Approaching Security from an "Architecture First" Perspective

Table of Contents

Approaching Security from an " Architecture First" Perspective

**056 Shane McGraw: So we're going to get on to our second talk, which will be Security From an Architecture-First Perspective, by Rick Kazman, and Rick's going to talk from about two fifteen to three o'clock.

Rick Kazman is a professor at the University of Hawaii and a principal researcher at the SEI. His primary research interests are software architecture, design and analysis tools, software visualization, and software engineering economics. He also has interest in human-computer interaction, and information retrieval. Kazman has created several highly influential methods and tools for architecture analysis, including the Software Architecture Analysis method, the ATM, or the Architecture Trade-off Analysis Method, the Cost-Benefit Analysis Method, and the Dali

Architecture Reverse Engineering tool, and he's also the author of over 1150 peer-reviewed papers, and coauthored several books. Welcome, all the way from Hawaii. Rick, all yours. Thank you.

Rick Kazman: Thanks, Shane. So, in this talk, I'll be reviewing some of my recent research on architectural approaches to security, and this is something that I've become more and more interested in over the last couple of years, and I'll be picking up on several of the threads that Linda introduced. One is accelerating capability-- what can we do to not only include security in our systems, to architect for security, but to do so in an efficient way?

The whole notion of assurance. As Linda mentioned, assurance is not limited to security assurance, but clearly security is one of the major assurance concerns that most people have in today's networked world. And also evidence as a technical challenge. Frequently as software architects, we are searching for evidence for how to guide our decisions, and so hopefully I'll present a little bit of evidence that will stimulate some thinking towards how you might address security from an architectural perspective. And I should just say, before moving on, that this is joint work with Professor Jungwoo Ryoo at Penn State University, and Professor Humberto Cervantes at the UAM-- I'm not going to attempt to pronounce it-- in Mexico City.

An Architectural Approach

Software security is a complex multi-dimensional problem, touching coding, design, operation, and policy.

Most software engineering effort goes into secure *coding*.

**057 Security is complex, and it's a multidimensional problem. There are aspects of security that touch operations and training and policies and process and procedures. But from a software engineering perspective, we are primarily interested in coding and design, and unfortunately, from my perspective, most of the research effort and most of the effort in practice in security has thus far gone into secure coding.

An Architectural Approach - 2

But secure coding is not enough.
Why?

1. Security is a “weakest link” phenomenon.
2. Secure coding practices are expensive.

**058 And my contention is that secure coding is not enough. Yes, we need secure coding. We absolutely need secure coding. We cannot live without it. But it in itself is not the entirety of the solution. My contention is that secure coding alone will not get you where you need to go when you think about those multiple challenges, like assurance and accelerating capability.

And there's a couple reasons why this is the case. First of all, security is a weakest-link phenomenon, and what I mean by that is that if you think about other quality attributes, like modifiability or performance-- if you take on some technical debt in your system, you do some hacks, you duct tape some components together to quickly get some capability to the

market or to make a quick change to an existing system, perhaps you'll compromise the modifiability of the system a little bit in doing so. Perhaps you'll undermine some of the modular structure of the system. And in doing so, that system will become a little less modifiable, and you'll have incurred a little bit of technical debt.

Or, if you maybe made some changes to an algorithm or data structure or scaled the system a little bit, you might compromise the performance of the system a little bit. And yes, these are important. Technical debt is like rust. It never stops; it keeps on growing. But security is different. If you introduce a vulnerability into your system, your system isn't a little bit more insecure, your system is insecure. It's binary. And so we need to address security flaws much more urgently that modifiability or performance flaws. It's all or nothing.

Secondly, secure coding is not enough because secure coding is expensive. And again, it's not that we can do without it, but that should not be our only approach. As software engineers, as software architects, we have to be cognizant of cost as one of the drivers in system development.

An Architectural Approach - 3

We advocate an *architectural* approach to software security.

Specifically we advocate the use of *security frameworks*

- encapsulate best practices in design and coding

**059 So we advocate-- not surprisingly, given you're listening to this webinar-- an architectural approach to software security. Specifically what I mean by that is that we advocate the use of security frameworks, and I will talk a little bit more about security frameworks in a minute, but essentially a framework is any encapsulation of a set of functionality, a reasonable set of functionality, that you're going to use over and over throughout your system. This could be your own framework. It could be something that you build and you design, or that you build and use for a small set of systems within your company. It could be a commercial framework or an open source framework. But the point of a security framework is that you have a consistent platform upon

which you build your approach to security. You are not doing security simply based on coding the right stuff, but you are allocating the security concerns to a specific portion of your architecture.

So this is a proposal. This is an advocacy statement.

An Architectural Approach - 4.

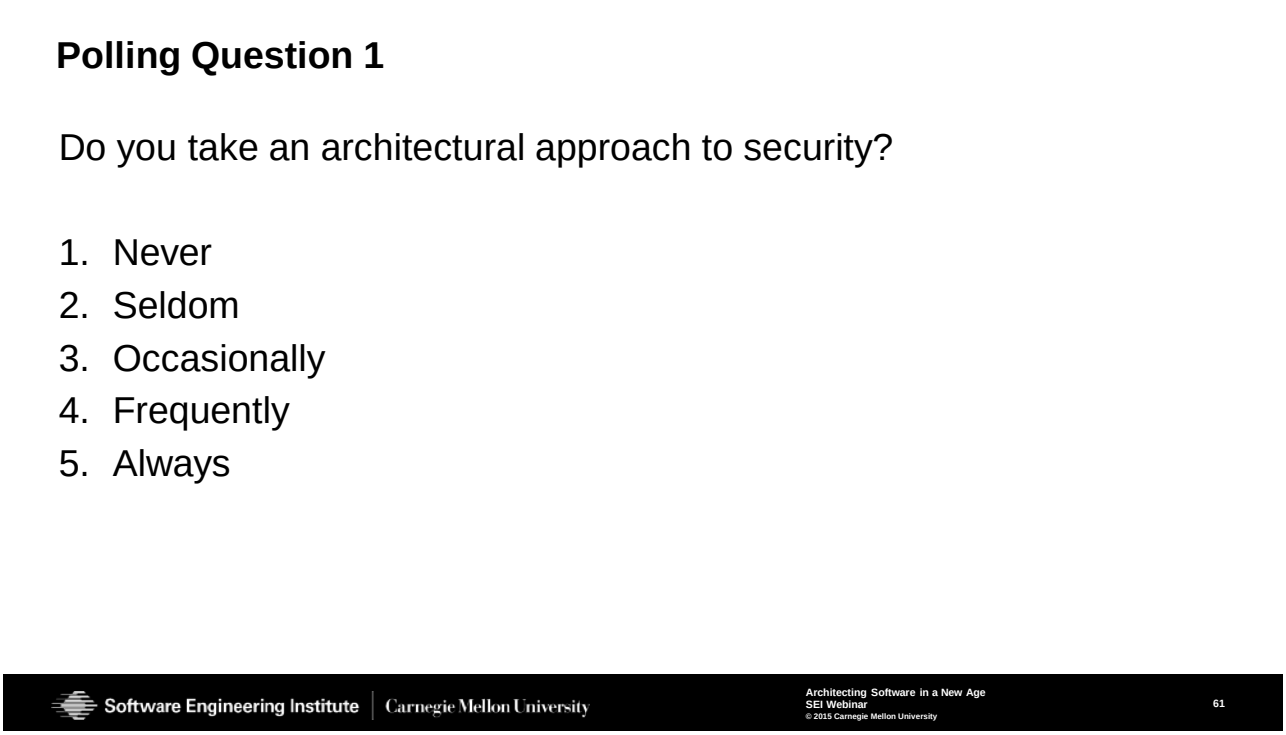
An Architectural Approach - 4.

