SECURITY NOW!

Transcript of Episode #266

Inside OAuth

Description: This week, after covering some rather significant security updates and news, Steve and Leo plow into the still-evolving Internet OAuth protocol. OAuth is used for managing the controlled delegation of access authorization to third-party websites and services. It sounds more confusing than it is. Well, maybe not.

High quality (64 kbps) mp3 audio file URL: http://media.GRC.com/sn/SN-266.mp3
Quarter size (16 kbps) mp3 audio file URL: http://media.GRC.com/sn/sn-266-lq.mp3

Leo Laporte: This is Security Now!, with Steve Gibson, Episode 266, recorded September 15, 2010: Inside OAuth.

It's time for Security Now!, the show that covers all your security needs; privacy, too. Steve Gibson is here. He is from GRC.com, the Gibson Research Corporation. That's where he works to save us from ourselves. Hello, Steve.

Steve Gibson: Hey, Leo. Great to be with you again, as always.

Leo: We have a great subject today. I'm really excited about this one.

Steve: Yeah, an important subject. It's a, well, it's an evolving/devolving protocol which addresses a very important need for the Internet and all of us. The protocol is known as OAuth. It was originally named OpenAuth, but it turns out there was a name collision. Somebody else had something called - I think it might have been AOL - had something called OpenAuth. So the original authors shortened it to just OAuth, but it means OpenAuth.

And the need it addresses is to allow a means for an agent of some sort to act on your behalf with an Internet service, without needing to have or know your credentials. That is, well, for example, the way I first encountered it, I think, was as I was playing with Twitter. And I think it might have been the Bitly service, which is a link-shortening service. I went over to Bitly and wanted to set up an account. And with Bitly it doesn't just shorten the links. You're able to put your whole message in there. And when you paste in a big link, it goes, whoa, that won't fit in the 140 characters that you have for a text message. So it scrunches it down. And the point is that it will then post it on your behalf.

Well, in order for it to do that, it has to have access to some aspects of your Twitter account. Now, if we didn't have a means to achieve this, when you set up an account with Bitly, it would have to ask you for your username and password so that it could literally log in as you and then act on your behalf. The problem is you really don't want to give your username and password out to all of the growing number of things that you want to have able to act on your behalf in a way like this. And you could imagine printing services, you might have all your photos over on Flickr, but then there's some other facility that you want to give limited controlled access to resources that are up in the cloud, which you have access to, but you don't want them to just - you don't want to keep giving out your username and password for all these different things.

So what happened in the case with Bitly, as I recall, is it said, okay, I need your authorization, we Bitly people need your authorization with Twitter in order to be able to post on your behalf. So what happened was I bounced over to Twitter, and it was like, oh, now I'm at Twitter. And sure enough, SSL connection, green bar, everything all happy. And Twitter is saying to me, hi there. Apparently you're considering giving Bitly, the Bitly service, access of some sort to your Twitter account. If you authorize this, then click here.

Now, had I not already been logged into Twitter, then I would have had to give Twitter my Twitter username and password to first authenticate with Twitter, then to say to Twitter, yes, it's my intention to give Bitly access to post on my behalf. And so I said yes, I clicked the "yes" button, I authorized this, and that bounced me back to Bitly, that was now all happy and had now the ability to do this.

So today's podcast is the protocol that makes this possible, which has had a bit of a checkered past. It's got some problems. A guy named Ryan Paul posted at the beginning of this month in Ars Technica a rather scathing attack on OAuth, talking about compromising Twitter's use of it, which is really not what he did. But there's a lot of complexity to it. It's interesting. And that's our topic for today.

Leo: It's interesting and very important. And a lot of companies are using it now, including, as you mentioned, Bitly. But also Twitter itself just recently turned off any other way of authentication with Twitter.

Steve: Precisely.

Leo: And rightly so. I mean, if you want to use a third-party program to access Twitter, giving it your Twitter credentials should feel bad, should feel wrong.

