How to More Effectively Manage Vulnerabilities and the Attacks that Exploit Them
Transcript

Part 1: Vulnerability Counts Likely Low by an Order of Magnitude

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 principal researcher at CERT, working on operational resilience. And I'm very pleased to welcome back my colleague, Art Manion.

Art is a member of CERT's Vulnerability Analysis Team. And today Art and I will be updating his prior podcast that we captured in 2008, discussing how security vulnerabilities in particular have evolved, and how business leaders can more effectively manage the thousands of new software vulnerabilities that are flying at them every year.

So welcome back, Art, really glad to have you on the podcast series again.

Art Manion: Thanks, Julia, great to be back. And it's been four years, it sounds like, what you just said; it's a lot longer than I thought. So happy to be back and I've got some things that have changed I hope, in the vulnerability world, to talk about.

Julia Allen: Yes, I think we have a lot of good content to discuss today that people will find interesting. So just to set the stage for those that aren't real familiar with our work, can you start with defining what -- when you talk about software vulnerabilities what are you talking about?

Art Manion: So our team here, Vulnerability Analysis at CERT -- we're talking about vulnerabilities; it's a more narrow definition than the broad Webster's dictionary vulnerability or general weakness or an exposure. It's almost always a vulnerability that's in software. So we often say 'software vulnerability' or 'security bug' or something, to sort of narrow that definition down.

There's roughly three classes of things. We'll call them an implementation defect -- so somebody writing the code has made a mistake and that becomes a vulnerability because it allows somebody to break into the software later. There are design time vulnerabilities -- maybe not picking, not designing a cryptographic protocol correctly or not expecting a certain type of attack.

And somewhat related to design, we talk about a less tractable thing -- but changing environmental conditions. So something was designed to work in the internet in the late '90s; 10 years later, in internet time especially, its security properties are not appropriate for what's happened now 10 years later. So that's not a design issue or a bug somebody wrote into the software. But nonetheless, because things have changed, the environments' changed around the software; now there are vulnerabilities in there.

So more narrow than the broad definition -- a software vulnerability is something related to the software that's going to allow somebody to break in, steal your data, tamper with your data, deny service to you -- that sort of thing.

Julia Allen: Great. So I think it's obvious in your definition but let's spend a little more time talking about why we care about vulnerabilities in the first place. How do they make our systems more susceptible to compromise?

Art Manion: Well the basic problem here is we're an internet-connected society. Our valuable data is stored on computers and on networks, and people who want that data -- and these might be financial criminals, it could be nation state actors, all kinds of things -- that's now where they're going to go to get that data or to cause trouble.

So attackers (the term we generally use) are coming after your systems. And a vulnerability is, not always but very often, part of the chain of events that can lead from an attacker somewhere on the internet, and all the steps they have to take to get into your computer and steal data for instance.

We often use terms like an attack graph or a tree. And you can follow the leaves and the branches to the tree and see how an attacker started from their starting point and how they ended up with stealing the crown jewels basically. And a vulnerability or more than one vulnerability are often nodes in that graph.

Another analogy I've seen used a lot is an attack chain or a kill chain. And at some point in there the attacker is very likely exploiting a vulnerability, a weakness in software, to get to the next point in the chain and get one step closer to their goal.

So vulnerabilities are part of the system that allows somebody to break in. That's why we study them. That's why we want to try to prevent them: either tactically, patch it; or strategically, how do you talk about managing vulnerabilities, the number of them? What are some policy concerns that might mitigate the effects of vulnerabilities?

Julia Allen: So I know you've got some great examples of some of the common attack vectors that you've seen, and some recent examples where the attacks are becoming more, much more sophisticated. Can you say a little bit about those?

Art Manion: Sure. A very common attack vector -- I guess I'll say in the past, it's been at least three or four years, probably before the last podcast. And I think the listeners out there will probably have read about this.

An attacker will target someone. This is a targeted attack potentially. They're after a certain company. So they might send email to people at that company. They might even do a lot of research and find a person in a certain position who is more likely to respond or be closer to the attacker's goal.

So an email that might be well-crafted -- it's effectively a phishing email. It's trying to entice the recipient of the message to maybe open a document, or click on a link to a document or to a website. In advance now, the attacker has found a vulnerability, perhaps in a PDF reader or Adobe Flash or an Office format. And the document they've attached in that email will look enticing, it'll look legitimate.