What is the evidence for this advocacy?

Until now … nothing.

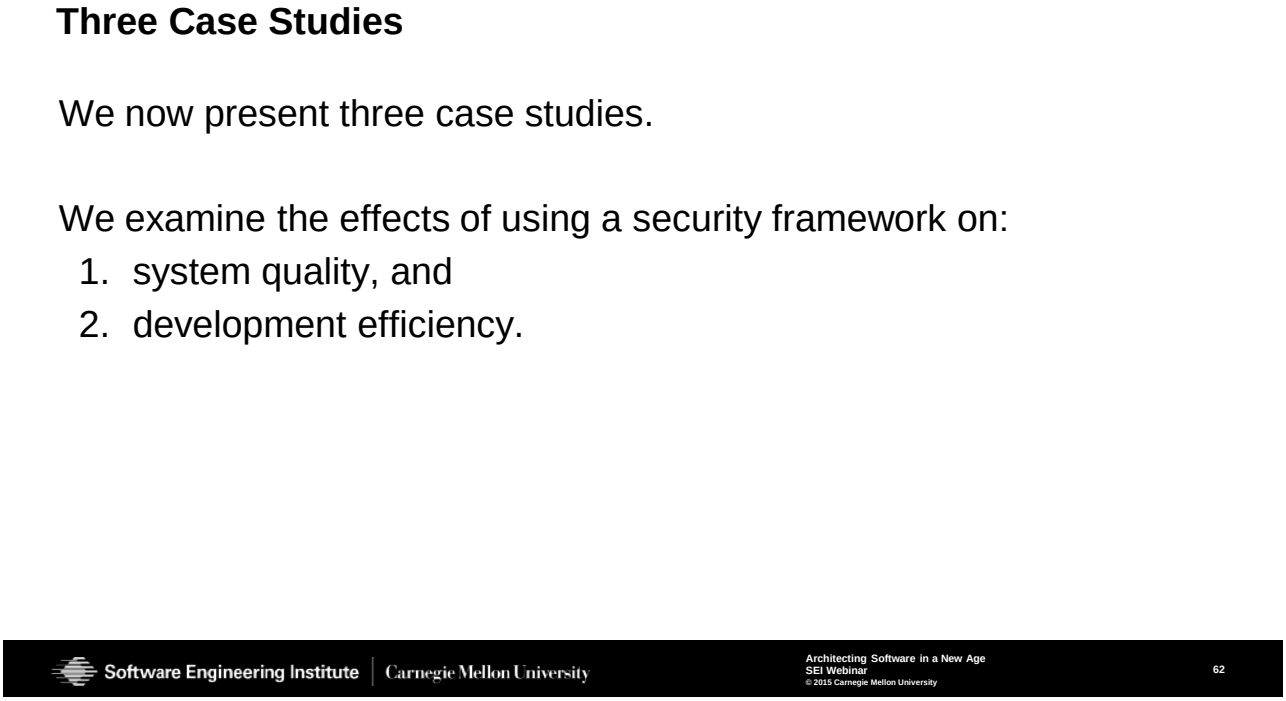
**060 What's the evidence for this advocacy? I would have to say, until now, nothing. This is an opinion, but I hope to present some evidence for this opinion.

Polling Question 1



**061 Shane McGraw: So we're going to launch the first polling question, one of two that will take place during Rick's presentation, and that is: Do you take an architectural approach to security? So take about 110 or 15 seconds to vote, and Rick, you can move on and we'll log in the results in about a minute or so.

