FILE DETAILS

Audio Length: *79.50 minutes*

Audio Quality: ☐ *High* ☐ *Average* ☐ *Low*

Other Comments:

START OF TRANSCRIPT

FEMALE: You are listening to the Open Web Application Security Project, with your host Jim Manico.

JIM MANICO: You are listening to the Open Web Application Security Project and this is OWASP 93 and this Part One of our SECAPPDEV Developer Security Series.

THOMAS HERLEA: Hello. I'm Thomas Herlea and I am curating the podcast release of SECAPPDEV lecture recordings.

SECAPPDEV stands for Secure Application Development. Its mission is to advance secure software engineering practises in the development community.

Once a year, since 2005, it has been hosting - in Leuven, Belgium - lectures and security topics that go well beyond basics and with developers and with speakers who are specialists in their fields.

At SECAPPDEV.org you can find screencasts and hand-outs of these lectures.

We have with us today, Frank Piessens, a founder of this course and a professor at the Leuven University. He is active in research, consulting and teaching of software security. His interests lie in operating systems and middleware, architectures and applications, Java and .NET and software interfaces to security technologies.

The following lecture was recorded March 5th, 2012.

FRANK PIESSENS: The five second summary of my talk is Secure Application Development is too damn hard and here are some ideas on what

we can do about it. That's the five second summary. Of course that doesn't tell you a lot, so let me elaborate a bit.

Secure Application Development is hard. I don't think I have to convince you of that. You wouldn't be here for a full week intensive course if you didn't see the difficulties, at least. But it is difficult for a lot of different reasons and I don't know how to address all of them - so, no silver bullets here. I will zoom into specific causes of why this is such a hard problem and try to propose solutions there.

In particular, what I also won't do is short-term low hanging fruit solutions. They idea of this talk is, I want to point you to some - what I think - foundational causes of why we have so much trouble and some mid-term to long-term solutions.

So, what I will talk about are ideas and developments from the research communities of software security. So, don't expect something you can use tomorrow to solve your problems. So, this is a bit of expectation management.

I still hope that I can provide you with some insights that will help you understand some problems better. So, I do think there is a short-term benefit, but it's not the case that this is a - kind of a silver bullet that you can apply to your problems today. Okay?

So, okay, with that expectation correctly set, let me start on the real content.

So, first, focusing - in a sense - is… I told you Secure Application Development is hard and it's for many reasons.

So, one example reason is it's very hard to understand your security requirements upfront. It's very hard to know, upfront, what you want as security properties.

I was at the meeting in Germany last week and there was a professor from the Netherlands speaking about his experience doing consulting in security to industry. He was talking about - he was called to a company to help them and the first question he asked them, "what are your requirements, what kind of properties do you want for your software product?"

The answer he got was, “yeah, well, they shouldn’t be able to attack us”.

Okay. That’s true. So, the next question was then, “okay, what are attacks? What do you consider a successful attack? What are the attacks you want to stop?” And then they gave him three examples and these were, exactly, the attacks they had had over the last three months.

So, the security requirements were extremely reactive and I think this is common. It’s very hard to think about, proactively, about how are you - what are the kind of things that you should think about? It’s not something I know how to resolve. The best thing you can do is use state of the art practices there - and they will be addressed in this course.

I think the session that Johan Peeters will teach on Threat Modelling, for example, is one of the state of the art techniques on how to proactively think about “what should I protect” and “how should I protect it”.

But, I will not talk about that difficulty today. So, yeah. Because I don’t know how to fundamentally resolve it.

Similarly, another reason why Secure Application Development is hard to do is because the technologies that we work with are extremely complicated.

An example is Crypto - Cryptography. I’ve been teaching Crypto to master students for a while. I don’t do it anymore. I’ve been doing it for a few years and at many occasions I failed to explain even basic concepts like the difference between a digital signature and a certificate, for example.

To people at master level - if you talk to a cryptographer about these things he will say things like, “do you need the signature scheme that’s resistant against existential forgery with an adaptive chosen plaintext attack” or something like that. This is complicated. If you ask this question to developers, 99% of them will say, “I beg your pardon?” The other percent will say, “what the fuck?”

It's a difficult technology and, again, I don't know what to do about that in the short-term. Again, the best advice I can give you is, indeed, to attend a course like this one. We will have several talks by Bart Preneel, for example, on the intricacies of Crypto. He will give you advice on how to reuse existing cryptographic solutions and so forth. And that's the best you can do for the moment.

So, okay, with those disclaimers these are aspects of the difficulty of Secure Application Development that I will not touch.

What I will talk about is how the platform on which we develop impact the difficulty of Secure Application Development.

I will try to make a case that we make life, for developers, way too difficult in the way we design, compile, execute our programming languages and in the security guarantees that the platforms that we develop for, offer us, out of the box. That's the case I will make. I hope I can convince you.

I will start with that introduction, that there are a lot of issues there, and then I hope to give you some vision on how to resolve some of these issues towards the future. That's maybe the more sensible summary of what I want to talk about.

So, that leads me to the following structure of the talk. So, in the introduction I will motivate these disadvantages of the platforms that we work on, focusing on two platforms:

One is what I would call low level codes. So, C codes or some other unsafe language on say a Wintel platform, or something similar, Linux on Intel or even [unclear 0:07:11.1]. The details don't matter but that kind of development or imbedded systems, a development would fall under the same umbrella.

Secondly, the web platform, which is a completely different type of platform distributed, totally different kind of languages, totally different kind of middleware underneath it, and so forth.

So, I will try to explain on the basis of these two platforms that, basically, we make life too hard for developers there. I will end the introduction with identifying number of key challenges. Problems

that I think there are today and for which, I hope, we can offer some solutions.

Then the rest of the talk will depend a bit on timing, so these are two completely independent - so, I will conclude here with two challenges and then I will show you how to address them.

On the one hand, with a kind of a vision, we could try to do this. But this is a vision that's not realised yet. I mean, this is something that research communities are working on. I will also give some small steps in that direction. To show you that this is, maybe, a feasible vision. That this is something we might be able to reach in the long-term.

I can do that for two topics, but this will depend a bit on timing. These are independent. So, I can talk about only this. I can talk about both, depending on timing. The good thing about that is, also, at some point I will become fairly technical. Should I lose you - ask questions of course. But if I lose you anyway, you can begin again at the second topic. This will be a relatively independent topic.

But, an additional point on timing - so, this also means that I don't mind being interrupted and asked questions and even challenged. I mean, at some point it's a kind of a vision talk. So this is subjective. I think this is a good direction in which we could, but maybe you disagree and maybe you disagree for good reasons. I welcome discussion on that. So, don't hesitate to interrupt me for questions. If I say something that's not clear, you should interrupt me. Otherwise you don't get full value from the talk, of course.

And I get an indication here that I should repeat any question asked so that they are recorded. I'll try to think of that. Otherwise interrupt me again.

