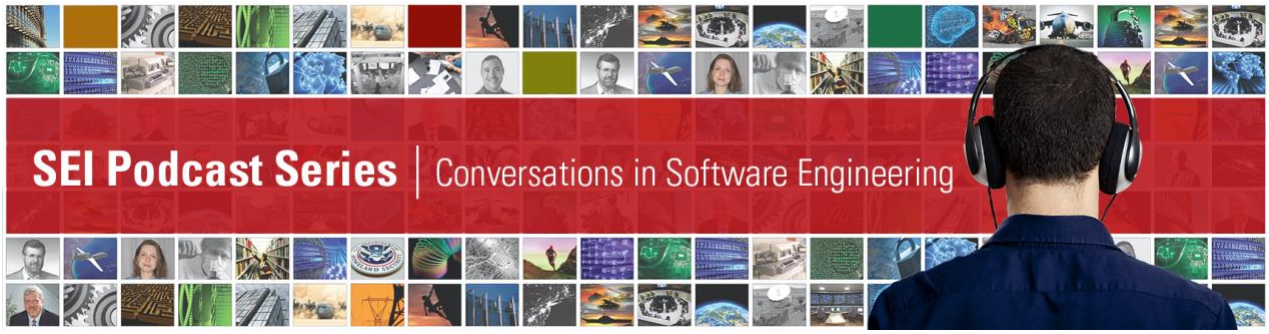

11 Rules for Ensuring a Security Model with AADL and Bell–LaPadula

Featuring Aaron Greenhouse as Interviewed by Suzanne Miller

Welcome to the SEI Podcast Series, a production of the Carnegie Mellon University Software Engineering Institute. The SEI is a federally funded research and development center sponsored by the U.S. Department of Defense. A transcript of today's podcast is posted on the SEI website at sei.cmu.edu/podcasts.

Suzanne Miller: Welcome to the SEI Podcast Series. My name is Suzanne Miller, and I am a principal researcher in the SEI Software Solutions Division. My guest for today's podcast, I'm pleased to bring in, is Aaron Greenhouse, a senior architect researcher also in our Software Solutions Division. Today, we are going to be talking about 11, not 10, not 12, but 11 rules for ensuring a security model, specifically by automating security with the Architecture Analysis and Design Language, AADL, and the Bell– I think it's La*Pad*ula, but it could also be LaPa*d*ula model. If anybody wants to give us a correction, let us know. Welcome, Aaron. I am so glad to see you again.

Aaron Greenhouse: Hi. Yes, it is good to be here.

Suzanne: I want to start off by having you tell us a little bit about yourself and the work that you do here. We have a lot of viewers that may be interested in the work we do, so we want to let them know a little bit about what is it like to work at the SEI and what kinds of stuff do you work on as a researcher here.

Aaron: As you said, my name is Aaron Greenhouse. I have a doctorate in computer science, Carnegie Mellon University. After I graduated, I came to work at SEI back in 2004. I was here for about three years, left in 2007, and I came back in 2017. So, that is an interesting career path, I think, at the SEI.

Suzanne: What made you come back to the SEI after being out in the industry for a while?

Aaron: Well, honestly, the not-so-started-up startup that I was in, it chugged along for 10 years and then came to an end. So, that is why I came back.

Suzanne: Very practical. What kind of work do you do here as a researcher?

Aaron: I have been working on the practical end of things trying to make the OSATE AADL development environment. I work on that. I worked on that 15 years ago, and I still work on it now.

Suzanne: You have evolved it quite a bit.

Aaron: Then I work on trying to see how we can make use of AADL in the real world, and that is where this work springs out of, which, actually, again, this work actually started back in 2006, I think. It was turned into a conference paper, I think, and part of an old version of it was presented in a conference in 2008. Then it sat on the shelf, but now is back because AADL…

Suzanne: The time has come.

Aaron: Well, AADL evolved since then, and security is always a hot issue, so I decided to revitalize it.

Suzanne: We have audience members who may not be familiar with AADL. We do have a lot of published work on that topic going back a long time, and we will include links to that work in the transcript. But the LaPadula model, can you give us a little overview of that and its relationship to AADL, and what was the catalyst to combine it with AADL?