Three Case Studies

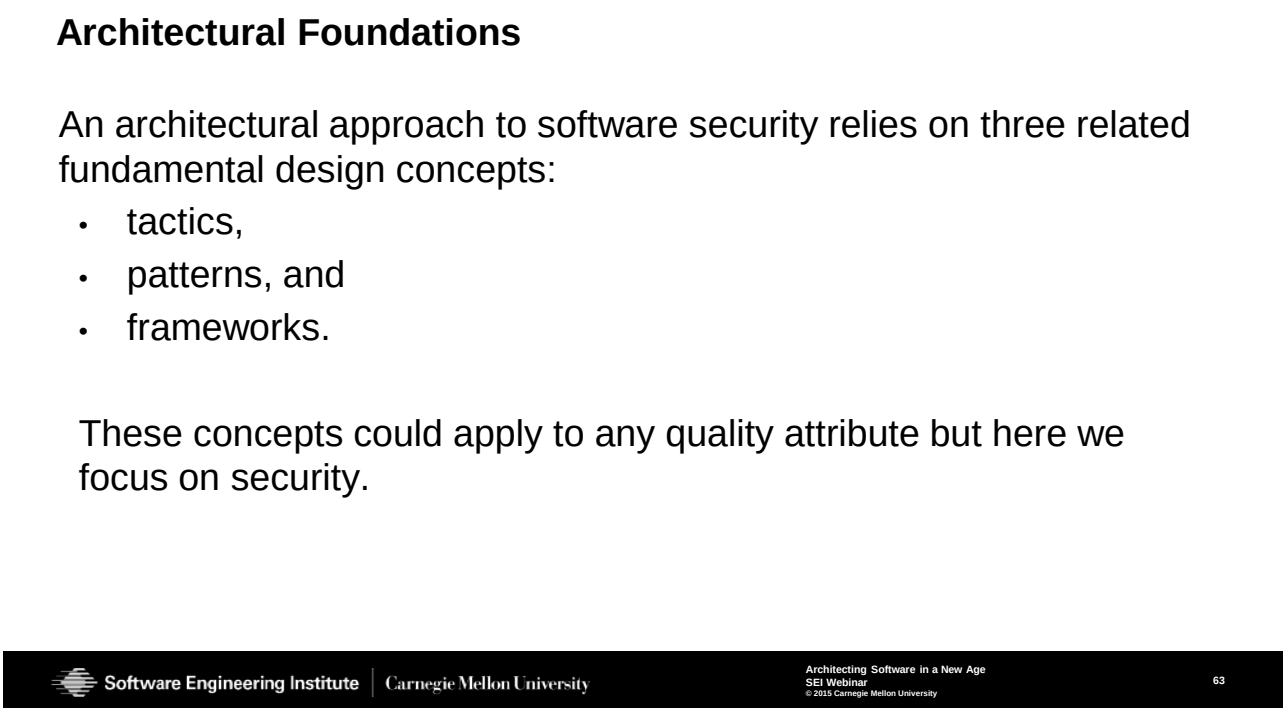


**062 Rick Kazman: Great, thank you.

So, to address this question, to provide some evidence for the advocacy, I'm now going to present three case studies. So, empirical evidence is hard to come by in software engineering, in part because it's very difficult to isolate the many, many factors that can vary from project to project and domain to domain. So a case study approach is one that allows us to probe the details of a software project and focus on what we think are the salient details, and try to understand the relationship between the decisions that have been made and the consequences, or the outcomes.

So in this study, we're going to look at the effects of using a security framework on both system quality-- specifically on security-- but also on development efficiency, on how much effort we spend on that security, on achieving a particular level of security.

Architectural Foundations



**063 So let me take a step backward before we delve into the details of the study and talk about what it means to take an architectural approach to security, and there are three foundational design concepts that I will briefly introduce, and if you're familiar with the SEI's body of work on software architecture, none of these will be a surprise to you. These are tactics, patterns, and frameworks.

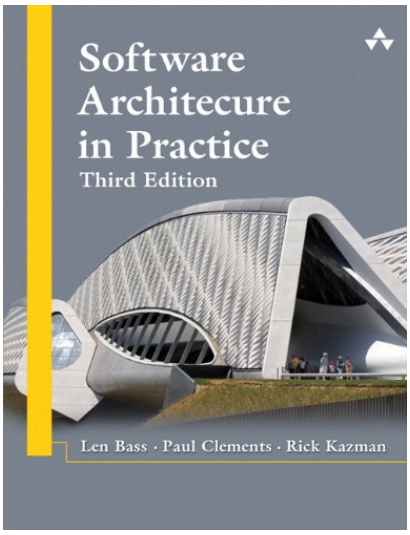
These concepts apply to any quality attribute-- in the book "Software Architecture and Practice" we have a chapter on each of seven quality attributes-- performance, availability, security, modifiability, interoperability, and so forth-- but here, of course, we're going to focus on these three concepts as they apply to security.

Tactics

Tactics

Architectural tactics are techniques that an architect can employ to achieve required quality attributes in a system.

The tactics used here are taken from:



**064 So, for those of you that are not familiar with this concept, an architectural tactic is a design primitive. It is a fundamental design choice that an architect makes and a design technique that an architect employs to achieve some quality attribute in a system. So, lest you think these only apply to security, let me give a quick example from availability.

If I want to architect a system for high availability, I'm going to have to make some design choices. I'm going to have to decide, "How am I going to detect a fault? Once I've detected a fault, how am I going to react to that fault? How am I going to recover from that fault?" And perhaps I may also have some desire to avoid or mask faults entirely.

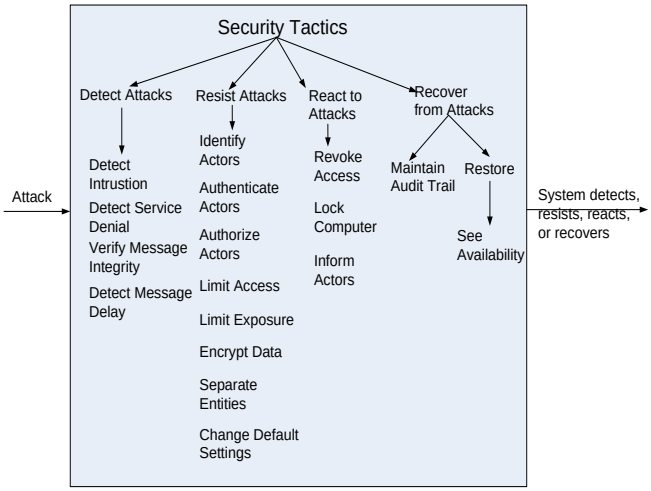
So the good news is that, for me as a software architect, I don't have to start from a blank page to achieve those desired quality attributes. To detect a fault, there's a set of tactics that I can employ. So I could use Ping Echo, I could use Heartbeat, I could use a system monitor, I could use exception detection, and so forth. So as an architect, I will choose one or more of those strategies and I will design to those, I will realize those strategies.

The tactics that we'll be talking about today are all available, all documented in "Software Architecture and Practice."

Security Tactics

Tactics provide a useful vocabulary for design and analysis.

But realizing them in code involves lots of interpretation.