So, please do. And not only questions for understanding but also if you think, "he's saying crap. This doesn't make sense", challenge me. I prefer to spend some time on discussion with the audience. Maybe some people will agree with you and attack me. Maybe

others will agree with me and we can have an interesting discussion and that will be just as good as me finishing all topics of the talk.

Okay? So, for the - let's say the biggest part of the talk - we'll see how far we get depending on the time we have and the amount of interaction that there is.

Okay. So, let's start with the introduction. Look at some potential issues.

So, a key point is we expect too much of developers. Let me try to elaborate a bit on that. So, to understand, today, whether a particular piece of C code that you write is secure, requires you to understand so much that nobody in this room and nobody anywhere understands all of that. Nobody.

So, what you need to understand - so, here's a non-complete list - you need to understand the complexities of the C language, of course. And I think that's reasonable to expect. If you want to develop in C, you should understand the C language.

This is not trivial, so the spec - I don't remember which version of the NCC spec I took - but, a recent version of the NCC spec was around 700 pages of specifications.

I think this is what we can - this is a reasonable thing to ask. If you're a developer, you develop in C, you should understand C, period. But that's, by far, not enough to understand the security of your code. If you want to understand whether a vulnerability that is in the code is exploitable - yes or no - whether there are vulnerabilities in the code, this - typically - also requires you to understand the details of the complier. For example, if you go to the session of Yves Younan that Pieter Philippaerts writes - either today or tomorrow - they will explain to you all kinds of attacks you can do against C.

Well, none of these attacks can be explained at the level of C source code. They're all explained at the [unclear 0:11:25.2] if the compiler lays out the activation record for a function code like this.

And if you are lucky - and there is a library loaded at that address with these instructions in the middle of the method of that - there are many details that you need to understand of the platform on which you're executing, in order to understand the security of your code.

So, you need to understand the compiler - typically this is not a standard. So, the best thing to measure complexity is lines of code. These are the lines of codes of GCC, the standard C compiler on Linux, including the compiler and the libraries it links to.

To understand the security of your code, you need to understand Runtime Library Implementations. Again, maybe this is reasonable to ask that you, at least, understand the specifications of the libraries that you call. But, in practise, for C, what you have today is that in order to understand the security of your code, you don't need to only understand the library methods that you call, you need to understand the entire library.

There are attacks that will make your program jump to places in the library that you never even heard of, that are normally - should be dead code, basically. Again, in the session of Yves and Pieter, if you don't know about these attacks already you will hear about them. Things like jump [unclear 0:12:50.4], return [unclear 0:12:51.3], attacks and so forth.

You need to understand the operating system. It's perfectly possible that you have a program. You run it on one operating system or you run it on another operating system - here it's exploitable, here it's not. So, in order to understand all the details of the security of your code, you need to understand several details of your operating system. In particular, for example, which counter-measures have already been activated in that operating system against exploitations of C code.

And, to make things worse, there are even attacks that exploit bugs in the hardware. So, a processor is also a, kind of, a software system. There are also bugs in there. And, there are attacks that even exploit details of bugs in the hardware. So, you also need to -

and this is probably an incomplete list. So, I think - well, I do all this talking to convince you of the fact that this is not the reasonable thing to expect of a developer. Understanding all these details is just way too hard.

But let's look at another platform. Let's look at the Web platform. And, there we use different kinds of languages but we have different kind of issues.

So, the web… At a distance the web platform is simple. It's a client server thing with a thin client, the browser. A simple protocol, ACDP, and then some server implementing technology.

Well, unfortunately, none of these components is simple. It's, actually, not just client server. It's much more complicated. So, let's look at each of these components in turn and understand the complexities there. And, again, the main point of my talk here is that too understand the security of web applications, you also need to understand a lot of the details of each of these parts of the platform.

You have vulnerabilities that exploit all intricacies of all these three components and, again, this is just undoable. Nobody understands all the details. Nobody. So, that's a bit of the main point.

So, the browser - okay, again, do I have to convince you that the browser is a complex beast? So, what does it do? It's a UY, in a sense. It displays HDML. Well the HDML5 spec, alone, is several hundreds of pages of specifications. Okay? It executes several programming languages. Most notably, of course, JavaScript. This is standardised in the ECMAScript standard. The ECMAScript 5.1 spec is, again, several hundreds of pages. So, we are in one component and already at two times several hundreds of pages.

Other programming languages are supported through plugins. Most notably, I guess, Flash with ActionScript and the Java language which you also can support through a plugin. Flash, for example, has almost complete penetration. Almost any browser - over 90% of the browsers have Flash installed so you should consider this as

part of the browser. And Flash, alone, is as least as complex as JavaScript. The ActionScript language looks a bit like JavaScript. It's not exactly the same.

The browser supports much more than just http. I have a list of protocols here and this is an extendable list. So, even saying that it's only http is just a lie. Interesting with the new standardisation efforts going on, the set of APIs that all these programming languages can access is expanding rapidly. You can now do audio/ video, you can ask where a browser is located. You have a kind of a simulated file system, on the client's side. You can do messaging between the different websites or the different frames that are displayed in your browser. There are APIs offered to that - to scripts for that.

To understand the complexities there, there is, again, a session this week. The session, by Philippe De Ryck, on HTML5 security hopefully will give you an idea of the complexities that these new possibilities bring with them.

So, the conclusion here is that your operating system - sorry, your browser is becoming - a bit - your operating system these days.

Actually the browser is interacting with many stakeholders at the same time. In one type you have your banking website. Open another type, you have your gaming website. Opening a third type, you may have your Facebook open. You don't want these too interfere too much and that's what the browser is trying to do. So, it's trying to isolate content and data, code and data from different stakeholders making sure they don't interact in a bad way.

Well, this is done - I will come back again, if there's time, I will come back to that in detail how the browser does that, why it does it in an inadequate way and what kind of things we could improve there in the future. So, that's the browser.

The server is - again, I'm not even going to spend too much time on that. It's even many times more complicated. What you see here is an impression of what could be a fairly small website which

has separate machines for web logic, application logic and the data. So three tiers, the data tier as the last one. You see a list of possible technologies that could run there and you have a wide variety of them and all servers will use different things.

And, these are only small servers. A real big site - say, Google, Facebook, Amazon - you don't even begin to describe them with a picture like this. They are very intricate distributed system with [unclear 0:18:30.8], load balancing. Saying that Google, Facebook or Amazon is a web server is like saying that McDonalds is a restaurant or Starbucks is a coffee shop. It is magnitudes bigger than that. Okay?

So, I'm not going to talk about the server site a lot, in the rest of the talk. I'm not going to spend too much time on the intricacies here. But, I just want to convince you that they are there.

And, finally, the same goes for the protocol. So, http was supposed to be a simple protocol. It's stateless. But then, again, of course most web applications want some form of state so they implement it on top of http. There are many ways to do that. There are many ways to attack that. You have attacks like session hijacking, session fixation. You will hear about them in the course of the week. So, it's not fair to say that http is a stateless product. It's used as a stateful protocol in almost all application that matter, from a point of view of security.

The protocol methods *get, post* are supposed to be simple. So *get*, for example, is assumed to have no side effects on the server. In practice they're used - *get* is used to do arbitrarily complex things.

