Insider Threat and the Software Development Life Cycle
Transcript

Part 1: An Update on CERT's Insider Threat Research

Julia Allen: Welcome to CERT's podcast series: Security for Business Leaders. The CERT Program is part of the Software Engineering Institute, a federally funded research and development center at Carnegie Mellon University in Pittsburgh, Pennsylvania. You can find out more about us at cert.org.

Show notes for today's conversation are available at the podcast website.

My name is Julia Allen. I'm a senior researcher at CERT working on security governance and executive outreach. Today I'm pleased to welcome back Dawn Cappelli, responsible for leading CERT's efforts on insider threat. Dawn's first podcast on protecting against insider threat is one of our best sellers. Today we'll be discussing lessons learned from real life incidents that exploited vulnerabilities introduced during the software development lifecycle. So welcome back Dawn.

Dawn Cappelli: Thanks.

Julia Allen: So could you bring our listeners up to date? Since we last spoke, what insider threat research topics has your team been investigating since then?

Dawn Cappelli: Well you might recall from the last time we talked that we had been basing our research on 150 insider threat cases that we had collected with the [U.S.] Secret Service as part of our insider threat study. Those cases went from 1996 through 2002.So what we've been doing in the past year is collecting new cases because we want to update our case material, and so we've been looking for every case that we can find that has occurred since that study. So these cases occurred between 2002 and the present. And we've come up with approximately 100 of those, and we created a new database where we're capturing all of the information that we can about those cases. And so now, in the beginning of 2008, we'll finally be able to take all of that new data and use it.

So what we're planning on doing is releasing a new version of our Common Sense Guide. The Common Sense Guide talks about best practices for preventing or detecting insider threats. We're also going to be creating two new models. One of them is for theft of confidential or sensitive information and one of them will be for fraud. And we're also going to be creating an insider threat risk assessment instrument so that we can condense everything that we have on our website into one instrument that organizations can take and use.

Julia Allen: Well it sounds like you've got a pretty full calendar for the coming year.

Dawn Cappelli: Yes, we certainly do.

Julia Allen: Well so from some of your recent presentations I see that you are finding some pretty interesting cases of insider exploitation that link back to the software development process, which is our subject for today. Why did this particular aspect attract your attention?

Dawn Cappelli: Well our previous research focused on type of crime. So we were looking at cases by what did they do. Did they commit fraud, did they steal information, or did they commit insider IT sabotage? And so that was the angle that we always came at these cases from.

And then someone from the Software Engineering Institute asked if we had anything that we could present at the SEPG (Software Engineering Process Group) conference last year. And so we thought, "Well that's interesting, I don't know if we do or not." And we looked at the cases again only this time from that angle, did we see cases where there were software development lifecycle issues involved? And we were actually surprised by what we found because it wasn't something that we had set out looking for, but once we did look for it we found some pretty interesting cases.

Julia Allen: So this is pretty interesting. So what type of impact and losses in the cases that you looked at can be caused by this particular type of insider threat?

Dawn Cappelli: Well we had cases with many different types of impacts. They ranged from the company going out of business. There were fraud losses. The one case we had involved a loss of $691M. We had cases where driver's licenses were created for individuals who couldn't get a legitimate driver's license. We had disruption of telecommunication services at a telecom firm. We had court records, credit records, other critical data that was modified, and we had a virus that was planted on the organization's customers' systems by an insider. And these all were related to vulnerabilities in the software development lifecycle.

Part 2: Requirements, Design, and Implementation: Oversights and Flaws

Julia Allen: Well so let's explore that a little bit more, kind of by lifecycle phase, to make this a little more tangible for our listeners. So what are some examples of software flaws or omissions that can occur during say software requirements definition that might lead to a compromise?

Dawn Cappelli: What we found was that organizations, when they were defining requirements, they sometimes just forgot to think about authentication requirements, role-based access controls. They just forgot about separation of duties requirements so they were gathering requirements for a system which was automating certain business processes but they didn't think about separation of duties. Also automated data integrity checks – they didn't think about how can we check the data, not only for just defects in the software but also for suspicious looking data.

So I have an example here. We had a case where — this involved the state police in one of the states — and this was a communications operator. So this woman basically was supposed to use a system to look up information on driver's licenses when officers out in the field would call them in. And so she would go in and access the information and give it to the officers. Well one day an acquaintance asked her to look up information on three people, and she did. And when she did she just sort of accidentally discovered that not only could she access the information but she also had the ability to modify it. And so at that point they started on this scheme where she opened a post office box and she started creating new driver's licenses for people who couldn't get a legitimate driver's license. And the only reason that they caught her was because a confidential informant alerted them as to what was happening. And it ends up that she created 195 driver's licenses for people who shouldn't have had them.