Steve: Yes. All of our listeners have certainly been well enough educated by now. And consider also the liability of that, that is, if you've given 10 different applications or other services or websites or something your username and password, now what happens if you regret, if you make a mistake with one of them? So the only recourse you have - there's no way to revoke that. The only recourse you have is to change your - go over to Twitter, for example, if this was the one that you'd been handing out, change your password, but in the process of doing that you break all of the other ones, all the other services that now have an old password. So you've got to go now through all of them and update them. So as you say, Leo, I mean, this is an important thing. This is big news for the Internet. And OAuth's promise is, which it's having trouble fulfilling, for reasons we're going to address, its promise is that it will create a common mode for doing this. I mean, Facebook has had their own protocol. FriendFeed has had one.

Leo: So Facebook Connect is not OAuth. That's something else.

Steve: Correct. And one of the problems is that this sort of started slowly. Nobody really took it seriously, so it didn't have many people involved. Then more people got involved. Well, they already had their way of doing it, so they kind of wanted to bring some of theirs along with it. And they stuck it all in there, and it ended up just being a mess. But we're going to cover that in detail, explain how this works, how they've made it secure, some of the problems that have been encountered, and so forth. But yes, it's very, very important. I mean, the idea of being able to delegate to a third party controlled access to stuff of yours that's on the 'Net. I mean, that's the future. There's just no doubt about it.

Leo: Yeah, yeah. Well, we need it to work, so I want to find out why it doesn't work. And we'll do that in just a little bit. But also, as always, we've got your security updates and...

Steve: [Laughing] And, you know, we've had a couple...

Leo: A busy time, is it?

Steve: Oh, well, we've had a couple very quiet weeks, where I've said, hey, guess what, nothing happened.

Leo: Right, right.

Steve: Well, as it turns out, a whole bunch of stuff happened, probably as we were recording last week's podcast, I mean, like, right then stuff was going on. And then a lot since. So we've got updates, and we've got news, and so forth.

Leo: Okay, Steverino. Bunch of updates, I guess.

Steve: Well, of course we are past the second Tuesday of the month, which happened a day or two ago, depending upon when you're listening to this. So this most recent Tuesday was that one. Microsoft dumped a bunch of updates on us, not surprisingly, as they typically do. Of a total of nine, four were rated critical, and five are rated just important. There's nothing too earth-shattering except that we found out that there had already been a botnet which was using one that was not previously known publicly, which surprised some security researchers.

There's a vulnerability in Microsoft's print spooler service, which could allow remote code execution, which had been publicly disclosed. And it turns out that, if you send a specially formed printer request to the file and printer sharing share, you're able to take over the machine remotely, which was the hook that it turns out a relatively well-known botnet was using, unbeknownst to anyone. There's also a vulnerability in the MPEG-4 codec, which can allow remote code execution. That one had been privately reported. There's a problem with the Unicode scripts processor, same thing, remote code execution, in the way they handle OpenType. And Microsoft Outlook also has a remote code execution problem. So that's all been patched, those four critical problems. And then there's five that Microsoft just rates as "important" as opposed to "critical," scattered through Windows and nowhere in particular. So just as usual, make sure that you're updating Windows.

I found that I only had one, actually Silverlight, which wasn't among them, was wanting to be updated, since I haven't yet moved myself to SP3. I've got to tackle that and get that done. And it's funny, I should mention how many listeners send me helpful hints about updating to SP3. It's not that my system won't. I have updated to SP3. But it did some strange things with the Start Menu, I recall. And when I backed out of it, everything got okay again. So I ought to just try it again. I'll make an image for safety's sake and then see if I can be there because I would like to stay on the update train along with all of our listeners.