You may have heard the war story of the administrator of a website who had his website wiped out by the Google robot because the crawler found links to things that looked like *gets*, but they were implemented to manage the content on his site. So, you could remove - following a link on the management side - you could just remove all the content. Then the Google robot came along, saw all these links, happily followed them and wiped out the entire site. This is a side-effect of - the effect that nobody takes the idea

behind these http protocol methods seriously. Well, some people do but, definitely, not everybody does.

In addition, http is, in a sense, extensible because you can have http headers. You have a proliferation of them and each of them needs their own standard. So, you can easily find 20 standards describing the effect of http header fields. These all add to complexity of the protocol and these headers may be important for certain attacks. They actually are important for some attacks, so you need to understand them if you want to fully understand security of your application.

Another thing is redirects. So, http supports redirects and that breaks the simple client server picture. The server can redirect you to somewhere else for some processing and then that server can redirect you back - to come back - and this is done, in practise. So, you may be redirected for authentication to another site. You may be redirected for third-party payment to another site.

So, this basically turns a simple request into a really distributed computation. So, it's not just simple client server anymore. You need to think about that. And, again, many attacks exploit these intricacies. So they're only relevant in the presence of redirects. So, again, you need to understand the details here.

You rely - both the browser and http rely on DNS. So, I know many other infrastructural things - so, problems in DNS impact security.

Some of you may have heard about the DNS changer virus. This was in the Belgium news, I think, two weeks ago or something like that. What it basically does was change the way which DNS is resolved on your PC, so that instead of going to your banking site you go to some malicious site where they can steal your credentials and so forth.

So, by attacking DNS you can attack web applications. This is - everybody knows that, but I just want to point out the many points of attack that are there. And, as I already said, http is only one of the many protocols.

So, how do we deal with that? How do we deal with that today?

The way we deal with it is coding guidelines, tooling and ad hoc counter-measures. Let me go over each of them and they are important. I don't want to play them down so I will give critique here, but I think this is the best we have for the moment. But, my point later on will be, but this cannot be the long-term solution. We need to think about better ways to do this. That's, in a sense, a bit - the main point of my talk.

So, coding guidelines - there are many of them. You will hear about many of them in the course of this week and they are important. I just put as example the cert - the secure coding standard for C and for Java. For example, for C this is 89 rules and 133 recommendations. That's a lot to keep in mind and they're not always very precise either.

So, in addition we get tools that help us:

1. Check compliance with these coding rules.
2. That help us find other potential vulnerabilities.

But, both coding rules and the tools have false positives and false negatives. So, they don't cover all the holes. So, the idea of coding rules is we'll be careful and maybe we'll avoid all these pitfalls that are in the platform underneath us.

But sometimes you miss pitfalls, that's a false negative. So, you follow guidelines and you're still vulnerable. And, sometimes, you're just too careful and you're not - that's the other kind of false, it's a false positive. So, you may be being too careful and, hence, have disadvantages from that point of view. This restricts you more than actually should be necessary.

Okay, in addition - so, you will hear, again… So, for example, the kind of analysis tools that I talk about here, these are the ones that will appear in Ken's session where you get practical experience with several of these analysis tools and so forth.

In addition, we harden the platform with ad hoc counter-measures. Some examples here, you've heard about stack canaries, Jim mentioned them. You have these hardening compilers already. You will hear about these again in the session by Yves today. We have, in the operating system, address space layout randomisation where you shuffle around your codes to make sure that it's more difficult to exploit vulnerabilities in codes running on the operating system.

In web scripting languages we have taint mode to make sure that you can track potentially malicious input better and so forth. Again, you will hear about many of these counter-measures in talks today and they are important.

So, using coding guidelines tools, implementing these ad hoc counter-measures in the platform, they do substantially increase application security. So, don't misunderstand my talk and saying this is all crap. That's not the case. This is important and you should do it. You should follow these best practices as well as you can, for today.

But I don't think it can be the long-term solution. As platforms become more and more complex, coding guideline - there will be more and more coding guidelines and more and more heuristic analysis tools and they may have more complicated false positives and false negatives and stuff.

So, we should have some kind of more solid improvement in mind and this is what I want to propose. At least, ideas for that is what I want to propose and potentially discuss with you here.

So, I end this extensive introduction with the identification of two key challenges that I'll try to address in platform or program language/platform security.

So, the first one is the idea of high level programming languages was that they isolate you from the details of the platform that you execute on. Unfortunately, from the point of view of security, this fails miserably. It works fine from the point of view of functionality - up to some sense, I'll talk about it in a minute - but from the

point of view of security, I hope the examples I gave you show you that it fails miserably. So this is something we should work on and I'll propose a partial fix there in a minute.

Secondly, another thing we might expect from the platform is that it provides you some basic security guarantees that are useful to some of the stakeholders involved in the application.

Remember operating systems when they were built in the 60s/70s? They were supposed to expose the resources that they managed in a safe way and in a convenient way to the stakeholders using the operating system. At the point, typically, multiple users. Yeah?

Unfortunately, the operating systems that were developed then - with particular use cases in mind - are still being used today. For example, even on your mobile phones or your mobile devices there are often variants of Linux or Windows - or Unix I should say. Linux is just a variant of Unix. These are multi-user operating systems, but your phone is almost never a multi-user device. So, this is a complete mismatch and although people try to fix that…

So, for example, on an android phone a user will be created in the operating system for each application running on top of it. So, you try to re-use the inadequate mechanisms that you have there as good as you can, but they are inadequate. We need a re-thinking of the kind of security guarantees that a platform should provide to applications with the use cases for devices that we have today.

Again, I'm overstating here. I very much value how we slowly make progress in - I understand that revolution is not always possible, but - as I said in the beginning - the purpose of this talk is to try to give a bit of a long-term view.

Okay. So my goal is discuss some directions that we could follow to rectify this situation.

So, now I come to concrete suggestions for improvement. No silver bullets, nothing that will solve everything, but things that I think will address some real problems in a good way. And, so they will make our life a bit easier.

The first [unclear 0:28:30.5] is secure compilation to native code. So, how can we make sure - in a sense this is about how can we make sure that the program language is a good obstruction? How can you make sure that you - as a developer - can just reason at source code level? That's that this is about.

So, I'll discuss things like, what does it mean for a compiler to be secure? How should we define that? And, I'll explain what I mean with the principle of source based reasoning. And then, a key question is, of course, it's very easy to come up with, "oh, it should be like this", but then how do you achieve it? So, how can you make it like that?

So, I will - there - try to talk about some recent research going on that shows that, maybe, there is something to be had there. So, this is maybe an achievable vision. Although, again, I can't put enough disclaimers - I mean, this is hot off the needle research, in a sense.

Okay, what is secure compilation? So, again, the compiler is the tool that is supposed to isolate the programmer from the low level platform. That's the idea of high level language that I talked about before.