So what we found was in the vulnerabilities introduced during the software requirements stage was typically – it was an oversight. So it wasn't a deliberate vulnerability introduced by the insider, it was an accident. And it was later down the road exploited by another insider who actually was an end user of the system and discovered that vulnerability.

Julia Allen: So in this particular case – this is pretty interesting because I think when people are building and developing software they don't think of these kinds of implications. But do you have any thoughts about, with the benefit of hindsight, how the team defining the requirements might have mitigated this during requirements definition?

Dawn Cappelli: I think they just need to be aware of the potential for insiders to abuse their access. I know I used to be a software developer and we did think about it, and I really credit the system owners that we were working with, the people who were defining the requirements with us, because they were very concerned about this confidential information in these systems and the people who would have access to it and what they may do with it. And so they were very careful to track what people were accessing, to track and send audit reports whenever information was modified.

And so I think it's just another step when you're defining your requirements to just think consciously about how could this data be misused by the people who will be using the system to access it.

Julia Allen: Okay, well let's move along the lifecycle, and do you have kind of a similar experience report or a perspective on flaws that show up, are introduced during design and implementation?

Dawn Cappelli: Yes, and the first thing that we sort of struggled with was how do we determine if a vulnerability came in the requirements stage or the design stage? And the way that we separate those is if they just completely forgot, then we call it a requirements flaw, but if they had it in the system but it just was not designed correctly, then we call it a design flaw.

So what we saw was systems where there were automated workflow processes defined and they just didn't give sufficient attention to security details. For instance, maybe they forgot about separation of duties – that would be a requirements flaw – or maybe they just didn't design it correctly and they didn't design any way for the check, someone to check the checker, basically.

Also we found that sometimes they designed authorized system overrides, kind of exception handling in their systems, and they didn't think about the fact that they were giving end users a way to get around the rules. And so an example of that is a system – this was again at a state agency that distributed food stamps – and two insiders ended up working together and they ended up making more than $70,000.00 in food stamp kickbacks. What they did was they realized that in the system there was separation of duties. It was implemented in the code, it was enforced technically. But there was also a special way to handle food stamp cases that were expedited; they just came in and they had to be put through quickly. There was a special process in the system for doing that. And if you entered a case as expedited then it could be executed without any supervisor's approval and it didn't have to have a legitimate Social Security number associated with it. So all of the built-in security aspects of the function were completely overridden if you just marked the case as expedited. So they ended up entering cases as expedited, they would generate these food stamps, then they would go back in and they would mark the cases as denied, so that they wouldn't show up on any of the caseworkers' reports because they just were denied. They weren't pending, they weren't approved, and as far as the system was concerned it didn't result in any food stamps. And like I said, they ended up then selling these food stamps for over $70,000.00.

Julia Allen: So by marking them denied they basically took them out of the normal checking system that would occur after a transaction was completed.

Dawn Cappelli: Right.

Julia Allen: Pretty interesting.

Dawn Cappelli: What we found in implementation was lack of code reviews allowed insiders to insert backdoors into the source code that they could then use later. So once you got past the requirements and the design phases and you got into implementation, deployment and maintenance, now you had the insiders deliberately inserting vulnerabilities so that they could use them later. And we didn't see too many actually inserted during implementation, we saw more of them during maintenance.

But we did have a case where a web application developer was developing applications and installed backdoors in his code. He ended up being terminated and he used those backdoors to get back into the organization's network. And he sent very malicious email to the organization's customers, he altered files, altered applications, initiated a denial of service attack, and the organization ended up declaring bankruptcy as a result.

Julia Allen: My goodness. So, and sometimes backdoors are legitimately inserted to help those responsible for maintaining the system to get in and do debugging and other types of maintenance actions. So it seems to me that the oversight or the code reviews or the checking for that kind of feature in the software, as you said, you have to create some training and awareness around the various ways those can be used.

Dawn Cappelli: Right. And if you aren't doing code reviews then you really have no idea of what is being put into that code, unfortunately.

Part 3: Deployment and Maintenance: Oversights and Flaws

Julia Allen: So do you have some additional examples you'd like to talk about during system deployment and maintenance?

Dawn Cappelli: Yes. During system deployment there were some basic oversights, like putting a system into deployment without setting up appropriate backup procedures, without making sure you have sufficient documentation. If you don't have sufficient backup procedures and an insider attacks then you're really in trouble.

We also had a case where they deliberately had a completely separate development environment and production environment. But when they released the system into production they used the same password file on both systems, and so the developers, although they should not have been able to access the production system, they were able to because that password file was overlooked.

Also just thinking about things, like I mentioned to you the one insider was able to deliberately plant a virus on all of the organization's customers' systems, because he was responsible for deploying any new releases to those customer systems, and there was no two-person rule or anything involved, so he could do whatever he wanted when he accessed their systems.