**065 So let's take a quick look at the tactics for security.

So, here you can see that the security tactics are divided into four categories, so we need to think about what it means to take an architectural approach to security. We need a way to detect an attack-- how do we know that we're under attack right now? Given that we believe we are being attacked, can we resist attacks? What are the architectural strategies we can take to resist attacks? If we are unsuccessful in resisting attack, we need to be able to react to those attacks. And finally, if all of that fails-- our system has been compromised, we've had resources compromised, we've lost data, whatever-- we need to be able to

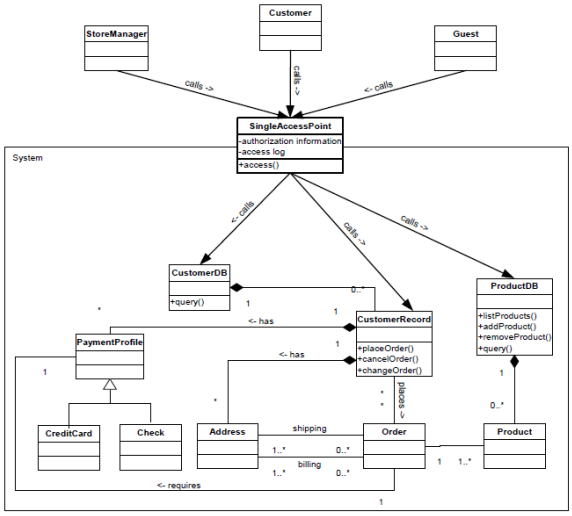
recover from the attacks, respond to them, and maybe even find the bad guys who did it.

So as an architect, again, there are choices that I can make, architectural decisions that I can make, to achieve each of those security goals, and each of those is an architectural strategy. So the tactics give us a really useful vocabulary for design, they give us essentially a checklist for design, and we've seen in design exercises in classes that we've taught for many years in architectural design that these are really useful resources for architects; giving them a checklist, giving them a set of things that are the primitives of design is really useful as a starting point. But they're only a starting point. You still have to realize an architectural tactic in code, and this involves a lot of interpretation.

Security Patterns

There are a number of well-established security pattern catalogs.

Patterns help to structure a design, but they are difficult to correctly implement, maintain, and combine.



**066 So one of the ways that we do this is by realizing an architectural tactic via a security pattern. In general, we realize tactics via patterns. And patterns are quite common these days. There are lots of books published on patterns an websites devoted to patterns. There are quality attribute communities devoted to documenting patterns in security, availability, performance, and so forth.

And patterns help us to structure a design. So when you choose a security pattern, you are given a description of the pattern, what it's for, the various forces that are at work, the tradeoffs involved, and you are typically given a design fragment, a UML diagram showing you the

pieces of that design and how they interact with each other.

And this is good. This is a big help to have this reusable design expertise. But patterns are notoriously difficult to correctly implement. So a couple of years ago I did another empirical study on the implementation of design patterns. And so we gave our subjects documented design patterns, they were taught about the patterns, they were taught how to use them, and then we let them implement them and we studied their implementations, and we found that about 70 percent of the implementations actually violated the design pattern, and violated them in ways that undermined the intent of the design pattern. The students got the systems to work, but they were introducing technical debt. So patterns are a great asset, but they're often difficult to correctly implement and maintain. They can erode over time. And no system is built from a single pattern. An architect may choose a number of patterns, perhaps dozens of patterns, in implementing an entire system, and those have to be combined; they have to live together and play together nicely.

Security Frameworks

Security Frameworks

A framework is: a reusable software element that provides generic functionality addressing recurring concerns across a broad range of applications.

There are security frameworks for many languages and technology stacks.

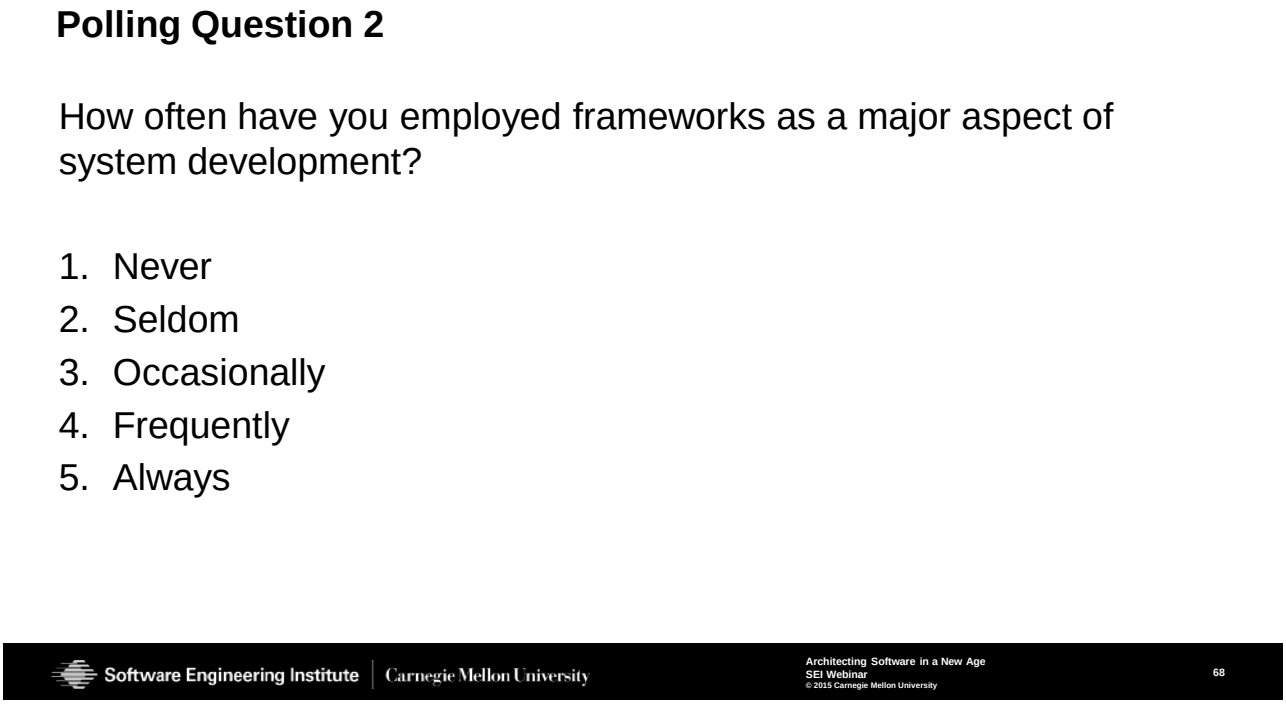
Frameworks increase productivity, but often have a steep learning curve and "lock-in".