I think this succeeds well with respect to functionality. If you write Quicksort in and you compile it and you write it not doing any special tricks with the language, and you compile it on any C platform, it will work as Quicksort. It will sort things for you. So, with respect to functionality the high level language is a reasonable isolation, insulation from the details of the low level management.

But - as I discussed in the introduction - with respect to security properties, it fails terribly. So, what is missing? What are today's compilers missing? What would make a compiler or a compiler and a corresponding run-time system secure in the sense that you really get a nice isolation.

Well the answer is: it depends. There are several cases to consider, unfortunately.

So, as always in security, security is dependent on the power of an attacker. As you give more power to attackers, it's harder to become secure. This is a no-brainer in a sense. You can make mistakes in two directions, you can - assumptions about your attackers can be too weak and then you can be secure, in theory, but in practise you will be attacked like hell. I will claim that this is the situation we have with languages like Java or safe languages as we have them today.

Or you can give your attackers too much power. We assume the attacker can do almost everything and then it becomes impossible or very expensive to be secure and we shouldn't make that mistake either, of course.

So, I will look at two cases, two attacker models and discuss what it means for a compiler to be secure under these two attacker models. Okay?

For the first case this is a known - this is stable technology. We know how to do this - this has actually been done - but I think it's an example of the first kind of mistake that I talked about where we don't give enough power to the attacker in our treatment of security. We think an attacker can only do this. Whereas, in reality, he can do more. Okay.

Then I look at the second case which I think is a strengthening - it may be a mistake in the other direction. I don't know yet. So, maybe we give too much power to the attacker and maybe - in that way - it becomes too expensive to do security. But - well, I think it's interesting enough to share with you. I think it's another way of thinking about secure compilation than the one we have here. I hope I can convince you that there is merit in it.

Okay, let's first look at case one. Okay, so I'm first going to look at case - so, I'm going to go over them in detail now. So, there will be plenty of time. I've only a few slides on this because it's known stuff and I've many slides on this.

Okay, so Case One is: the attacker can only provide input to the program under attack. That's - so, all you can do as an attacker - there's a piece of software you write a module/an application and you have people who want to do something bad with it. We assume that all the attacker can do is send input to your program. This is sometimes realistic. For example, if you implement a network service that runs on a hardened and well protected server machine, then it may be the case - up to reasonable, it might be a reasonable assessment that all attackers can do is connect over the network to that service and then send data over a socket and they can send all kinds of malicious data, arbitrary data, data that the server doesn't expect - but that's all they can do. They can send bad input.

So, I will, in the following slide, explain that for this case a secure compiler - what he should do is just make sure that behaviour of programs is well defined - and I'll explain what I mean with that - for all possible inputs and that's what safe languages like Java, C#, Scala and so forth do.

Our second case is: the attacker can interact with the program in more intricate ways than just sending inputs. For example, the attacker may be able to load code in the same address space as the program that it's attacking. In that case, of course, you get a huge amount of extra power. You can scan memory, you can maybe even change the code that is there, unless there is some protective measure for that and so forth.

So, the - Case 2, I will call that: the attacker can interact with the program at the low level - and I'll be more specific of what I mean with low level. But, a very concrete example is - an example where this is a realistic attacker model is: software running on client machines.

There are studies that show that a significant fraction of internet connected end-user PCs are infected with kernel level malware. This is just a thing that is there, that we have to take into account, but that means that an attacker - at that point - can do much more

than just provide input to software running on the machine of the client.

This is one of the reasons why this is so hard, for example, to do secure web based banking these days. A lot of malware targets, for example, financial transactions that people are doing through their home PCs.

And I will explain in detail, for this case, that a secure compiler should have another notion of security - and I will explain, again, this doesn't make sense at this point yet - but that what we want here is that instead of just being well defined for all possible inputs, the compiler should preserve contextual equivalence.

You can't expect a compiler to solve all security issues for you. And, of course - you can't. So, there is a very specific division of responsibilities here. What I'm aiming for is that the programmer is responsible for any vulnerability that can be explained at source code level.

So, if you can explain an attack by going to the source code, then the developer is to blame. Sorry. You should understand this is already difficult. I mean, I don't solve everything. So, the division of responsibility that I'm after here is any attack that you can explain at the level of source code, the developer is to blame. Any attack that you have to explain, taking into account more platform details, the compiler is to blame. That's the division.

So, it won't solve some of the - so, if you need defensive input checking in the sense that… So, for example, think of SQL injection. If you are implementing in Java and you have a SQL injection vulnerability, this is something that you can explain at the level Java source code. So, I would say this is not a compiler's concern - sorry.

On the other hand, if you program your web application in a higher level language where you don't think about the database in SQL, but you think about persistent objects and things like that, then a

SQL injection attack cannot be explained anymore at the level of that high level source code and it's the compiler's fault.

So that's the division we should make here.

So, there are many kinds of vulnerabilities that are the responsibility of developers and I can't solve them at the compiler platform level. But there are, also, a lot of issues… So, for example, if you see a piece of JavaScript code that's allocating all kinds of objects and then does an API call to change the cursor, the effect that he then wipes out your machine is not - this is the kind of thing that a platform should protect against.

So, you will learn about these kinds of attacks in later modules, but in JavaScript what you can do is heap spraying, so you spray machine code all over the memory and then you call some vulnerable API methods which makes the processor jump to that machine code and all bets are off.

This is the kind of thing that the platform should protect against and, of course, this is a relative line in the sense that at what level of obstruction do we put our programming language? I would claim - for the web platform - Java is at a too low level. We need higher level languages - and I'll come back to it, although very briefly - we need what is called multi-tier languages. You write a single program and then the compiler decides - maybe based on an annotations - what to compile to JavaScript, what to compile to SQL, what to compile to Java at the server tier. And, then you can have a lot more expectations of the compiler solving some of the security issues for you. But, not everything.

There is always - I mean, business logic faults will always be the developer's problem. So, again, I did disclaim enough that I don't solve everything, but I think there's a very well delineated thing that we can solve it.

So, when all an attacker can do is give input to the program, then, basically, you can be secure or safe, or whatever, if you just make sure that the programming language always defines exactly what

should happen. We call a programming language safe if the language spec defines everything that should happen in all possible cases. This is less trivial than it seems.

If you look at the simple statement like the one here:

a[i] = (int) x.f()

So, this is written in some, say Java-like dialect. You call a method f on an object x, you cast a result to an int and you store it in an array.

The straightforward meaning of that statement is clear to everybody. But, there are many, many corner cases and the difficulty of language safety is nailing down all these corner cases.

What if x doesn't point to an object? What if it's null? What if x is pointing to an object, but it doesn't have a method f? What if it does have a method f, but the method takes more than zero parameters? What if it does have a method f with no parameters, but it doesn't return an integer with an object. Can you do that cast? What happens with that cast?

What if a is not pointing to an array? What if a is pointing to an array but it's not pointing to - i is not within the bounds of the array. What if a is in array and i is within bounds, but it doesn't contain integers? And, so forth.

So, the difficulty of program language safety is to make sure that for all these cases, it is well defined what should happen. Okay? And if you have that then, in a sense, you have a completely portable programming language because it's always defined - whatever corner case you come across, it's always defined what should happen.