During maintenance, that's really where we saw the malicious code planted because this is when - organizations sometimes have very good processes in place during requirements definition, design, implementation. Then the system gets deployed, it's out there in maintenance, everything's going smoothly, and that's where the processes tend to fall apart. They also don't tend to practice real strict change controls whenever they have the system out there in the maintenance mode.

And so we saw various kinds of problems introduced at that stage. This is the one case that I mentioned where the fraud of $691M was created. That was because the end user had access to the source code for the system and he inserted his own malicious code into the system, enabling him to commit his fraud. Another one, the telecommunications example that I told you, this was a developer who was very disgruntled. Someone else got a promotion that he felt he should've gotten. So he went into this telecommunications firm's source code and he planted malicious code but he didn't set it to go off yet. Six months later he got a new job and before he left he set that malicious code to go off but not for another six months. So six months later, a whole year after he planted the malicious code, it finally went off and it disrupted their telecommunications functions.

Julia Allen: So how did they eventually track him down, given he'd been so smart about putting these time delays in his actions?

Dawn Cappelli: I don't know how they tracked him down but I know it was very difficult.

Julia Allen: I would definitely think so. And so it sounds like maybe authentication, absence of authentication or separation of duties and effective change management were absent in these cases where malicious code could be inserted during maintenance.

Dawn Cappelli: Right, particularly the change management. He didn't write a lot of malicious code. He just inserted a few lines. I believe that they actually did use the configuration management logs to figure out what had happened. So they had the logs but no one was reviewing the logs. And so that's a typical problem that we also saw in these cases.

Part 4: Software Process and Raising Awareness

Julia Allen: So this has been a great review of different things that can occur during the lifecycle, and you've certainly talked about lots of different challenges. If a business leader or a program manager wanted to start to tackle this issue head on, what are some of the challenges that they might face?

Dawn Cappelli: Well we have made two presentations of this software development lifecycle insider threat issue, and both times, well really the second time mainly, we got the comment that it's easy for us to say that they should do code reviews and they should have change management, but it's very difficult to really have the resources allocated to that. And so once you do have a system out there in maintenance and everything's fine, it is hard for them to justify the expense of dedicating someone to look at the change control logs and doing code reviews on every change that goes out. I think that's the main challenge.

Julia Allen: Because it seems to me when you think about software development there's always a big push, a big cost push, a big schedule push, add more features, get this fixed, get it up and running, and too some might feel that you're actually slowing down or jeopardizing the delivery date by putting some of these checks and balances in. Do you find that too?

Dawn Cappelli: Yes. That's what we heard at the one conference, which was more of a security-oriented conference. Now at SEPG where you have more software process oriented people they understood the importance of following the process but they really just hadn't thought about the security issues and the insider threat potential. So I think both sides can learn from each other.

Julia Allen: So given these challenges and some of the hurdles, what have you found in your work are some good, sound practices for at least starting to minimize this particular type of insider threat and risk?

Dawn Cappelli: Well for one thing the Software Engineering Institute of course is very process-oriented and has a very strong program on software process. And so if organizations follow good software processes then they can catch these things, they can mitigate this risk. But in order to mitigate that risk they need to be aware of the potential of insider threat. So at this point our mission is to really raise awareness so that first of all if you're not following good software development processes you do, and if you are following good software development processes you now are aware of the potential for insider threats, so you can build that into your process.

Julia Allen: I know in our build-security-in work that we're doing for the Department of Homeland Security we certainly talk about that as well, and this whole issue of trying to address the flaws and defects as early in the lifecycle as possible. You can get more of a cost advantage if you can find it in requirements definition versus waiting till operations.

Dawn Cappelli: Right, exactly.

Julia Allen: So this has really been very informative and a nice summary and build on your previous podcast. Are there some other places where our listeners can learn more about the subject?

Dawn Cappelli: Yes, if they go to our website, which is at www.cert.org/insider_threat, we have a lot of materials there on insider threat. And one thing that they may want to look at is the information that we've published on insider IT sabotage, because those cases where insiders are deliberately inserting malicious code for the purpose of causing harm, they fall into our insider IT sabotage case criteria. And so if they look at that, they'll see that there is a very distinct pattern of behavior leading up to that crime. And so that may give them some clues so that they know who to look at. So if you don't have time to do code reviews for everything that goes out, then at least it will alert you that you may have this malicious insider, and so you need to look at what they've been doing; so then you look at the code that they've written, look at the configuration control logs for what they have released.

Julia Allen: Right, because we all know that you can't secure everything, you can't review everything, so having the kind of criteria that you're describing allows you to focus your attention on the things that are – maybe have the highest likelihood of exploitation.

Dawn Cappelli: Exactly.

Julia Allen: Well Dawn, thanks so much. It's just been fabulous talking with you again, and I just am hopeful that we'll be able to follow your research as it goes forward and have another conversation.

Dawn Cappelli: Okay, thank you very much.