Aaron: OK. I will try to give a brief summary of what that is all about. So the Bell–LaPadula or *LaPaDUla*…again, we don't know how to pronounce his last name, which is embarrassing. I have talked to many other people who also do not know how to pronounce the name, so that is where we are. It is a mathematical model that tries to formally describe what it means for a system to protect data and to only allow in provable ways that the things that are supposed to read the data are the only ones reading it, and it is being managed in all the appropriate ways. This is a model that came out of MITRE in the 1970s by David Bell and Len LaPadula. Really, it captures the two concepts that I think most people are familiar with: the idea of having a security level, and it is also combined with a need to know. So, there are security levels, which everybody is familiar with—Secret, Top Secret, Classified—although it can really be whatever you want it to be. Then these labels, additional labels, that get attached to it that can be used to further restrict based on categories or whatever who can see the information. We chose this model not to say that it is the best model, but it is a well-known model, and it was one of the first models. I think everybody pretty much understands what it is about. It was really meant to be more of a case study of how would you map this family of problems of security to AADL. Even though it

is a reasonably simple model, it turns out there are a lot of complicated issues that were involved that we hope are transferable in the future if a different model was being applied. It could assist in that development.

Suzanne: OK, so we are used to those two concepts, the security level and *need to know*, being applied to people. That is the thing that people are most familiar with is if you have a security level, and you as that person have a need to know, you can get access to something. Here, what we are talking about is actually getting into automation in terms of the software being the entity that needs to know and has a security level. So, that is the piece to me that is really fascinating is, how do you determine that a piece of software, an element of the software that is trying to access data, has that need to know.

Aaron: Right. So, essentially the model itself it just talks in terms of subjects and objects. So, subjects are things that consume information…

Suzanne: Data.

Aaron: …objects, or data, and they could be anything. Now, obviously it was developed in the context of computers specifically. But, it is easy to think of people in a library. In fact, I have a footnote where I give an example of using books and notebooks where the books are the objects. So, it is really quite agnostic. In fact, something can have both roles. Something can be both a subject and an object depending on how it is being used in the system. I actually wasn't going to say this, but since we went down this path, you can actually have pieces of the model as objects in the model itself. So when we talk about…

Suzanne: I'm sure people love recursion.

Aaron: Yes. So, when we talk about modifying access-control lists in a file system, they are part of the model because they influence which files can be accessed and how, but they also are controlled by the model because you need to control who can change the control list. So, that, I think, was an interesting…

Suzanne: So, what I am taking from this is that that assignment of need-to-know object and subject, security level are not as simple as they may seem on the surface. That is one of the reasons that we want a rich language like AADL to be involved in managing some of the accesses that are provided by the model because it is not just a simple *if this, then that*. We have got recursive issues. We have got subject-equals-object issues. We have got a lot of different things that we have to deal with. So, we need a rich language for dealing with that, which AADL is the richest language that I know of in being able to describe the context and the behavior of the software that we want to happen. Given that, you recently talked in a technical report and a blog post—thank you—in this area about security in particular and the 11 Rules for Ensuring a

Security Model. Let's talk about what those 11 rules are or at least a sampling of what those are, and what is it that those contribute to being able to leverage a model like the LaPadula model in modeling that engineers are doing.

Aaron: OK. So, before I discuss our rules, I do need to point out, within the Bell–LaPadula model, there are two basic properties that need to be enforced. These influence where our rules come from. One is called the *Simple Security Property*, and it corresponds to the problem of assuring that you have the right access level. So, it basically says, any object you touch needs to be within the realm of what you are allowed to touch. This is regulated in the model by having every subject [having] a security label. It is like a pair of a security level with a set of categories. So, it could be like, for example, Top Secret and F22. So, you have to be allowed to look at F22 information and to have the Top Secret clearance level, for example. So, these labels, pretty much everything has a label. Subjects have a label to say what they are allowed to look at. Objects have a label that says what is required to look at them. Again, so that is the Simple Security Property. And then, there is a second property that they call Star Property, and it is designed to control information leakage. So, just controlling that you can only touch things you are allowed to touch isn't sufficient because if you are, for example, touching a high-level object and a low-level object, you do not want to copy data from the high-level object into the low-level object.

Suzanne: You need to contain and set a boundary on what is the object that you're allowed to access.