And this is what, for example, languages like Java, C#, Scala and so forth do and this is what languages like C, C++, Pascal don't do.

So, the reason for unsafety is usually efficiency or overlooking things. So, for C, for example, unsafety - the consequence - is

done on purpose to make sure that the compiler doesn't have to do too many defensive checks.

In a language like Java, the language spec will say that a combination of a type checker and run-time checks emitted by the compiler has to make sure that for all possible corner cases, that you can arrive in, something well defined happens. And that can be expensive. And you will see that many of these languages, therefore, are indeed statically typed because you can rule out some of the defensive checking through type checking at compile time instead of doing it at run-time.

But you can do it for a language like JavaScript where performance is less of a concern, is also safe and you do all the checks at run-time. Okay?

So, for the unsafe language you - either for performance, you don't do that through oversight because you forgot some corner case. So, for example, Pascal is an old language, of course, was unsafe - not by design, but because of oversight.

But this is something that we know how to do now. So, new language features - so, think of a modern language like Scala - all features that are added in that language are studied very thoroughly from this point of view. So, for each feature added to Scala there will be proof that it is safe in this sense. So, this is known technology.

Of course, the language can be unsafe but the compiler can be more safe than the language. So, compiler can decide, for all undefined situations that there are - to do some kind of defensive check anyway.

So, the C standard may say, *if you access an area out of bounds what happens is undefined*, then a specific C compiler may decide to say, *okay, I will do defensive bounds checking and I will terminate the program immediately at that point*. So, we'll say a compiler is safe if it does that. If there are any remaining undefined behaviour, it leads to immediate terminations.

So, that means that you can't exploit undefinedness to do arbitrary things. Okay? And that's when you have a safe compiler, so compiler for safe languages are always safe because there is no undefinedness left.

Fully safe compilers for C-like languages, typically, are very expensive. So, they bear a huge performance cost. Okay? Although we get better and better there - I'll come back to that in a minute.

But - so, in principle, it's possible to implement a fully safe compiler for C, but for the moment making it fully safe is still quite expensive.

Okay, what are the benefits of a safe compiler? Well, the ones I talked about. So, a safe compiler is really fully portable. If you take a program and compile it on a safe compiler here, on a safe compiler there, you get the same behaviour on both sides.

You also cannot use programs to attack the obstruction introduced by the compiler. So, the compiler will use a run-time stack - on C you have things like stack smashing attacks - if you have a safe compiler you don't have that anymore, because as soon as you do something undefined, the program terminates.

So, if you would try to write beyond the bounds of an array with a safe compiler to access the stack, instead of memory that belongs to the program, this will be stopped.

So, the most important side-effect, from the point of view of security, is that any bugs you leave in your program, they don't become opportunities for an attacker to take over the machine. All an attacker can do is then terminate your program. So, instead of being able to take over a machine, all you can do is denial of service. You can still terminate a program by triggering a specific bug.

Whereas, in an unsafe compiler - well, it's really up to the developer to avoid undefined situations because otherwise anything could happen and that "anything could happen" is then very platform dependent.

Okay, I'm going to stop about Case 1 there. I think this is something you understand, this is exactly why Java is better for secure software than C. And it is the case, actually, because we understand this better, that C compilers also get closer and closer to being safe so the C compilers introduce all kinds of checks like stack canaries, like even bounds checking with 60% performance cost.

So, there is research going on, for 20 years, to improve compilers of C-like languages to become more and more safe. Again, Yves and Pieter will talk about that in later sessions. And we may have some hope that we get to reasonable safe sub-sets in the near future.

But, that will not solve everything. In many cases - that's the Case 2 that I want to talk about - in many cases attackers can do much more than this. So, for example, this is the worst case scenario because they infected the software system with malware. I already talked about that. Because the application supports plugins - suppose you have a browser and plugins in the browser - you can go over your browser code in detail and even formerly prove that you have no bugs in there. As soon as you load the plugin, you're toast.

The plugin is, again, loaded as binary code in the same address space as your main program so it can do whatever it wants. It's as bad, from the point of view of the browser, as when you would have kernel level malware on your machine.

Another thing, even if you're using Java or Scala or some safe language, they always (at some point) call native libraries - for example, to do IO and things like that - and as soon as you have a problem in there, then an attacker can do what is called a code injection attack. Which, again, let's an attacker run arbitrary other machine code in the memory space that the hosting process is running in.

So, even a Java program, if there is a vulnerability in one of the native libraries, can be attacked at a low level, as I would say. Not just by providing input. And there are many more examples.

As Jim already pointed out, all current compilers give up in that case. So, if you just look at the worst case - so let's keep that in mind that the operating system is infected - basically you give up.

But, as a consequence of that, this is why it's impossible to do secure web based banking. This is how attacks happen against users using their client PC to do valuable stuff. And banking is just one example. Okay.

So, an interesting question is, could we acknowledge the fact that current operating systems are too big to secure. That there's millions of lines of codes. So, can we accept the fact that they become malicious and still do something valuable with reasonable security guarantees. That's, in a sense, what we're aiming for. And the interesting thing is, it seems like we might be able to do it.

There are some recent breakthroughs in system security, in particular the PhD thesis of Bryan Parno, this is a PhD student from Carnegie Mellon Institute in the US, and the work of John McQuinn, also from Carnegie Mellon Institute. What they did - and I'll talk about it in a minute - is they developed security architectures that relies on the virtualisation support that you have in today's processors, that allow you to run small pieces of code - for example, only your application or only a small part of your application - on a PC today, in such a way that even if the operating system is infected, it cannot mess with your code. And that's exactly what we need to do more secure compilation and execution of code, safe against the Case 2 attackers that I talked about.

So, I'll explain - first I'll tell you a bit about the work that these people did. So, here is picture that I took from Bryan Parno's PhD thesis. He won, with that PhD thesis, the ACM Dissertation Award in 2010 which means - at least according to the jury - it was the best PhD thesis in Computer Science, worldwide in 2010. So, it is really an achievement.

So, what did he do? He built several security architectures with small variations, one of them is Flicker, another is TrustVisor. The essential idea of these - or at least one essential idea of this security architecture is that you can achieve what you see on the picture here.

So, suppose you have an application - think a browser - with some security critical piece of code - think some plugin that does web banking, so the thing that signs your transactions before the go off to the bank, for example. This is not how this is implemented today, but that's how you can think about it.

So, the question is, if you do that today, what other code do you need to trust before you can execute your small piece of code securely? And, Bryan Parno shows that, indeed, you need to trust all of this. As soon as there is a bug in any part that's coloured here - so it's the light grey - as soon as there's a bug in any of the light grey areas, an attacker can - in principle - exploit that to also attack your security critical code.

So, this is what's called a Trusted Computing Base in security, so the amount of code you need to trust before you can be sure of your security guarantees. Well that's huge. This operating system - he draws it this big on purpose because it is millions of lines of code, of course. Typically much, much bigger than the piece of code that you want to protect.