As the recipient opens that document, the specially crafted document that will actually exploit a vulnerability, install some stage of the attacker's attack tools on the system. And now the attacker has gained a foothold or they've gained one more link in their chain towards their goal.

So there's a social engineering aspect (a phishing email, an enticing email) and there's a vulnerability and exploit aspect (specially crafted document attached) that exploits a vulnerability on the user's system. There are lots of variants of that. But that's a very basic type of attack. It's proven fairly successful unfortunately, and it's still in widespread use in different variants of that.

Julia Allen: And what about some of the more sophisticated ones that we've been hearing about in the press over the last year that are more troublesome where it may not be so obvious?

Art Manion: Well so right, there's a range of this type of attack. And the very broad financial crime ones that are generally after your banking information -- those might be pretty obvious to even an untrained eye. The emails aren't written well. The English is not correct. They look suspicious right off the bat. The filename is suspicious. This looks like spam, this looks like a scam coming in sense.

A more carefully targeted attack does not look like that -- much better written email; much more enticing message; much more legitimate seeming. And you can look at the news stories in the past several years, and I see names like RSA, Lockheed Martin. I see Google published several years ago a class of attacks that hit them and other major software vendors, and those were called Aurora in the press I believe.

So these are companies that are not new to information security. And I would doubt that any of them are just running around without IDS, without patching, without antivirus. They're doing the basic security things. But a motivated attacker, a motivated and capable attacker, is able to find an unpatched or a zero-day vulnerability and craft this enticing message and get a foothold into these companies using a zero-day vulnerability and the enticing message.

So the bad news defensively here is even if you're doing everything right, this attack vector is still effective.

Julia Allen: Got it. And I know, just in terms of looking at sheer numbers, what do you see in terms of a shift or a change in the characteristics and volume of vulnerabilities in the past year or so?

Art Manion: Well to be honest, we've stopped -- CERT and the Vulnerability Team -- we've stopped counting. We used to try to catalog every public report of a vulnerability. And in late 2008 we stopped doing that. And even looking at our old numbers and other databases that currently still do try to track public disclosures, the number I usually use is about 8,000 a year.

If you read the mailing lists, look at the databases, look at blog posts, follow the security conversations in Twitter, look at vendor disclosures, Microsoft patch Tuesday, Oracle patch releases, you get to about the 8,000 range. So 10,000 max per year.

Other work we've been doing recently at CERT, which we call vulnerability discovery -- and we're focusing on fuzz testing tools, which I'll be happy to talk a little more about -- that has shown us that this 8,000 to 10,000 a year number is probably at least an order of magnitude low; too low.

We can automatically find bugs and many of those bugs are vulnerabilities. And we've found so many just in developing our tools that we haven't even been able to report them all to vendors. And we can easily overwhelm a vendor by sending them dozens or even hundreds of crashing, examples of a crashing test case that could be a vulnerability.

So it's not Microsoft patch Tuesday; there are five bulletins, there are three vulnerabilities in this, one in that, one in that, two in that; maybe a dozen vulnerabilities total; maybe half a dozen patches cover all of them.

The actual scale is much greater and it's not, it's like we're seeing the top of the iceberg. And then what's not being talked about is well there are many, many more crashers, crashing test cases (we call them crashers) and those might also be vulnerabilities. And we just don't have the capacity to handle them. But they are there. An attacker just needs one to get in.

Julia Allen: Right, we have to defend all points of entry and they only need one, right?

Art Manion: Yes, and that's a classic defense problem basically.

Part 2: Fuzz Testing; Security Hygiene; Vulnerability Intelligence

Julia Allen: So let's dig into this. Your order of magnitude statement is really alarming. And if I think about it -- if I'm faced with having to manage this environment and this scenario that you described, just the notion of first of all what's out there, and being able to prioritize, is pretty challenging.

But I'd like to do a little bit deeper dive on the discovery and fuzz testing work that you talked about for vulnerability discovery. So can you say a little bit about the work that CERT's been doing in that space?

Art Manion: Oh yes, absolutely. So first let me just explain fuzz testing. That may not be the most common term. The term dates back probably several decades in the computer industry. Basically it was a way to test the robustness of a program. So you take an input to the program, probably a known good input -- and in the case we're talking about this is probably a properly formed PDF file, for example. PDF is a good example. I'll stick with that.