Aaron: Right. So, the Star Property is a rule that basically says you can't do that. It says information can only flow to a higher level, which intuitively everybody nods their head and says, *Yes, of course. Now that you said that, that makes sense.* But in fact, that's often…in the real world, that is too restrictive. So, then there is this caveat of the *trusted subject*, and this originally applied…

Suzanne: And that is a whole can of worms of its own. We have another podcast on that.

Aaron: But why do we trust it? Well, that's a separate topic. But it was originally designed for things like the operating-system scheduler, which needs to be allowed to touch everything. But it has received significant enough scrutiny in its development—you hope—that it's not going to mess anything up. In our context, we're more interested in using the idea of being trusted for enabling operations such as data obfuscation. So if you want to take…you have a component that's going to scrub data, you're going to take a high-resolution image that may be very classified, if you lower the resolution, you can make it less classified. So that involves high-level data flowing towards a lower security level, and the Star Property would normally not allow that.

But in this case, this operation we call *trusted*. Or another trusted operation would be to encrypt data so you can actually put it over a lower level channel.

So those are some real-world operations that need to be escaped from the system. And so an interesting anecdote, why is the Star Property called the Star Property? I remember when I was reading this 15 years ago, I said that's a strange name, there must be a story to it, but it wasn't indicated anywhere in the document I was reading. But when I revisited in the last year to write the tech report, I found…David Bell had written a number of years ago a retrospective article on his model where he revealed the secret to the name, which was that when he presented it to LaPadula in a meeting, he didn't know what to call it, so he wrote Star Property as a placeholder [laughter]. And he said, *We need to come up with a better name or we're going to get stuck with this name*. And they never came up with a better name. And that's what the world is stuck with now.

Suzanne: Okay. There are little bits like that, like where did the term *bug* come from and things like that, that for the community that's very deeply into the software stuff are actually really fun things to know. So thank you for sharing that with us. All right so back to the 11 rules, we've got Simple Properties. We have got the Star Properties, and so I am imagining that the rules are about how you configure those properties and manipulate those to create your own security model.

Aaron: Right. So, AADL came up with this set of rules that basically have to do with…You can look at it from either two directions, either how you can check that a model is in good shape or you can look at it really as constraints on the design of the model. It's really two sides of the same coin.

Suzanne: And we use AADL in both ways…

Aaron: Exactly.

Suzanne: So we're going to have both branches available to people.

Aaron: You can think of it really as constraining the model, so that you can guarantee that the Simple Security Property and the Star Property make sense in the system design. I also thought it was interesting that we have 11 rules, and their first big application of this model was to Multics, so that really shows you how old the model is. When they applied it to Multics, they came up with a set of kernel primitives that had the same role. If you stuck to using these primitives, then everything would be in conformance with the model. And they also came up with 11. So I don't...there's something about…

Suzanne: Magic11.

Aaron: All right, so the first set of rules that we have, actually, the Simple Security Property turns out not to be so simple because it actually spawns six of our rules.

Suzanne: Oh wow.

Aaron: That is mainly because of the hierarchical nature of the modeling within AADL, which in general is a great benefit of how it allows you to design the system. But, it does make for some more complicated relationships that need to be checked. I will start off by saying the first thing we need to do is present the security labels within AADL. That is reasonably simple to do. You just make use of the AADL property mechanism. So, we have a pair of properties that can be used to express the security level and the set of categories. We just make sure that everything that we are interested in has these property associations applied to them. We also need to know in our world what are objects and what are subjects. Again, the subjects being the active participants within the model, those that manipulate the data. In AADL, it is pretty clear that that would be the components themselves. And the data is a little more challenging because the data is not really explicit so much. In AADL, you can have these data components, but that is just really a place where data lives. The data itself really doesn't exist at all, it just floats through the system. So, we have to use the features of the components as proxies for the data because that is where all the data comes in and out of the components. So, we are going to label components with AADL properties to make them subjects. And, we are going to label the features with the same properties, and those will be the data objects that need to be controlled. That said, so the first rule we get, which derives from the Simple Security Property, is that the security labels we annotate on the features must be within the bounds of the security label of the component whose features it is. So, the security label, the component has to be greater than the security label of each of its features.

Suzanne: Right. That is a basic property of security in a hierarchy.