And, if you can take over the OS, if you can take over parts of the hardware - that is, typically, something that's less of a problem today, it may become a problem later - but hardware hacking is not something that we see as a realistic threat at this point in time.

Even if you can take over some of the other applications running as the same user, you often can, basically, also attack the security critical piece of code. You can, for example, steal the sign in key that is used to sign the transactions that are sent to the bank and then all bets are off.

So, what Bryan Parno did was he built a security architecture that, basically, runs two virtual machines. Virtual machines as you know them. So you know virtualisation - so what you get is two virtual machines that are strongly isolated from each other, with hardware measures. So, modern Intel and AMD processors have hardware support for doing this.

And, what he does is, he runs the entire old operating system and applications in one virtual machine and he runs just the security critical part in a second virtual machine and then his security architecture makes it possible to have some interaction between the hosting application and the security critical code.

But the key thing is - I'll explain in a bit more detail how this could work in practise by discussing our own implementation of this so. So, some of my PhD students also made an implementation of this and I'll discuss technical details in a minute - but, essentially, what you get is that you get the same kind of protection - if the OS is infected, if you have a server environment where you have several virtualised hosts next to each other and one is compromised, the others are still protected.

Well, that's the kind of protection you get here too. So, somebody can only - can completely take over this operating system, but he still cannot read or write arbitrarily to memory belonging to the second virtual machine.

So, we have here now - this is, say, three million lines of codes. We now have, here, something that is ten thousand lines of codes, that's the size of Bryan Parno's hypervisor. What you say is, okay you could attack the three million lines of code… Well we still can attack the ten thousand lines of code through - and it's very difficult to get these ten thousand lines of code secure and for the moment. So Parno's implementation and our implementation is not verified and it can probably be attacked.

But, I would say, going from three million lines of code with potential bugs to ten thousand is a good step and, in addition, there is hope that for this kind of size of code we can do very

strong security checks like the former methods that [unclear 0:52:14.4] mentioned here. For example, Microsoft proved some properties - no everything and there's still holes, I know. But, I mean, remember this is a long-term goal to go towards - but they did formerly prove some isolation properties of hyperv - their hypervisor.

In Australia, the NICTA people proved full functional correctness of a hypervisor, the L4 hypervisor, using formal methods. So, this becomes really… And there is the possibility of interaction between virtual machines, but to show you cannot do anything out of what is allowed by the specifications of the hypervisor and these are very strong assurances that we can never hope to have for these beasts.

There is no way that anybody will prove this of Windows, of Linux or whatever. This is actually indeed the long-term vision that people provide different modules to run on your device and they can communicate, but to be sure of the security of your module, you don't depend on the other modules anymore

You can rely on the fact that the way in which they can interact with you is limited to just providing input. So, we reduce to the Case 1. Even if they can load arbitrary code on the platform, that's a bit - so, you want to go to a situation where on a phone, on a client PC, different stakeholders can load modules on that - a bank, Facebook or whatever. But they don't - they shouldn't trust all the others in order to be sure that, for example, their secrets can be kept safe on the client's machine.

Actually two of my PhD students - I will mention their work - are working on that too and we have an implementation, we have a working implementation, but would I run my business on it? Not yet. These are research prototypes, I mean. But, I do believe there is hope. There is hope there in the mid-term, let's say. My disclaimers in the beginning are still valid. This is mid-term to long-term things.

Okay, so let me try to give you an idea. Let's see - how am I doing on time? So, let me try to give you an idea of how you could do

this kind of secure compilation and this is where I talk about the work we do, which was inspired by Bryan Parno's PhD thesis.

This is, basically, a substantial part of the PhD thesis of Raoul Strackx and of Pieter Agten - who is here by the way - so if you have questions about this you can contact him in the course of the week.

I will be a bit technical on this part, so don't hesitate to interrupt me if you have questions. But I still will not give you the full picture. I mean, I will have to over-simplify at some point. So, if you're interested or if you want to challenge it, if you want to attack it, either talk now, talk to me later, or talk to Pieter later. Whatever you prefer. Okay.

So, okay. I will show you how we could really build such a secure compiler today by describing in a very simplified setting. So, I will define a very simple source language - a bit Java-like, but much simpler than Java. I will describe what a low level platform that has some Flicker-like architecture on it looks like. Very simplified, but I will explain to you how it works. So, you could - I will explain to you, if you do this on an Intel processor, what it would look like. And then I will very briefly outline how you could compile Java to Intel plus Flicker and get this very strong security guarantees that I talked about.

I will also explain to you why you can think of that security property as providing contextual, preserving contextual equivalents as I mentioned before.

Okay, so here I will get more technical. Don't hesitate to interrupt. It's very difficult for me to assess how deep I should go in some things and how quickly I can go over others. So, don't hesitate to interact with me where you want to.

Okay, so let's first look at the language we will compile. So, this is the source language. You remember the ideas, so what we are after is, even in the case where attackers can take over your operating system, any kind of attack you can do against a module

that you implement in this language, you can explain at source code level.

So, any attack that you can do - even when you can load arbitrary machine code - you as the developer will be to blame because you could have seen it and the level of the source code.

There is no saying, "well, okay, yeah. The OS is infected with malware. Oh, game's over". No. There's a clear responsibilisation here. Yeah? But that's why it's also important to understand the source code. I will give some examples of security properties you might want to have and that might break in this source code. So, that's why I need to spend some time on it.

So, it's a very simple language. For simplicity we don't have dynamic memory allocation. So, we don't have classes anew. This is just for simplicity. We could extend it, but for the language that we have here, Pieter actually proved the security of compilation and in order to do that kind of proof you need to simplify.

But, fundamentally, we see no reason why we couldn't extend it even towards full Java. I mean, there will be many engineering challenges but no - hopefully - no fundamental challenges.

So, we don't have dynamic memory allocation. That's why we immediately declare objects. So, it's better to think of these objects as a kind of module that have private state. So, you have private fields. For simplicity all fields are private and all metas are public. Again, we could support other accessibility modifiers but for simplicity just have that.

So, think of an object as a module that manages some private state and that offers some operations on that private state. It's single threaded. This is something - how to deal with multi-threading is something we haven't worked out yet. That's more challenging than the dynamic memory allocation.

It has all the usual contraflow instructions. The one thing that we did want to have is ways of interacting intensely with other code and that's we why support indirect method calls, or delegates. If

you know C# or typed function pointers, as you have them in C#, in Scala and so forth.

So, fields - this is reflected in the typed system as follows. We have the unit type which is devoid type. So, it means you return nothing. We have the integer type, which is what you know and we have this MetaType - mType. What it says is, this is a pointer to a method with this signature. So, this is a method that takes to integer values and returns void or unit. Yeah?

So, this is - and then you - if you have such function pointers and you can pause them in as parameters or you can store them into fields and then you can call these functions using this syntax, as here. So, this calls the function that listener is pointing to with these two parameters. So, if you know delegates from C#, this is exactly the same thing.