Adobe's back in the doghouse once again. Two, count 'em, two new zero-day vulnerabilities. One that occurred exactly a week ago as we were recording this podcast, on September 8th, is a new problem in Reader and Acrobat. There are no fixes for this. This is a zero-day exploit that was first seen in the wild. It's in the CoolType.dll, which we've had problems with before. So another problem there. It's called the SING "uniqueName" Buffer Overflow Vulnerability. Adobe talks about it on their site. They are going to attempt to push out a fix for this one week early. The normally scheduled quarterly update for Adobe stuff was scheduled for October 12th, which was the second Tuesday in October. They do second Tuesday of the months; but, unlike Microsoft that does it every month, they only do it quarterly, so every third month. But because this is being actively exploited they're going to do any updates, including something to fix this, the week earlier. We don't have a date yet. They're just saying the week of October 4th.

And oddly enough, Microsoft has a toolkit whose name I just love. It's the Enhanced Mitigation Experience Toolkit.

Leo: Everything's an experience for Microsoft.

Steve: Who names these things?

Leo: Jimi Hendrix.

Steve: So we have - we're Microsoft, and we have an operating system that we can't get quite right. It's got bugs and flaws and vulnerabilities and exploits happening all the time. But we want to mitigate those. So we're going to come out with the Enhanced Mitigation Experience Toolkit...

Leo: Sounds like a Disneyland ride.

Steve: Doesn't it? Because we want to mitigate the pain of the experience of using the operating system. So the EMET, Enhanced Mitigation Experience Toolkit, now at v2.0, actually just recently at 2.0.0.1. It was released to fix bugs in the first 2.0 version. So I guess that would be the mitigation experience experience...

Leo: It's kind of [indiscernible], but...

Steve: ...where they fix those, yeah. And so it turns out that this exploit in the CoolType.dll, in Adobe's product, it's a so-called ROP, a Return-Oriented Programming bug. We've talked about this in the past, the idea being that, rather than attempting to execute code in the data area, which DEP, Data Execution Prevention, will block, which is now of course turned on in the later operating systems that are affected by this, instead clever programmers arrange to loop through the ends of existing code in order to achieve their goals. So it's called Return-Oriented Programming because you return into existing code, get it to do something; then it comes back to you; then you jump to somewhere else, to some other existing code, basically using code that other people wrote at known locations in order to do your dirty work.

Well, the "known locations" is the key, which is why Microsoft introduced ASLR, Address Space Layout Randomization, which causes code in Windows to be relocated, to essentially be loaded into random or semi-random locations. And that breaks Return-Oriented Programming. The problem is that the one vulnerable, actually I guess two out of many DLLs that are in Acrobat. And it's got a funny name. It's not meant for primetime. It's icucnv36.dll. That's one of two that does not have ASLR enabled. So that particular DLL is always at a fixed location. And it's the exploit that occurs in Acrobat in the CoolType.dll, it ends up being used to execute code in this icucnv36.dll, which is always loaded at the fixed location.

Well, what the Enhanced Mitigation Experience Toolkit from Microsoft allows you to do is to override this non-enabling of ASLR and force DLLs that always sort of just don't have ASLR enabled, force them to be loaded at different locations. And I got a kick out of it because Adobe links to this on Microsoft's site. Microsoft has a page where you land from Adobe saying, okay, we're going to help you out with this buffer overflow vulnerability. Now, I should tell people, or our listeners, I'm not doing this, and I'm not sure I would recommend anyone do it. First of all, the Enhanced Mitigation Experience Toolkit v2.0.0.1, brings a lot of baggage with it. You have to have .NET installed. And it just seems like a heavyweight thing to do unless you're already using it and have it installed, in order just to protect Acrobat for a few weeks until the week of October 4th when they fix this problem.

But on their page, quoting from their page, it says, "In order to enable EMET" - that's the Enhanced Mitigation Experience Toolkit - "for Adobe Reader and Acrobat you have to install EMET and run the following simple command line as an Administrator." Now, I looked at it. I thought, okay, wait a minute. Simple command line: emet_conf.exe --add "c:\program files (x86)\Adobe\Reader 9.0\Reader\acrord32.exe". It's that easy. So...