Aaron: That is really the same basic principle we are now going to just keep enforcing in different places. Again, because AADL is so layered and enables a lot of different hierarchical containment, the next thing we look at is, well, how can features be hierarchical? Well, we have this thing called a *feature group*, which is really just a blob of features. So, we have to apply this all the way down. So, we now say, *Well, the security label we gave the feature group as a feature needs to be greater than the security labels of all its subfeatures*. That is not really a surprising result, but it's just kind of…

Suzanne: Not surprising, but necessary. A lot of these foundational rules are things that people know about, but when we move from a space into where we are automating, those simple rules are often the ones we forget. This basis that we all assume is happening, we can't assume that. We have to tell the models to check for that.

Aaron: Right. And now, in a similar sense, the components themselves in the system are hierarchically defined to have some components. So, we have a system. The system contains processes. The processes contain threads. Again, now, the original model, this Bell–LaPadula model, doesn't directly consider hierarchical composition in this way. It is not a big stretch to extend it if you just consider that a subcomponent, it's doing work on behalf of its container. It is easy to argue that whatever it does, its container needs to be allowed to do. It is not really a deep realization, but you need to take that step. Now the container, its security level has to be greater than the security level...at least as great as the security levels of all its subcomponents. Now, subcomponents can be more specific. They might have a smaller security level based on whatever it is they need to do, but the higher level aggregate needs to be allowed to do everything.

Suzanne: So, one of the things...the containerization aspect of current software architecture wasn't even conceived of in the way that we think of it in the '70s. So, it is actually very comforting in some ways that this model actually accommodates this evolution in how we think about things, even though it was designed and conceived prior to that mechanism being in place. That is actually very cool.

Aaron: Right. I agree. The model, the way it is defined, it is very agnostic and general about how it defines things, and it's very flexible, like I said, even allowing pieces of itself to be controlled by itself. So it's definitely refreshing that it just adapts.

The one part that is a little more interesting semantically within AADL is how we deal with the subprogram calls, which is the next rule that I have, because subprogram calls in AADL are a little bit weird in that you define a component to be the subprogram, but when you model a call within a thread, that is an instantiation almost of the subprogram. AADL semantics provide a variety of...Well, the standard provides a variety of semantics, I should say, of where that subprogram gets executed. Because AADL provides a couple of different ways of making it. Like you can be importing the subprogram from another process, you can have it within your own process. You can be getting it from some supplied library somewhere. Depending on exactly how you set up the linkage within the AADL model, the subprogram is actually going to be executed by a different device. It could sometimes be executed by the thread that is making the call, but it might also be executed in the address space of a different thread because you can be modeling a remote procedure call. It could be running off in a device driver somewhere. What is interesting is the relevance from the point of view of the security model because you have to make sure you apply the correct security...you have to identify the correct subject that is accessing the data.

Suzanne: And you want to constrain that. You don't want a library that you don't have security information known about to be allowed to be executed within a secure container. This is a very

simple example. So, the fact that AADL can handle that rule is actually very useful for making secure models.

Aaron: So, this needs to be a separate rule from our previous hierarchy rule because these subprograms...The point is the subprograms don't respect the containment hierarchy. Their semantics come from somewhere else. So the rule is simply that the subprogram call...the security label...the component that executes the subprogram must be greater than the data that it accesses. But the trick here is that you need to use the AADL semantics to determine what that component is. Of all the rules we have, that is the most semantically deep of all of them because of the complications involved in subprograms.

Suzanne: But, it is useful that we are addressing that kind of semantic complexity because the reality of the software that many of our customers build is that they are accessing data from here and from here and from here to create a composite picture of something. Those are all threat points in terms of, do I have the right pairing of level and need while I am composing and then of course as I do the composition. Being able to model that is one of the things that makes modeling useful, so that I can make choices. This is where the design comes into play, right? If I have these different sources, I may decide that this source over here, I am never going to be able to verify its security level. So, I can't use that source. I have to find another source for doing this. That is a very useful thing to find out before I actually do the coding. That is one of the things I love about AADL is that ability to tease out some of these really pretty esoteric problems, but ones that are very, very important when we get into security and safety. You can also apply, I am sure, the 11 rules for safety very similarly.

Suzanne: That's another blog [laughter].