So take a properly formed file and feed it to the program. Open it with Adobe Reader or Apple Preview or some PDF reading program. And that's your control case. It opens correctly; it displays correctly; nothing unexpected happens; no problem. That's the control case, great. Close the program and start over. Take your input file and modify it in some way. And in fact what we found -- we call this dumb fuzzing-- you can modify the file almost randomly. You can flip some bits, change some bytes, add something to the end, take something out. And feed that again to the program. Open it with your, the target application -- Adobe Reader, whatever your PDF reader is.

So basically you're taking known good input, manipulating it in some way, opening it with the program, observing your results. And that's just, that's the loop, that's the fuzz testing loop. And you do that over and over and over again. And of course we have a program for a computer to do this because we don't want to do it -- sit down clicking ourselves through all this stuff all day.

And what's interesting is not when the program correctly opens the file, or it tells you, "Hey this file's corrupt, I can't open it." What's interesting is in between there if the program tries to open the file but the program crashes.

So you do a fuzz testing run. Maybe there are a million fuzz testing cases that get fired. Maybe thousands of those result in the program crashing in some way. Maybe hundreds of those, the crash shows evidence that it might be a security vulnerability, not just a crash that killed the program and stops the processing.

So as we're refining from a million test cases to a thousand crashers to a hundred crashers that look like they might be vulnerabilities, that's where things get interesting. And defensively we're looking at those hundred crashers and reporting that to a vendor. Or we're giving our entire toolset to vendors actually and suggesting they run it themselves. Those are what are interesting in terms of getting them fixed because they are serious security bugs. From an attacker's perspective, those are things that are interesting because that's something I might use to develop further into something that's actually an exploitable attack.

Julia Allen: So attackers obviously have access to these tools as well and they can do, use the approaches that you described to find an exploitable vulnerability that might be more stealth or might be more sophisticated than one that's more easily detected, right?

Art Manion: Yes. And in fact it is, it's a race. If a vendor finds, fuzzes their software and finds the bugs that the fuzz testing tool can find and fixes them before an attacker finds those bugs and exploits them, that's one way the path, the race can go. If the attackers are in advance, then that's another way the race can go.

And again, as you mentioned earlier, we have this classic defense problem where the attacker just needs one unpatched avenue in. And the defenders need to find lots or almost all of the bugs and fix them.

Julia Allen: Well, so I think we've painted a fairly challenging, daunting and perhaps grim picture of what we're up against. So why don't we turn our attention a little bit to the solution side of the equation?

You did mention a couple of the traditional approaches: IDS, IPS, scanning, other types of automated support for prevention. But what do you find are some of the more effective approaches? And I know we're going to talk to some promising research in the next part of our discussion. But what do you find are some of the more tried and true that seem to still be fairly reliable, fairly useful as a way to help mitigate this risk?

Art Manion: Sure. Well yes, it's a pretty bleak picture when you think about it. And that's exactly what we thought, sitting around and finding that there were more vulnerabilities than we thought; that it was very difficult to manage the numbers. The conclusion we keep coming to is there's an effectively endless supply of vulnerabilities.

We keep seeing reports coming in. We keep finding things ourselves. Other people keep finding things. Every so often there's some report in the press: there's a new zero-day vulnerability found in used by an exploit in this program, etc. So I've not seen, in 11 years at CERT, I have not seen the rate go down at all, basically. So effectively an endless supply; bad news for the defensive side; not a very great looking picture.

There are lots of, I'll call them at this point, standard defense-in-depth techniques that do help. These are basically if you're on the internet operating with people sending you email and running your own web servers and things, these are things you just have to do just to be at the baseline anymore. And of course you need to patch your systems. And patching should rarely be an emergency situation.

Change management, inventory management, scanning for vulnerabilities, keeping track of which systems are patched, that really needs to be built in to your infrastructure. Patching should be a regular occurrence, whether that's monthly, weekly, every quarter. Figure out a schedule that works for your business and your operation cycle. But patching should be a normal everyday sort of thing. Patching takes care of known vulnerabilities if you can patch in time.

For things that are unknown, it starts to get a little bit trickier. That's where it's possible that an IDS or IPS signature might be out before the patch is ready. It might give you some extra defense. Antivirus operates in a similar way.

Even more philosophically: principle of least privilege; separating services, external web services; stay away from your internal databases. Limited the least access required to your users; ACLs that reflect that sort of least privilege. All of those things can help contain or mitigate attacks or limit the damage when they happen. I did just mention antivirus and intrusion detection, intrusion prevention. Those are things that can help you, help fill the gaps between a patched vulnerability and something that is being exploited but isn't patched yet.