Leo: I can remember that. I've got it written down, yeah.

Steve: Yeah, you got it.

Leo: Oh, yeah.

Steve: So if you have the Enhanced Mitigation Experience Toolkit already installed, then I would say, hey, go for it. Protect yourself from this by forcing this DLL to load in different locations which will protect you until Adobe fixes this problem, sometime the week of October 4th. Otherwise, I would say, eh. It's being used in targeted attacks. So standard caution with PDFs is not to open PDFs coming in email. And I would say not open them unless it's from a very trusted source.

Now, there is an interesting new solution for the whole PDF problem, which I've run across, but it didn't seem to work for me. And I haven't figured out why yet. But there is a new Firefox add-on called gPDF. And what's clever about it is, if you click on a link when you're browsing around, Firefox uses the Google Viewer to open the PDF. So instead of opening the PDF on your system, which is always hair-raising and frightening and prone to exploitation, Firefox opens it, full-screen, in the Google Viewer. Which means it's not being rendered on your system, it's being rendered within this Google Viewer system and presented to you on your browser, which protects you from PDF exploits. But for me it didn't work. So I'm going to track that down because it seems like a useful thing to do, as soon as I figure out how to make it go. But it's called gPDF, for anybody who's interested, for Firefox.

Leo: Very good.

Steve: And that was the first of and the longest-winded zero-day vulnerability. The second one...

Leo: Oh, really, there's more than one, wow.

Steve: Yeah, two, two zero-day vulnerabilities in the span of one podcast.

Leo: Great.

Steve: Yeah. This one is with Flash Player. It's a critical vulnerability in Adobe's Flash Player 10.1.82.76 and earlier. It affects pretty much everything - Windows, Mac, Linux, Solaris - and Flash Player, which is actually a different version, 10.1.92.10, for Android. It also affects Reader. Apparently Reader can be used to invoke this under Windows, Mac, and Unix, 9.3.4; and Acrobat 9.3.4, same version and earlier, for Windows and Mac. It's being actively exploited in the wild. And they are saying we have to wait till September 27th. So we're going to get a fix sooner than that, but they're saying the week of September 27th. So that's week after next we should have a fix for that. So not been a good week for Adobe, not that they actually have a good week very often.

Leo: They haven't had a good week in a long - in months.

Steve: Yeah, it's, boy. And in fact, what I'm reading out in the security world is people just shaking their heads, I mean, really feeling like Adobe's becoming a laughingstock, security-wise. It's just, I mean, this is a problem when for a huge length of time you are writing code sort of casually, without an eye toward security, and you build this massive code base which people then, the bad guys then turn their attention to and start looking for problems with. Well, I mean, it's just riddled, clearly it's just riddled with problems.

Leo: Because these are not all related. These are all different issues.

Steve: Yeah, they're just different mistakes that Adobe makes. And they're things that work, but they only work when you give them really good code, if you give - they're rendering technology, basically, they're rendering PDFs or they're rendering Flash files. So that means that basically they have interpreters that they've written which are interpreting the PDF content and interpreting the Flash data. And as long as you give them valid PDFs and Flash files to interpret, they work fine. So Adobe says, "Ship it." And everyone's got them on their machines, and everyone's happy.

Turns out, though, that these things are - they were never written to be defensive. They were just written to work. So they're not defending themselves successfully from bad guys who say, oh, look, this thing didn't check for negative numbers, so let's give it a negative number when it only expects positive ones, and see - oh, look, that allows us to reach backwards through the buffer, rather than forwards, the way it was meant. But the code didn't check for negative because no one ever expected a negative number. But if we give it a negative number, then, look, it lets us go backwards behind the buffer, and let's figure out what we can do. Oh, look, we're able to execute code that wasn't meant to be executed if we give it a negative number. So it's that kind of thing.