Aaron: The next pair of rules also applies to more real-world problems where we deal with what we call the *AADL bindings*. If you are familiar with AADL, you know that there are really two categories of components. There are software components, such as processes and threads, and the same model will have hardware components, say memory and device and processor or bus. Then you are allowed to go between them by making in AADL what they call a *binding* where you say, *This process is running on this CPU, and this other process is running on another CPU, and when they exchange data, the data is actually going over this bus. But when it's communicating to this other subsystem, it's actually going to go over another bus*. So, you are allowed to see how the virtual exists on top of the physical. Again, this has security implications because you don't want to send your Top Secret data over a network that isn't rated for that kind of communication. You need to have this modeled in your system. So, again in AADL, we are going to have that you have applied security labels to the software components and to the hardware components, and so then we just have this set of rules that really make sure that the bindings respect the security rules as well. Again, they are going to be the obvious rules that

whatever you are bound to, it needs to have the permission to do whatever it is that you want to do. Again, the real exercise was in identifying that this is a potential problem within the model as it applies to AADL.

Suzanne: This is another area where AADL is different from some of the other modeling tools that we see that are very software-centric, even though AADL definitely has software-centricity as part of how it is built and how the language is set up. It also accommodates hardware as a real thing and does allow modeling of physical aspects, so that we can look at...To me, that is the real key to system integration is understanding your hardware and software together, embedded systems. This is *the* problem, the hardware and software integration. So, the fact that we can apply these kinds of security rules to the hardware part of the model, not just the software part of the model, gives it a lot of power in my mind.

Aaron: Agreed. Those are six rules. Again, the Simple Property turns out to have spawned six rules. The more intellectually complicated Star Property only gives us one rule in AADL, which, is kind of interesting.

Suzanne: But I'll bet it's a doozy.

Aaron: Well, it's a simple rule, but I would say figuring out how to deal with it, there was a lot of work in figuring it out. Once you did it, you realize, *Wow, that was kind of not surprising, I should have realized that originally*. I remember at the time I had many false starts as to how it should be represented before I had the epiphany, which is that—and I kind of already hinted at this in the way I spoke about the Star Property earlier. The basic point of the Star Property is that information should only flow from a lower security object to a higher-level object. I have chosen the word *flow* on purpose because one of the big features of AADL is that it allows for modeling at an abstract level of data flows.

My big realization was that, *Well, we just need to look at the flows*, which should seem obvious, but especially when you are new to AADL, it's not the way you are necessarily used to thinking. So, the rule is basically for each flow in a component, the security level of the destination of the flow needs to be higher than the highest source of the flow. And, that is it.

Suzanne: This is one of those rules that is simple, but not easy.

Aaron: Right. Actually, I say that, but this is one of the catches because in general the flows are not really a required component of an AADL model. The flow itself is almost kind of an application of the model because you are marking off paths of interest. It depends on what problem you are trying to solve, what are those paths of interest. So, you may only mark a subset of the flows that are possible within the model. In this case, for this application, we are really requiring that all flows will be specified and...

Suzanne: Again, being cognizant of the threat space that threat actors are going to take advantage of all the possible flows, not just the ones that are, I'll call it the *sunshine path*. That awareness is part of what makes a security model like this important.

Aaron: I am going to jump ahead. One of the future questions was what are some impediments to the deployment. This would be one of the stumbling blocks that I will identify because especially if you have a complex model, it would be pretty burdensome to identify all the flows, even though it is understandable why you would need them. Fortunately, this is the point where the analyzability of AADL and the presence of toolkits such as OSATE come in because you can— I am not going to talk about it now—but we outline in the tech report a process by which a tool could assist in identifying flows and lessening the burden on the modeler for this particular aspect.

Suzanne: And, that is actually part of the importance of these rules because they are rules that work together with your pipeline and with the automation that you are intending to use so that that automation supports understanding all of the security aspects of the model. It is like, *Well, that one is too hard for me as a human, so I am going to ignore it*. The automation can look at data differently and analyze things more robustly so that we can get a better picture of where the security gaps are.

Aaron: Right. And there's value...just in the exercise of finding the flows, there is value because you can imagine an automated process turning up flows you wouldn't have considered. And you are thinking, *Well, I did not intend for that to be the case at all. I don't want it to even be a potential for that to exist.*

Suzanne: Yes. OK. Are there some other drawbacks that people should be aware of?