Okay, so given that - I think with that you should be able to read what's here. So, what this is, is a very simple subject observer implementation. So, you have an object that manages a value, just an integer, you can get it and you can set it. If you set it and the value changes, a listener or an observer will be notified that something has changed.

So, you can register a listener, which is a function that a client of this module provides and that this module will call whenever the value changes. This is standard subject observer, I assume that most of you will know that. Okay? Is that code clear? Can anyone read this? If something is not clear here, speak up now because we'll look at some pieces of code in this source a few times on the following slides.

Thanks to the fact that we have private fields, you can encapsulate - you have encapsulation in this language. So, it may be impossible for a client of an object to distinguish two different implementations of an object, but that behave the same. This, again, if you are familiar with all programming, this should not be surprising. Here is a very simple example.

We have here an object that implements +2 in one go and we have one that implements it in two steps and, obviously, they behave the same, but technically what we say is - and this will be important later to express security properties - technically what we say is two objects are contextually equivalent - we write it like this - if no third test object can differentiate between them.

So, if it is impossible at source code level to write a client - another object, a test object that interacts with an object and it would return true if it's interacting with this one and it would return false if it's interacting with this one.

So, if there is a way - at source code level - to distinguish one object from another, then they are not contextually equivalent. If there is not a way to distinguish them, then they are contextually equivalent.

There are many ways in which you can distinguish implementations. You can do that based on behaviour, you can do that based on - do they ever terminate or not? You can do that based on how long does it take? You can do that based on how much memory does the code section use, and so forth.

So, to be precise - I can be precise about exactly what kind of measurements about objects that we can close and what not. Timing is not one of them. Timing level attacks still remain possible. What you get - how you can think of the protection that we get here is, that we can protect a module to the same level towards other machine codes in the same process, to the same level as we could protect a module and we could run it on a separate server and the attacker can only interact with it over the network.

Then you could still measure the time, how it long it took for the - and you can still do that here too. But it is still a significant step upward, but it doesn't close all channels.

There is another leak that we still have, that we think we can close, is - this is getting technical so maybe I should come back to it later

- but, it might still be the case that an implementation has a stack overflow on one side, another stack overflow on the other side. And since you don't have stack overflows at the high level language you can have them when you compile the machine code. This is also a way which you could still distinguish two objects.

But, I will show you the many things that you cannot do anymore and I think the gain is substantial. But it's not - I mean, there is never perfect security. Definitely.

This thing I've been saying about responsibilisation, what I'm basically am supporting here is what is called the principle of source based reasoning for security - this is not my idea, it's an idea from Andy Gordon from Microsoft Research, Cambridge - so the fact that you can find and understand any vulnerability in code by just looking at source codes.

So, now, why is contextual equivalents relevant here? Well, I want to show that thinking of attacks - so, we will want to be able to explain any attack as a malicious client module that's interacting with your module at source code level.

It's trying to get to your private key by calling your methods or trying registering malicious call-backs and so forth. That's all an attacker can do. For that kind of attackers, it's - a good way of thinking about security properties is thinking in terms of contextual equivalents and that's what I want to demonstrate now.

So, by - instead of thinking of an attacker, a test object is trying to break a security property. We can also think about an attacker who's trying to break contextual equivalents. He's trying to distinguish two objects and any kind of security property can be recast as trying to break contextual equivalents. And so, from that point on, we'll focus on just attackers trying to break contextual equivalents and it will simplify things.

And, I will show you that that's a good way of reasoning by just giving a few examples. So, here is a simple example. Again, unfortunately, the examples are small. I have to of course. Here is

a very small example where you have a field and you care about the integrity of that field. So, the context should not be able to modify that field. Yet, you will call the context. So, here is a place where you do a call back.

So, at some point in the execution of your module, you will give control to other code. Yeah? Of course, if you would compile this to machine code in Intel today, all bets are off. I mean, this is like a browser calling a plugin. Well, if the plugin can be malicious, it can start scanning your memory and write to any variable that you own and you are toast, basically.

So, this isn't problem that our compiler will have to solve. At source code level, I think it is true that you have integrity. If you write this module, whatever source code you could write to attack this - this is not the attacker source code - so whatever source code you could write to attack this, it will never succeed into changing the value of that private zero field, unless you provide methods that allow the context to do so and that would be mistake of you as a developer. Yeah?

But - so, if you have justice, there's nothing you can do. So, an attacker breaking integrity of this field is the same as an attacker distinguishing this object from the same object that does some defensive checking of the security property. So, if I can write an attacker that can change the integrity - that can attack the integrity of this field, then I can also write an attacker that can distinguish this object, from this object.

So, example, attacking something is the same as distinguishing contextual equivalence. Okay?

Second example - and I will just give two examples and then you'll have to believe me - but this is - there are - I could give many others and - well, any attack we care about we can phrase as a contextual equivalent. I'm just trying to convince you giving a few small examples here. But challenge me if you feel - if you see a hole in this.

So, here is a bit more complicated example. You may care about an invariant of your data. So, your balance should always be higher than zero. In this example it's - the maximum should always be higher than or equal than the minimum. Again, an attacker being able to break that property is the same as an attacker who can distinguish these two objects, where this object just adds defensive checking code for that invariant. Yeah?

So, again, any attack that I would have against this object and that breaks that invariant would also be an attack that distinguishes that object from this object. So, again, attacking a security property is the same as attacking contextual equivalents. Okay? So, the rationale is, any security property that you can express by writing more defensive checking about it, you can actually express as a contextual equivalence. Okay?

So, then - from now what we will aim for is a compiler that preserves contextual equivalence. Why is that good? So, we want to be able to reason about attacks at the high level only. Okay? What we care about, in real life, there are attacks where the attacker injects machine code into your process address space. Okay?

Now, we've seen that we can think of attacks as just breaking contextual equivalents. Now, if we have a compiler that preserves contextual equivalents, then we are okay. Why? Suppose we have an attack at a low level, suppose it's possible to attack at the low level by injecting machine codes, it's possible to have an attack.

Then, that low level attacker - which is a block of machine code - can distinguish the contextual equivalence of two modules at the low level. Yeah? That's what I just gave you evidence for. Okay?

But, since our compiler is contextual equivalence preserving, if you can distinguish the two low level modules you can distinguish the two corresponding high level modules. Yeah? Because if they would be equivalent here, and the compiler preserves it, then - well, you couldn't distinguish them at a low level either. Yeah?

So, the high level attacker - which is source code - can distinguish the two high level modules, hence, an attack exists at a high level, hence, the attack can be explained at source code.

So, if we have a compiler that preserves contextual equivalence we get our principle of source based reasoning and you can explain any attack at a high level and that's what we want.

So, let me - still briefly - summarise how that would work and for that I have to explain what a low level platform looks like and that's the last technical edition of this talk.

So, the low level platform that we compile to is an Intel platform, a standard Intel platform. Extend it with some access control module, which is basically the Flicker style thing - and I'll explain that in a minute.