**067 So one of the ways that we can deal with this complexity is by using a framework, because frameworks realize-- frameworks implement tactics and patterns. So a framework here is a reusable software element that provides some generic functionality, and this generic functionality addresses a recurring set of concerns, and these concerns recur across many applications. And as I said, you could build your own framework, for security or for any other quality attribute, but there are lots of commercial and open source security frameworks out there in the world, and there's just a few examples on the right-hand side of your screen right now.

These frameworks exist for many programming languages and

integrate with many different technology stacks, but nothing is free in software engineering; everything's about tradeoffs. So frameworks will often increase productivity in the long run, but they have a steep learning curve. You have to understand the concepts of the framework and how they work and how you can integrate them. And there's a lock-in. Once you have committed to a framework, the cost of switching can be quite expensive, and so you have to be very careful when choosing a security framework, or any framework.

Polling Question 2



**068 Shane McGraw: So we'll launch our third and final polling question today, and as we're doing that, Rick, we'll review the results from the first polling question. So

that polling question is launched. And then from the first question-- which was "Do you take an architectural approach to security?"-- we had 8 percent say never, 13 percent seldom, 41 percent occasionally, 27 percent frequently, and 11 percent always. So is that 38 percent, looking at this, surprising to you? Is that something you're seeing more and more?

Rick Kazman: I'm encouraged. I'm encouraged by that, but obviously it's still a minority, and so hopefully the evidence that I'm presenting will encourage more of you out there to at least consider an architectural approach to security.

Shane McGraw: Okay. We'll review the next one a little bit later.

Case Studies

Given this wealth of design concepts, we were interested to understand:

- how architects approach security,
- how well these design approaches “perform” in terms of securing the system and reducing the cost of creating and maintaining a secure architecture.

**069 Rick Kazman: Thank you.

Okay, so given this set of design concepts-- tactics, patterns, and frameworks-- we wanted to understand how architects approach security in the real world and how well these various design approaches perform. How well do they succeed in securing the system? How vulnerable is a system, given that you've employed one or more of these concepts? And also, how does it affect your cost of creating and maintaining a secure architecture?

Case Study Subjects

Case Study Subjects

Organization name	Description	Case study	Frameworks used
CodeOne	Creator of a security framework in Korea	"ACME" web application	CodeOne Security Framework (“After”)
Quarksoft	Software consulting firm in Mexico City	Internal project management web application	ZK Spring Security
OpenEMR	Open source project	Electronic health records system	None

**070 In our case study, we had three subjects. These three subjects were from different domains, located in different countries, working on systems of different size. So the first one was a system maintained by a company called CodeOne in South Korea. They've created their own security framework, and they had been hired by another company, whose name we cannot reveal. CodeOne had been hired to refactor a system that we've called Acme, which was a web application. And so when we look at the Acme system, we're going to look at Acme before, where there was no security framework employed, and Acme after, where CodeOne had refactored the system and had inserted their own security framework.

The second example comes from a company called Quarksoft-- oh, I should say that CodeOne in South Korea, we have direct contact with that company via Professor Ryoo at Penn State. The second one, Quarksoft, is a software consulting firm in Mexico City, which Professor Cervantes had a relationship with, and as a software consulting company they are dealing with and managing many projects simultaneously, and so they had developed an internal project management application, a web-based application, for managing and monitoring and collecting information about the progress of these projects; and they had developed this application from the ground-up using two security frameworks, ZK and Spring Security.

The third example is an open source project on electronic medical record system called OpenEMR, and they had written a system in PHP; they had not employed any security framework whatsoever.

So those were our three case studies, our three subjects.

Case Study Protocol

Case Study Protocol

1. Interview the architect regarding the approach to security, the size of the system, and the effort expended on security.
2. Scan the system to identify its vulnerabilities using *AppScan* from IBM.

Goal: explore tradeoff space between costs and benefits (effectiveness) of different approaches to security, and determine if there are optimal project strategies employing the approaches.

**071 And what we did in the case study is, first of all, we interviewed the architect. One person in the team had established a personal relationship with the architect on each of these projects. So we interviewed the architect regarding their approach to security, we asked them about the size of the system, and we asked them about how much effort they spent on securing the system, and then we scanned the system using AppScan, which is a commercial security scanner from IBM-- a more or less industry standard security scanner-- and our goal was to understand the tradeoff, the costs and benefits, of the various approaches to security to determine whether there's an optimal project strategy.

Interview Questions

1. What were your primary drivers (quality attributes for the system) and how important is security among them?
2. With respect to security, what are the approaches that you have taken to address this quality attribute?
3. How do you reason about tradeoffs?
4. How did you ensure that your programmers conform to the security approaches? (policies, inspections, etc.)
5. What percentage of project effort do you estimate goes into security without the use of a security framework? If using a security framework, what percentage of effort does this take?
6. Other comments

**072 So first of all, our interview: the interview took about two hours in total. We started off by asking the architect what were the primary drivers, what were the most important quality attributes for the system; and among those quality attributes, where did security rank-- how important was security.

For the second question we asked what approaches had they taken to address security in the architecture, and what we did there is we used our catalog of security tactics as interview questions, and I'll show you examples in a moment. So for each, we took each security tactic and we turned it into a question, and we asked the architect, "Are you doing this in your architecture? And tell us

something about how you're actually realizing his tactic."

We asked them how they reason about tradeoffs in the system, and we were also interested to understand how they know that their programmers conform to the architectural approaches to security that they've taken. As an architect, you can design a beautiful system with layering and with patterns and so forth, and then perhaps your programmers go and implement whatever they like with little or no relationship to what you've designed. One of the jobs of an architect is to ensure conformance of the implementation to the design. So we wanted to know how did they ensure conformance, how did they know that the programmers were really programming what was expected of them.

Then we asked them to estimate what percentage of total project effort goes into security, and if there was a case, as there was with Acme, of a before-and-after state, and with- and without-a-framework state, we wanted to know an estimate for each of those cases-- so how much effort went in with the framework, how much effort went in when they didn't use the framework. And finally, we had an open question portion.