Aaron: I think that was the main one that is really related to this model. I would say the other issues would be those in general related to modeling, which is that you need to make sure that the actual system that is constructed is true to the model, that there is a correspondence between the two. And that, unfortunately, is a whole larger set of issues.

Suzanne: That is another podcast.

Aaron: It is easy to become discouraged when you think about how to make that process work. But I encourage people not to be discouraged because as we have discussed, just the process of producing the model at least is going to make you aware of things you wouldn't have considered. By having more formalized steps… I don't want to belittle it by calling it a checklist, but more analogize it to *like* having a checklist. By going through the process, by at least attempting to model some aspects of the system, you are going to be asking yourself questions that are easy to overlook. Even if you can't guarantee 100 percent that you built identical to what you have

planned, you have revealed some information that you would not have otherwise, and it has informed your process down the line.

Suzanne: Or, you have learned things that will prevent you from adding in vulnerabilities that you are going to have to deal with later, and later is always more expensive, so it saves you in that. Even modeling that is challenging to do gives you that kind of learning. We are seeing a lot of that as people move into this space.

Aaron: I don't want the perfect to become the enemy of the good, in this case.

Suzanne: Right. There you go.

Aaron: A little bit is beneficial.

Suzanne: If I am an engineer that is actually using AADL, and I want to apply this approach to my security, what kinds of resources do we have available for them? Where should they start in figuring out how to apply these 11 rules to their AADL models?

Aaron: Well first, I would encourage them to take a look at the technical report. It has all of the details in it, and that can be found off the SEI webpage. And of course, we also would have to recommend that they get a copy of the OSATE toolkit, which they can download from OSATE, which is O-S-A-T-E dot org.

Suzanne: And, we will have that in our transcript. If you didn't catch it, we will give it to you.

Aaron: We are developing a plugin based on what is described in the technical report so that there will be analysis within OSATE to support this. Otherwise, they can always contact us by sending to info@sei.cmu.edu, and we will be happy to try to work with you.

Suzanne: OK. And for you yourself, what is next for you in this area of work? I hear you talk about the plugin. I imagine you are working on that. But, what other things are you working on related to this?

Aaron: Well, within the realm of security, there is a security annex being developed for the AADL language. This is mainly a set of standardized properties that can be used within AADL to define security levels and encryption and authentication schemes. Really, I would say that this work is one of the earliest consumers of this annex, that it relies on some of the properties defined within it. That is in the pipeline. It is not standardized yet, but it is in the works. I would imagine more work to come out of that line, or we look at how to apply the annex to some more real-world problems.

Suzanne: So, you will be busy for a while.

Aaron: Yes, I will.

Suzanne: When it comes to security, we're all going to be busy because every time we think we've got it nailed, some new threat comes in that we've got to adapt to. But I'm very encouraged that this Bell–LaPadula model seems to have the robustness and the flexibility to adapt to the new ways we are doing things and to some of the new ways that we have to characterize threats. So, that is actually very encouraging that we have a foundational model that is going to allow us to adapt to many different cases, not just a single one.

I want to thank you for joining me today. It was really nice to talk to you. It has been several years since we've been able to have this conversation, and I really appreciate that you came back to us and have been able to help progress this work. The AADL work is some of our more complicated work that we have done at the SEI. But in terms of safety and security, that modeling approach has got so many benefits to it, and I am really pleased to see that we are continuing to refine and evolve the utility of the security and safety aspects of this. So, I want to thank you again.

I want to remind our viewers that the things that we talked about today in terms of resources will be included as links in the transcript that we mentioned during the podcast. And I want to thank you all again for joining us today.

Thanks for joining us. This episode is available where you download podcasts, including SoundCloud, Stitcher, TuneIn Radio, Google Podcasts, and Apple Podcasts. It is also available on the SEI website at sei.cmu.edu/podcasts and the SEI's YouTube channel. This copyrighted work is made available through the Software Engineering Institute, a federally funded research and development center sponsored by the U.S. Department of Defense. For more information about the SEI and this work, please visit www.sei.cmu.edu. As always, if you have any questions, please do not hesitate to email us at info@sei.cmu.edu. Thank you.