may not be specific standardized rules for them. But tool vendors are trying to help develop software in lots of popular languages, even if they're web transactional software.

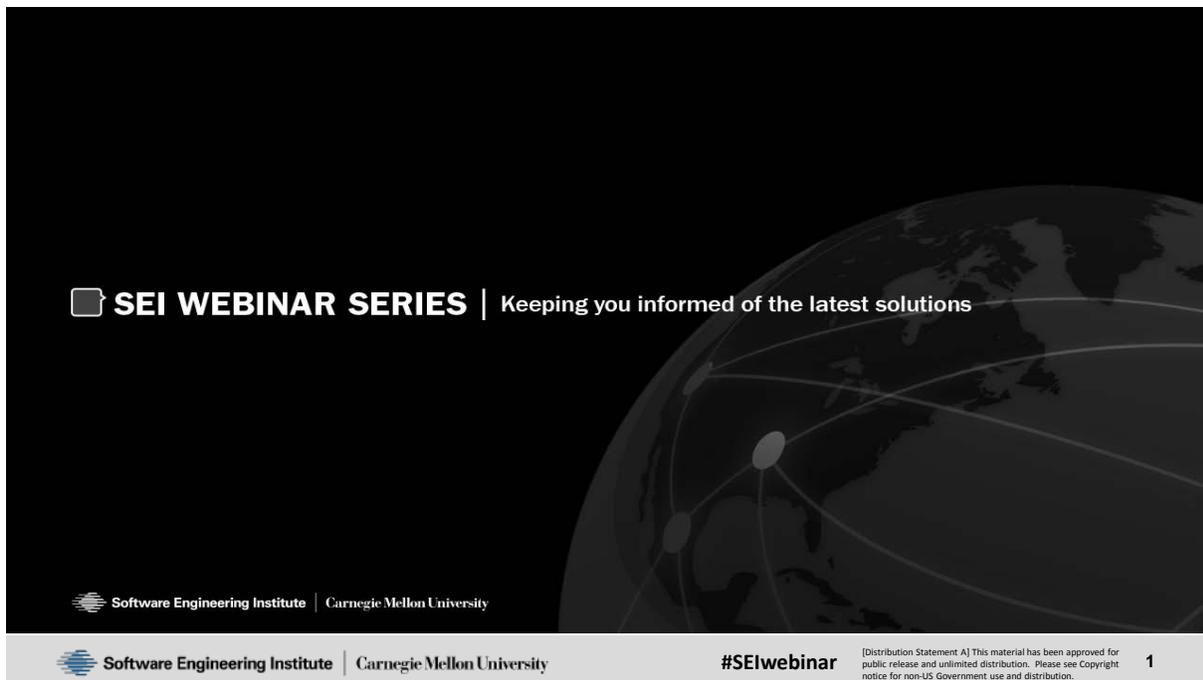
Presenter: But I think the summary statement is pretty accurate in that most of the attention has been given to conventional IT systems. And

there are a whole host of additional issues that are important to the embedded systems that I don't think a careful analysis of what's necessary for security in those systems has been carried out yet.

Presenter: Great. Bob, thank you for your talk.

Presenter: Thank you.

SEI WEBINAR SERIES | Keeping you informed of the latest solutions



SEI WEBINAR SERIES | Keeping you informed of the latest solutions

Software Engineering Institute | Carnegie Mellon University

Software Engineering Institute | Carnegie Mellon University

#SEIwebinar

[Distribution Statement A] This material has been approved for public release and unlimited distribution. Please see Copyright notice for non-US Government use and distribution.

1