Example Questions

Example Questions

Tactic	Description
Detect Intrusion	Does the system support the **detection of intrusions**? An example is comparing network traffic or service request patterns within a system to a set of signatures or known patterns of malicious behavior stored in a database.
Detect Service Denial	Does the system support the **detection of denial of service** attacks? An example is the comparison of the pattern or signature of network traffic coming into a system to historic profiles of known Denial of Service (DoS) attacks.
Verify Message Integrity	Does the system support the **verification of message integrity**? An example is the use of techniques such as checksums or hash values to verify the integrity of messages, resource files, deployment files, and configuration files.
Detect Message Delay	Does the system support the **detection of message delays**? An example is checking the time that it takes to deliver a message.
Limit Exposure	Does the system support **limiting exposure**? An example is reducing the probability of a successful attack, or restricting the amount of potential damage, e.g. concealing facts about a system ("security by obscurity") or dividing and distributing critical resources ("don't put all your eggs in one basket").

**073 So let's focus on Part 2 of the interview. This is where we asked about the tactics.

So, for example one tactic is detect intrusion. We wanted to know: Does the system support detecting an intrusion? So to do this, for example, you might have patterns of network traffic or patterns of service requests within a system that you store in a database, both normal usage patterns and patterns of malicious usage, and the system then compares the actual traffic with those patterns to determine if we are under attack.

Similarly, to detect whether you are under a denial-of-service attack, you would store patterns of network traffic coming into a system and

compare this to the actual patterns to determine whether there's a denial-of-service attack going on.

We asked them whether they verify message integrity. So does the system have some way, perhaps using checksums or hash values, to verify that a message that is being received by the system hasn't been tampered with.

Do they detect message delay? So, this is a way of determining whether you are subject to a man-in-the-middle attack, because that man-in-the-middle has to introduce some latency in the message delivery time.

Do you have a way of limiting exposure? So if your system is compromised, if somebody manage to break into the system, can you limit the amount of damage they can cause? Can you limit the amount of data that they can steal? So the idea is don't put all your eggs into one basket.

As I said, for each of these tactics we asked them: Are you doing this, and if so, how are you doing this?

Example Answers

Example Answers

Tactic	How is it achieved?
Detect Intrusion	- Primarily enforced through the use of hardware firewalls - Spring Security also guarantees that a session comes from a single place
Detect Service Denial	- Covered by ZK - Use of hardware Firewall
Verify Message Integrity	- Covered by ZK. All requests are associated with a checksum and IDs. Most of the processing is done on the server.
Detect Message Delay	- Covered by ZK. When a session is created in ZK, many short-lived objects are created and each has a UID. The UID is verified by the framework so it would be hard to replicate these IDs.
Identify Actors	- Covered by Spring Security
Authenticate Actors	- Covered by Spring Security. All URLs are handled by Spring Security, transmission of content is a responsibility of ZK
Authorize Actors	- Covered by Spring Security
Limit Access	- Covered by Spring Security. The system runs over Tomcat, Spring Security overwrites the JAS standard from J2EE (just roles were defined in the web.xml configuration file of the web server)

**074 And here's some examples of the kinds of responses that we got from the architects. So for Detect Intrusion, the architect said, "Well, this is primarily enforced through the use of hardware firewalls, and Spring Security guarantees that a session comes from a single place." For detecting service denial, this is covered by the ZK framework and the use of a hardware firewall. For identifying and authenticating actors, this is covered by Spring Security, as well in part by ZK.

Example Answers

Example Answers

Tactic	How is it achieved?
Limit Exposure	- Not covered. Perhaps the fact that the application runs in an intranet?
Encrypt Data	- Use of HTTPS
Separate Entities	- Database server is physically separated, Identity Manager is also separated (it uses a Windows Active Directory).
Change Default Settings	- Not supported
Revoke access	- This can only be performed manually through the Active Directory.
Lock Computer	- Spring Security blocks the user if there are several attempts at accessing resources for which permissions are not granted.
Inform Actors	- Not supported
Maintain audit trail	- Several audit trails: Web server (audits web access), Spring Security (audits access to resources), ZK also creates logs.
Restore	- Not supported

**075 We asked about limiting exposure, and in this case the architect said, "Well, we don't really do that. We don't have an architectural approach to that." Or, "We don't have an architectural approach to automatically revoking access." So if an actor in the system appears to be behaving suspiciously and we would like to revoke their privileges, we would have to do that manually.

Do you have a way of informing actors that the system is under attack? "Well, no, we don't have a way within the system of doing that."

Metrics Collected

Vulnerability metrics were collected using *AppScan* which categorizes vulnerabilities as: High (H), Medium (M), Low (L), or Informational (I). Application size was measured using CLOC and MetricsReloaded. Security effort was estimated by the interviewees.

**076 So that was our interview. We also collected some metrics, some objective measures of system quality, of security. And again, we ran AppScan over each of the systems, and AppScan categories vulnerabilities as high, medium, low, or informational. We were most interested in the high-consequence vulnerabilities. We also measured the application's size using a couple packages, and security effort was estimated by the interviewees.

Discussion

Our case studies represent three different security approaches, in terms of their architectural support for security (degree of adoption of frameworks):

- *Full adoption*: security framework used throughout the lifetime of the software, e.g. Quarksoft.
- *Partial adoption*: security framework is introduced in the middle of the lifetime, e.g. ACME “After”.
- *No adoption*: no use of any third-party security framework, e.g. OpenEMR, ACME “Before”.

**077 So, what we had in our case studies was a spectrum of different security approaches, in terms of the level of architectural support for security, the degree of adoption of frameworks. There was a full adoption case-- that was Quarksoft, where they architected from day one using security frameworks. There was the partial adoption approach, and this was what was taken by CodeOne. They adopted a system that did not have an architectural approach to security and they inserted that partway through the lifecycle. And there was the no-adoption approach, where OpenEMR and Acme before didn't use any security framework whatsoever.

Results

Results

Case	Acme Before	Acme After	Quarksoft	OpenEMR
Approach	No adoption	Partial adoption (CodeOne fwk)	Full adoption (ZK + Spring fwks)	No adoption
Size (KLOC)	7.93	8.55	16.56	255.6
Detected Vulnerabilities	H: 154 M: 50 L: 99	H: 0 M: 25 L: 99	H: 0 M: 0 L: 0	H: 8 M: 9 L: 475
# Tactics Employed	6	12	13	9
# Tactics in Bus Logic	5	5	0	6
Estimated security effort	20%	10%	3% (30% without frameworks)	20%

**078 Here are the results. You can see we have four systems represented here. There's Acme Before, Acme After, Quarksoft and OpenEMR, along with the approach of each. In the second row you see the size of each system in KLOC.