The processor itself - well I assume you know this. This would actually work on other processors too. It's not Intel specific, but with our prototype it's on Intel. So we have a processor with a program counter, some general purpose registers, some stack pointers, a status register that has a sign flag, for example, and a zero flag. A standard address space, 32-bit addresses with 32-bit words. Again, the standard x86, 32-bit processor with instructions as you - at least if you've ever seen machine code, as you know them, instructions to load words into registers to store registers into main memory and so forth to do our method. These are the standard - this is a subset of the standard Intel instruction set and there would be no difficulty extending it to more instructions it that instruction set.

So, this is what a compiled, what compilation standard would look like. So, you have some source code and you compile it to blocks of machine code for each of the methods and you map each of the fields to specific words and memory and, obviously, the standard compilation if you would compile Java with GCC to machine code, it would not preserve contextual equivalence at all. For example, this secret value here - which you cannot read at source code level - if you have a block of malicious machine code it can just go to this

and this address and read it and write it or whatever it wants. So, I mean, this is why Jim said here, you're toast. As soon as the kernel has been taken over because you have all the power that you need.

So, we need some kind of low level protection mechanise and this is how it works. This is a variant of the Flicker architecture but it's the variant that was implemented by Raoul Strackx and it's the variant for which Pieter then proved that secure compilation is possible.

So, what you do is - at the hardware level or at the security architectural - you allow the platform to be configured with protected memory blocks. So, of memory space you can indicate certain areas as being protected. And for simplicity here I will only consider one protected memory block, but our implementation supports more than one.

And then in that protected memory you can have code and data sections. And this is something you already have in standard operating systems actually. The one thing you have extra is that you have meta data about the code section that is a list of entry points to the code section. This will be the places in which you are allowed to start executing the module that will be loaded in protected memory.

And then what the hardware access control or the security architecture access control does is very simple, if the - this is program counter dependent memory access control, if you are in unprotected memory - so if the PC, if the program counter is somewhere - this is off limits. You can't read or write code, you can't read or write data. The hardware or the security architecture enforces that.

All you can do is start executing at an entry point. So you can jump - so, basically, call one of the methods of the module. Then as soon as you're here - program counter dependent memory access control, the program counter is in protected memory - so, then you can read and execute code and read and write data. And this very

simple kind of low level access control is sufficient to do a secure compilation from a high level Java-like language. Okay.

So, an important point is, of course, that you can implement that kind of thing efficiently. Otherwise, yeah, what's the value of it? You can implement it in two ways:

- Flicker-style. So as a hypervisor, as a small hypervisor and this is what Raoul Strackx has already done.

  So, we have an implementation of that where you basically have again these two virtual machines. You have all the legacy code in one virtual machine and you have all the protected modules and some small kernel that implements the access control model in the other virtual machine.

- Or you can implement it in hardware and there was a paper at a security conference a few weeks ago that actually proposed the first implementation in hardware for imbedded systems of such access controls.

And both are feasible. So both - the performance costs you pay is relatively low.

Okay. Then I have to wrap up here. I have three minutes left, so I will just briefly say something about the compilation scheme. It's largely as expected, so you compile methods and put them in the code section. You allocate space for fields in the data section, you generate an entry point for each public method. But there are many tricky details and Pieter has investigated them all and proposed a solution for them. For example, where do you put the call stack? It's a tricky thing to resolve, that you have to split the call stack in two parts.

You need to be careful about how you handle returns and how you handle indirect calls to make sure the low level context cannot do - kind of jump to [unclear 1:14:15.8] or return [unclear 1:14:16.7] attacks against your module.

But you can address all these issues. I'm not going to go over them, for lack of time.

Pieter has implemented a compiler for the language that I've shown you and has proven that it covers - or that that it preserves contextual equivalents.

Raoul has implemented the run-time platform that you can compile to in an efficient way.

Okay. So, let me conclude. So, I hope - keep in mind my disclaimer from the beginning. This is mid-term to long-term stuff. But, I hope you see that this might be a way to eradicate a whole class of potential problems, potential security problems. Not all of them. I mean, several people have pointed out, "Oh, but this you can't address". No, that's still the developer's concern.

But, I hope - I think where we can go to is really allowing one to reason about security at the source code level. You don't need to understand the details below that.

And, I think - yeah, so, I had the exercise now but we'll skip it. So, to give you an idea of the intricacy of the kind of vulnerabilities that we still don't cover is: you can have source code, which has a vulnerability. If you can explain the attack at source code level, the vulnerability will remain. It's your problem. But all the other attacks are the compiler's problem.

So, this is a piece of code that manages a pin like on a smartcard and it allows you to test values of the pin and will give you access if the pin is right and it allows you to do this only three times. As soon as you've missed three times it will basically die and you can only - the test method for testing your pin won't do anything anymore.

If the pin is the correct one it will call you back with success. Which is represented with a zero and it will reset the counter of false tries. Otherwise it will call you back with a failure and it will increase the counter. And as you all see, of course, your social security, this has a bug and you can actually enumerate all the pins and brute force,

basically, the module. You can explain that attack at source code level.

Jim sees it undoubtedly and will write a code for you - I'm joking.

This is just to show how tricky it is. Here is how you could write source code and brute force the pin. The trick you do, basically, is you use the call-back that the module does to you, to notify you of failure - so, this one. You use that call-back to have another try and since the counter is only increased after the call-back returns, you can actually make this work.

And it's tricky because you have to make sure that you reset the counter after each attempt and you do that by - once you know the value by into - leaving a call to a successful test. So, the details are tricky. But - this is the disclaimer part - so these kinds of vulnerabilities will still be the developer's responsibility. That's something that the compiler can't fix.

But the fact that there would be a piece of machine code that can scan your memory and find the pin, that is now gone. Even in the presence of powerful attackers.

Okay, I think I should stop there. I just want to say - broadening this - so at the end I zoomed in a bit to a research prototype that we are doing now. I mean, we were working on this as we speak and this is course fairly narrow. But what I want you to remember is that broad idea behind separation of responsibilities between compiler and developer by using these secure compilation techniques and I think this is even more valuable in the web context.

We don't have results in Leuven about this but other people - for example, Andy Gordon from Cambridge - is doing this for web languages, where you write your web application in one single high level language. Then the compiler assures you that whatever low level attack you can do in a low level attack there, then it's injecting JavaScript somewhere or doing a reflection attack on a protocol or doing response splitting and that kind of stuff - that if

you have these low level attacks - the compiler will make sure that this is not possible. And we're far from there, but I think the vision is very valuable because it weeds out a whole range of problems and I think there is enough evidence that it might work, that we should continue working on it.

JIM MANICO: You've been listening to OWASP Podcast 93, an interview with Frank Piessens.

FEMALE: The Open Web Application Security Project is a 501(c)(3) not-for-profit worldwide charitable organisation focused on improving the security of application software.

Our mission is to make application security visible so that people and organisations can make informed decisions about true application security risks.

Everyone is free to participate in OWASP and all of our materials are available under a free and open software license. Please consider becoming an OWASP member today.

JIM MANICO: For more information please visit www.owasp.org.

END OF TRANSCRIPT