I do want to add a bit of a caveat there. Most IDS, IPS and antivirus -- they're detection based. So what I mean by that is yes they can detect things if they know about them. So again there may be a window; a zero-day exploit is found; IDS signatures are available before the vendor's tested patch is available. So you have that extra time of protection with the IDS.

However, before that vulnerability is known to the IDS people, no one knows about it. And you can't detect something you're not looking for. So I would suggest the IDS/IPS/AV stuff -- probably necessary but do consider how much you're getting, how much of an extra window of defense you're getting by having those detection signatures in place.

Julia Allen: Right, because we've talked about this before -- there are these standard, like you said, baseline or hygiene kinds of things. But in our preparation, I think you also talk about some things that might get you a little bit further down the path, like white listing and knowing what some of your exposures are and anticipating that it's not a question of if, it's a question of when. So are there some things in that middle ground that might get you a little bit further down the road in terms of protection?

Art Manion: If you're able to white list things -- you can white list at a host level, at an operating system level, a list of known programs that are allowed to execute. That's always a good idea. It's a great firewall policy: only allow things that you know should be allowed.

There's some interesting article (I don't recall the author) but it talks about, it uses the term 'vulnerability intelligence'- vulnerability intelligence project. And the idea there I thought was very interesting is that if there are 8,000 vulnerabilities disclosed in a year, it's a big number, very hard to manage, right? But how many of those do you actually, really, really matter to you?

Probably all 8,000 aren't going to be exploited or you don't run systems that have all 8,000. And the slide deck that I saw talked about maybe there are a couple of dozen that are widely used, for instance, in the baseline financial crime browser drive-by, steal your banking, steal your banking information kinds of attacks. And of those a handful of patches and a handful of browser configurations: things like turning off Java in the browser, turning off Flash in the browser.

Those things, some of those mitigation techniques can really I guess give you a much more efficient return on your investment. For instance, we advocated at CERT removing Flash support and 3D rendering support from Adobe Reader because that was an attack vector. And as best we could tell it wasn't a very widely used feature of the PDF reader.

So that requires a system administrator or an organization to go through and change permissions on a DLL file or remove some functionality from some software. But the result is a whole class of attack is now off the table. You didn't patch one vulnerability used by one attack, you've eliminated 3D rendering or Flash in a PDF file from being used as an attack vector. So in this vulnerability intelligence way of thinking, finding some configuration options or some mitigation steps that remove more than one vulnerability at a shot is a really interesting idea and we advocate that sort of thing.

Julia Allen: Right, and then that way you can start to begin to get, as you said, get closer to prioritizing what's most important to you, dealing with whole classes of vulnerabilities, and make that 8,000 number much more tractable in terms of, if you will, managing risk, identifying where you're at most risk.

Art Manion: Yes.

Julia Allen: And then putting in mitigations to address that, right?

Art Manion: Yes, absolutely. You probably won't have to patch all 8,000 or mitigate all 8,000. Figure out which ones you're getting hit with, which ones you're likely to get hit with, and handle those first, absolutely.

Part 3: Game Changers that Can Thwart Attacks

Julia Allen: Excellent, excellent. Well I know that you and others in your research community have started to identify and implement some promising solutions that really help get to root cause at the system level. And I'm eager for us to get into that part of our discussion.

So can you tell us a little bit about some of the techniques that you're most excited about going forward that could, perhaps even are starting to be game changers in this whole arena?

Art Manion: Yes. As I mentioned earlier, the bad news defensively is there's this effectively endless supply of new vulnerabilities. The numbers are an order of magnitude higher than the eight or ten-thousand a year that we're seeing; a bleak picture, as you mentioned.

The good news defensively is that there are operating system level techniques that are generally called exploit mitigation or exploit prevention. And the idea here is look at the scale of vulnerabilities. We're not going to be able to patch them all in time. We're not going to know about them all in time. There are going to be zero-day style attacks. Someone's going to get, find a vulnerability and exploit it.

So the vulnerability is there. We haven't been able to patch it; haven't been able to detect it. The attackers have gotten that far in their chain. They're exploiting the vulnerability. But what we can do at the operating system level are these exploit mitigation techniques that make it very difficult for the attacker to do something useful. And usually the attacker needs to -- CERT's terms are exploit arbitrary code -- run some program of their choice, install a backdoor, install a Trojan horse, install remote control software, install something that's going to exfiltrate your data. They need to run their program and their code and their software on your computer in order to do that.