The third row tells an interesting story. Acme Before suffered from 1154 high-impact vulnerabilities. Acme After suffered from zero. Quarksoft, zero. OpenEMR eight. And again, remember, security is binary. Any number greater than zero is potentially a disaster for the system, so you really want to see zero high-impact vulnerabilities. That would be your desire.

We also cataloged the number of tactics employed by each of the

architects in the architecture, and you can see that OpenEMR and Acme Before, again, employed the fewest; Acme After and Quarksoft employed the most. But the next row, the number of tactics in business logic, tells you how much of the security was implemented by the programmers in the stuff that they had to write versus how much was in the frameworks. And there you see Quarksoft wins. All of the security was handled by the frameworks. They implemented zero tactics in their business logic.

Finally, they estimated security effort. In Acme Before and in OpenEMR, the two cases of no adoption, they estimated about 20 percent of total project effort went into security. In Acme After, partial adoption, that number goes down to 10 percent; and in Quarksoft, full adoption, the architect estimated 3 percent. These were independent measurements. They didn't know about each other, they didn't know about each other's responses, but you see a surprising consistency. Furthermore, the Quarksoft architect estimated that had they not used frameworks, based on their experience with other systems, it would have been about 30 percent of project effort devoted to security, which is pretty consistent with the 20 percent numbers for Acme Before and for OpenEMR.

Inferences from the Results

Inferences from the Results

1. The superiority of using security frameworks as an architectural approach, either through partial adoption or through full adoption.
2. The effort required for partial adoption is, however, significant when compared to the full adoption approach.

**079 So what can we infer from these results? Well, it's good to take an architectural approach to security. As an architect, you probably know that the thing that you do last sucks the most, and if you leave security as an afterthought, it's going to be more expensive and you will not do as good a job, and this is exactly what our case studies demonstrated.

Furthermore, the effort required for partial adoption, while less than no adoption, is still significant compared to the full adoption approach. So earlier is better.

Inferences from the Results - 2

Thus, we recommend the use of security frameworks from the early phases of the construction of a system (full adoption).

No big surprise: adopting a framework after the system has been built will clearly be more costly than doing so from the start.

**080 So we recommend the use of an architectural approach to security, as manifested by security frameworks, from the early phases of the system. We recommend the full adoption approach, and this should not be a big surprise to anyone.

Inferences from the Results - 3.

Partial Adoption is a sub-optimal but common way of adopting security frameworks.

⇒ Most developers and architects worry about functionality first and security (and other quality attributes) later.

**081 Partial adoption is suboptimal, but very common. Most developers and most architects worry about functionality first and security and other quality attributes later. That is suboptimal, but that is in fact the state of the practice, and that's probably why our polling results showed only 38 percent adopt frameworks frequently or, I should say, take an architectural approach to security frequently.

Conclusions

Conclusions

Why is it best to address security via frameworks?

1. while application developers may be experts in their domains, they are typically not security experts
2. even if developers are experienced in security, they should not write their own security controls
3. using a framework increases the likelihood that security controls will be applied consistently
4. delegating security issues to frameworks allows developers to devote their energy to application logic, increasing overall productivity

**082 So to conclude, why is it best to address security via frameworks? Well, first of all, application developers are typically experts in their domain. They understand their domain, but they're typically not security experts. And even if developers are experienced in security, they should not write their own security controls, just the same way, for the same reason we have separate quality assurance groups. We have separate testers. You shouldn't be writing your own security controls.

Using a framework increases the likelihood that security controls will be applied consistently. Having all of the security code in one place means it can be more intensively tested. Reusing a framework across many

projects decreases the likelihood that there are undiscovered bugs in the framework itself, and this allows the developers to focus on what they do best and what adds the most value to their project, which is application logic.

Future Work

Future Work

We are currently pursuing (and actively looking for) additional case studies

- Interview with the architect
- *AppScan* vulnerability analysis

**083 So, just a final comment, we're currently pursuing additional case studies where we interview the architect and do an AppScan analysis. We have one that will be happening quite shortly with a much larger system, but if anyone out there is interested, we can talk about it. So now I'm happy to take questions.

Shane McGraw: Rick, again, thank you. Excellent talk. Just to circle back on that last polling question,

which was "How often have you employed frameworks as a major aspect of system development?"-- again, it was about 60 percent that have not, or occasionally have not done that. So, similar result to that question.

Before we dive into our Q&A, let me just put a quick plug in for our SATURN conference, which will take place April 27th through 30th, and it's going to be in Baltimore, Maryland this year. SATURN stands for the SEI Architecture Technology User Network. It was something started by Linda. This is the 11th one. It has grown into one of the largest software architecture conferences in North America. There's a great program that's available now on the SEI website, or you can just google SATURN 2015, or go to the SEI homepage. You'll see a story on registration being open, and by registering for today's webinar, you will all get an invite with a discount code to attend SATURN 2015.

So let's get on to the questions. First one for Linda, during her talk. Chris had a talk about-- you mentioned a keynote talk from SATURN 2010. Was it Jim Highsmith? They wanted to know the last name.

Linda Northrop: It was Highsmith. H-I-G-H-S-M-I-T-H. And those slides are on our website, and it was a great talk.

Shane McGraw: One from Prasanth asking, "If one is using

domain-driven design, will that help develop a coherent architecture without a lot of busy work?"

Rick Kazman: Our perspective is that architecture is driven by the quality attributes, not by the domain functionality. So while it is important to analyze your domain, and you need to have a coherent view of your domain requirements, you will not get a coherent architecture solely by focusing on the domain. You in fact must focus on the quality attributes because those are what drive the architecture. And in fact, if you think about it, when you refactor an architecture, what you're doing is packaging the same functionality, the same domain functionality, in a different way to change some quality attribute, to make it more modifiable or higher-performing. So quality attributes and domain functionality are orthogonal to each other, so you have to pay attention to both, but it's the architectural concerns, the quality attributes, that drive the architecture.

Shane McGraw: From Dawn, asking, "To what extent does the acceptance of technical debt run counter to meeting each of the four technical challenges introduced by Linda, or does technical debt in fact comprise a fifth technical challenge?"

Linda Northrop: No, I think that technical debt is very much associated with accelerated capability, because in fact the reason why people accrue technical debt is to avoid the cost of delay-- in other

words, to get the system out faster. And so you make a shortcut. You do a clone-and-own. You take some code you've used before and you use it again, knowing that that will make a part of your system brittle and will make it more resistant to change later on. But in many situations, you can't afford the delay, and so it's all about accelerated capability. So I think it's part and parcel of our move to get things out the door faster, to accelerate, to take incremental approaches, and we make some decisions. And there are some companies that would go out of business if they didn't accrue technical debt because they have to get the product out at a certain time, and they'll take the hit for architectural decisions that aren't perfect. But the point about technical debt is realizing that you've made that compromise, and if that system is going to have a longer life than that release, then over time you're going to pay. And so you really need to manage, and to take not unnecessary technical debt. Sometimes people get excited about the metaphor and use it as a blanket excuse for doing lots of shortcuts, and that's not what it's meant to do. It's meant to make visible that we in life, in software development, will make shortcuts because-- take some shortcuts-- because that's life, and it's all about accelerated capability. One thing-- if I could add, Shane-- I concluded my talk but I didn't address the last couple of slides, and one gave thanks and acknowledgement to all the many

people of the SEI who have worked on the architecture agenda, and their names are there, and there's a figure that shows that I stand on the shoulders of giants, and literally I do, Rick being one of many at the SEI who have worked on all these techniques. So in particular, in technical debt, Ipek Ozkaya and Rod Nord are the experts, and there are a lot of publications out there that you can find a lot of this information. The second slide that I didn't talk about has the URLs for lots of the work that I gave an overview of. So if you want to find out more about technical debt or to get the Hard Choices game, you can go there and get that information, find it right on our website.

Shane McGraw: And before we go to Rick, just a reminder, those slides are available now in the Files tab on the console, so be sure to walk away with those materials, and also a reminder to fill out the survey before exiting today's event. So Rick, you wanted to chime in as well?

Rick Kazman: Yeah, just to add on to what Linda said, there is conscious technical debt that you choose to accept because you want to accelerate delivery. But there's also the unconscious technical debt, what I referred to as rust earlier, and that just accumulates, and frequently you don't realize that it's accumulating, but you see the consequences of it. Your system becomes harder and harder to modify, to debug, to evolve, to understand. And we have

a suite of tools now that allow us to identify that technical debt and even to reason about the economic consequences of it and the consequences of refactoring it. So both kinds of technical debt are important; the point is you need to be conscious of it, you need to measure it, and you need to manage it.

Linda Northrop: Right, and the tools that Rick just mentioned are part of this workbench that I described, and there are other tools that you can get from vendors. The point is, be aware, and be prudent.

Shane McGraw: Okay, from Carl, asking, "Linda touched on architecture for cloud computing. I agree that failing fast and cheap is a desired objective. Can she share more thoughts regarding software architecture to enable resiliency? I understand web services, service-oriented, is key, and interested in any additional thoughts.

Linda Northrop: Right. Well, it's a web services approach. But my point is, really the primary architectural precepts don't change here. People always think that they're getting something free. So, "I'm going to use the cloud so I get this SLA and automatically I get free performance, free security. Life is good." There's very little in life that's free, and certainly not in cloud computing, certainly not in a lot of the frameworks you're bringing on, and a lot of the tech stacks you embrace.

We develop software in a very different way than we used to. Almost nobody I know starts with a blank sheet of paper and does development and design the way we used to. We shop. We look for frameworks, open source, tech stack. We use web services that are tried and true. We go to cloud providers for storage and for computation power that we used to provide in-house. But in doing all of that, you are not excused from architecting your solution, and that's what I was talking about. And a lot of the principles that we address in terms of architecture are very relevant in cloud computing. Now, specifically, in taking so approaches, there is again a lot of information on the SEI website. I have about how, in fact, to take a service-oriented-based approach. But you see, lots of organizations who are building up their IT solutions, who are building their enterprise systems, and what we see more than ever is people who want to modernize their IT systems, who want to move from a non-cloud situation to a cloud situation, people who want to take some legacy systems and somehow give them a facelift-- and in doing so, you have to look at the architecture of those systems before you suddenly give up a lot of what you had done in-house before to the cloud. So.

Shane McGraw: From Richard-- this is one for Rick here-- "I am a SharePoint architect and considering security. SharePoint forces you to consider access, permission, security

to the farm, and how that is accomplished. So how do we, or should we, enhance that thinking?" And these questions come in-- they maybe came in before your talk, so maybe recap your talk, or your thoughts on that.

Rick Kazman: Yeah. So, I can't speak to the specifics of SharePoint, because I'm not familiar with its security architecture, but anytime you're employing a major component in your system, you have to evaluate the degree to which that component provides those security dimensions to you. So SharePoint provides, let's say, authentication and authorization, but it doesn't provide intrusion detection. I'm making this up, but I think that's probably plausible. So as an architect, you could use the list of security tactics as a checklist, a design checklist, determine which security aspects are missing from the component, like SharePoint, that you are adopting, and then figure out how to plug the gaps. Because if you don't plug the gaps, your-- the attackers will find them. Right? That's an assumption you have to make in designing for security, is if there is a gap, a bad person out there will exploit it. So the tactics used as a checklist gives you at least a starting point for reasoning about where the gaps are, and then you can think about what other tools or frameworks you might employ to address those shortcomings.

Shane McGraw: There's a couple questions we're not going to be able

to get to because we're down to about a minute, so I would invite you all to join the SATURN LinkedIn group, post your questions there. We can try to keep the-- continue the conversation there. So if you just go to LinkedIn and look for Groups, and check for SATURN, you can post your question there. So we'll end it with a quick one for Rick: "Is there a list of available security frameworks and their respective features and qualities that we can reference?"

Rick Kazman: Is there a list? I'm not aware of a publicly available list, but feel free to send me an email and I can send you a list of the frameworks that I'm familiar with, and we've now built up a pretty big list of those. But I don't know of any public resource at the moment for that.

Shane McGraw: Okay. Folks, it's three o'clock. That's all the time we have for today. Linda, Rick, thank you very much for your excellent presentations. We thank you again for attending today's event, and just a reminder, our next webinar will be on January 27, and the topic will be Advancing Cyber Intelligence Practices Through the SEI's Consortium, by Jay McAllister and Melissa Kasan Ludwick. We hope to see you there. Thanks, everyone.