These exploit mitigation techniques make that difficult and in some cases very, very difficult for the attacker. And these techniques have been around for years, in the research community, and they've been in operating systems in various ways and forms for years: Linux, the BSD community. Apple OSX has them; Windows has them. All these major operating systems have their own version of these core techniques.

What our team has been very interested in recently is that Microsoft, for the very widespread, predominant Windows platform, has released a tool called EMET. And that's E-M-E-T and it's Enhanced Mitigation Experience Toolkit. And EMET's this small utility that actually exposes these mitigation techniques to the user or to the administrator, to make it easy to turn these things on and see how they're applied to your system and get these things in play. And that's really something we've been advocating a lot recently. Having EMET turned on and having these techniques turned on really does buy you a lot in terms of defending against an unpatched zero-day attack.

Julia Allen: So just, because I know we're going to talk a little bit about some of these techniques. But just to make sure I understand what you're saying. You're basically saying is that you assume the attacker is going to get in. You assume that they have some arbitrary code that they need to execute to be successful. And these tools are intended to either make that difficult or impossible for them to do, right?

Art Manion: Yes. And it's a very interesting technology arms race here, the offense versus the defense. And let me get into some of the -- I'm not going to cover all the techniques. There are five or six classes of things that Windows does and most of those things are done on other platforms as well. Some of them are Windows specific, just due to the way the Windows architecture works.

Julia Allen: Right. But why don't you just maybe walk us through a scenario or an example and how these tools might be used in minimizing the damage?

Art Manion: Sure absolutely. And I'm only going to touch two or three of them because they're the main techniques that matter the most.

Julia Allen: Great.

Art Manion: So in terms of attacking, there's the classic, if you will, stack buffer overflow. And without getting into too much detail, this is probably the most simple type of attack where the attacker provides too much input. It flows past what the program has allotted in memory to store that input and the attacker can overwrite parts of the program that control how the program works.

So there's a piece of memory called the stack, and you write over the stack and past the end of it. And the attacker can trick the program into running whatever they've put in there, past the end of this piece of stack memory. So that's the attack. That worked, still works today sometimes. So there the attacker is leading the race; they can do this stack buffer overflow.

An exploit mitigation technique is called Data Execution Prevention or DEP for short; sometimes it's called No Execute, NX, in different operating systems. And this takes advantage of a little piece of some actual hardware in the processor, the CPU. And this marks the stack memory as not executable. So let's say the attacker finds a vulnerability.

They exploit the stack buffer overflow and they install the program they want to run past the end of the stack, and they go to run it. Guess what? They can't because No Execute, Data Execution Prevention. They are not allowed to execute data at that point in memory. So there's the vulnerability. They exploited it. But their program doesn't run. So advantage defenders. Attackers aren't sitting around idly. They want to get around this Data Execution Prevention technique. So there's a technique where the attacker has to find somewhere else in program memory to find somewhere that they can execute.

So that's possible. The attacker knows that they have a test system and they install it and they can find some other code, some part of a program, or some library has loaded at a known location in memory. And every time you run the program, it's at the same location. So the attacker can now, when they attack the system, they can install the program they want to run at this known location, jump to it, and run their code. They're no longer in the stack; they're no longer protected by, they're no longer up against DEP, execution prevention. They're somewhere in memory where they're allowed to run again. Advantage attacker. They can now run their code.

Defenders are not sitting around idly either. So the next technique is ASLR Address Space Layout Randomization. The idea here is since the attacker is trying to find a known consistent location in memory to install their program and run their code, let's make that hard. Every time the program runs or the system boots, it randomizes where things are loaded in memory. Now the attacker can't reliably guess where their program's going to get installed and where they can go run things.

Julia Allen: Right, because the locations, the addresses have all changed, right?

Art Manion: Right. The attackers' test system isn't the same. Every time they try to test it's not the same anymore. It changes each time the program's run or each time the operation system is booted. And it has to be statistically difficult for the attacker to guess right. Maybe they guess right one in 16 times or one in 256 or something like that. But that's still not very reliable for an attacker. They don't want, that's not good enough, right?

Julia Allen: Right. They don't want to have to work that hard, right? Because their exploit tools are all automated as well.

Art Manion: Right, and they want them to be reliable. And a weaponized exploit is going to work every time or 80 or 90%. They want some high efficacy for their tools. One in eight is not good enough. One in 256 is not good enough. So ASLR has defeated that technique. We're back to advantage defenders; they're now, we're now a step ahead on the defense side.

Julia Allen: Excellent.

Art Manion: Okay, the bad news is the attackers have stepped up one more time. So they're going to scan through memory. Even though they don't know exactly where things are going to be loaded, they're going to look around and they're going to find existing code -- not their own code but stuff that's already in part of the program or part of another library that's loaded.

They're going to find something called a ROP gadget. I'll explain in just a minute. ROP is a technique called Return Oriented Programming; and a gadget, a ROP gadget is, it's two, three, four, a handful of instructions that already exist in a legitimate program but the attacker can use those gadgets and put them together in a way that the attacker can still execute the code that they want and control the program execution.

So that's a way to try to get around ASLR. It's very difficult with ASLR. Let me just mention quickly one more technical bit of this. ASLR is really only effective if everything, everything loaded in memory is randomized. Because just one library that's not randomized -- the attacker can exploit that one and find the gadgets they need and find the path they need to execute something.

So what we're advocating is to have DEP turned on with EMET, and ASLR turned on for everything. And if you don't, if it's not on for everything, you're leaving an opening basically. And that's enough for an attacker to get through and defeat these things.

Julia Allen: Great. Well before we come to our close and leave this topic though, so let's talk a little bit about obviously these kinds of protection approaches, these exploit mitigation techniques don't come for free. So what have you observed might be some of the performance and other types of impacts of putting these tools in play?

Art Manion: Well when people started talking about these techniques -- and this was years ago; and as you know, computer and internet time are much more compressed than the real world time -- one of the initial concerns was a performance hit. If my program and my operating system have to do these extra checks and randomize things and do some extra checking before it executes code, every operation's going to have a performance hit that's going to add up. Is my busy database server going to get slowed down? That's a real valid concern.

Fortunately in today's terms of computing power, none of the stuff in, none of the techniques in EMET are really introducing enough overhead to be noticed. Maybe, maybe on a very busy system -- probably a server class system that hopefully is not exposed to these types of attacks very often -- maybe there's a difference there that could be noticed but I'm not sure that's even possible.

So performance is really not an issue for EMET and these techniques. But there are other side-effects. And most of those are, have to do with application compatibility. Older applications, especially things that were not designed with the ideas of DEP and ASLR in mind, some older programs won't run at all with DEP enabled; or they won't run at all with ASLR enabled; or they'll run for a while and crash when a certain operation occurs. Something the program is legitimately doing will get tripped up by one of these mitigation techniques.

So the advice for an organization considering turning on any of these features in EMET is test. Test your applications, especially the older stuff; run some systems with EMET turned on; monitor for crashes and odd behavior. But once you've got, you're through your testing period and your

applications and your systems are working well with EMET turned on, really, really encourage people to roll this out. We have a great presentation by a colleague of mine who he shows an exploit that works; turns on EMET; the exploit doesn't work. And it's not, nothing's 100%. There's no silver bullet here, it's not 100% protection for zero-day attacks. But turning on EMET really can protect you against an unpatched vulnerability, a zero-day exploit. EMET can stop those things. And that's great news defensively.

Julia Allen: Excellent, excellent. Well I sure appreciate your taking the time to describe this with such great examples. And I know you and I have talked about maybe doing a follow-up podcast that does a deeper dive on some of these tools and techniques.

So I hope we get a chance to do that. But I do need to bring us to our close. So do you have some places that you'd recommend that our listeners check out for more information?

Art Manion: Sure. So the main, one of the main sort of outputs of our analysis work are documents we call Vulnerability Notes. And there's a Vulnerability Notes database on the CERT website. So when we find a vulnerability in something, or someone reports one to us, we talk to vendors, we work out a publication date, we list who's affected, we list how to fix the vulnerability. All that information is in these documents: Vulnerability Notes database.

And we also, of course, have a blog where we talk about things that are less specific than "this vulnerability in this software." But in our blog we've talked a lot about the fuzz testing and the vulnerability discovery tools I mentioned earlier. We've talked about the use of EMET and exploit mitigation techniques on the blog. So the CERT/CC Vulnerability Analysis blog is another place you can find more information.

Julia Allen: Excellent Art. Well thank you so very much for your time today, for your preparation, and for all of this excellent guidance and updates for our listeners. Really appreciate it.

Art Manion: You're welcome. Great to be here. Thanks.