And imagine, I mean, to have created a huge code base, I mean, these things are not small anymore. These are tens of megabytes of executables and DLLs because of course Adobe has just bloated this thing, their whole suite, as they've tried to add features and new bells and whistles in order to further their cause, their own commercial interests. So we end up with mistakes throughout this huge amount of code. And I'm sure Adobe obviously wishes that they didn't have any mistakes. But what they've got is so big, I don't know if they're ever going to be able to fix it. And now it's just an extravaganza of exploits.

Leo: A bonanza.

Steve: A bonanza. So Firefox had an update also, to 3.6.9. A big bunch of vulnerability fixes, about 14 were listed in the CVE, the Common Vulnerabilities and Exploits database. And Mozilla had the same number, 14, one for one, matching those. And just I'll quote briefly from the SANS Security Newsletter that had a nice summary. They wrote: "Mozilla has released patches for multiple vulnerabilities in Firefox. Some of these vulnerabilities may be exploitable for code execution. Potentially serious vulnerabilities include multiple memory corruption vulnerabilities in the browser engine, a heap-based buffer overflow in Firefox's code handling of HTML frameset elements, an invalid pointer in plug-in handling code, a buffer overflow in the code responsible for transforming text runs, a use-after-free vulnerability in the code for handling XUL tree structures, a memory corruption vulnerability in the code handling XUL tree objects, a memory corruption vulnerability in the code handling of 'nsTreeContentView,' a memory corruption vulnerability in the code used to normalize documents, and a memory corruption vulnerability in font-handling code on Macintosh systems. Note that all of these problems require an attacker to entice the target to view a malicious web page." Which actually is not much of a caveat because that's how all these things happen these days. You click on a link in mail, you go to a web page, and that's not good for you. So we want to make sure everybody's updated to 3.6.9.

Chrome had one of its standard little updates to 6.0.472.53. They're as tight-lipped as always about what was fixed. So all they said was multiple remote code execution vulnerabilities were fixed in that. And even Apple's iOS 4 had 24 problems fixed...

Leo: Wow.

Steve: ...most of which - yeah. I looked at the list over on Apple's site. It just scrolled on and on and on. Most of them were things that resided over in WebKit. So these were WebKit-located problems, which Apple fixed.

Leo: Presumably patched in the open source stuff, but Apple has to do it on their own. You know, it's funny, I have a funny reaction when I see a long list of patches. That makes me happy. It's like, well, there's some things that are fixed. I don't know, I mean, I just presume there's bugs, so - now, you know, IE9 came out today in its first public beta.

Steve: Oh, and really does look nice.

Leo: Good.

Steve: The news is that IE9 is way fast. Apparently it is really fast, economically designed, more standards-compliant. And let's just hope, I mean, we know what "new" means in security. New is never good. New is just a whole bunch of new problems.

Leo: Unless you're like Adobe, and you take that old ugly code base, and you start from scratch, maybe with some eye to security. I mean, that sounds like what they should be really doing with their code base. Start over.

Steve: Yeah. I don't know what else, I mean, I have to say again, whenever I'm assembling all of this for the podcast, I find myself just shaking my head, thinking, you know, the fundamental architecture of our systems is broken. I mean, the idea that systems can be compromised by one program's misbehavior, I mean, that's fundamentally broken. We need a system that is inherently sandboxed; a system, I mean an operating system, where no process is able to break out of a sandbox. And if that could be created correctly, then our problems would go away. I mean, it's just that we don't have that today. Someday we're going to have that. That's the only way we're going to solve this because every time something is created that's new, it's going to be right back to square zero.

I mean, I guarantee you, mark my words, Leo, we're going to be talking about IE9 bugs coming out of our ears. I mean, I'm glad there's a better IE, that it's more standards-compliant, that it's faster, that Microsoft has put an IE team back together again because they disbanded it pretty much after IE6. They said, okay, well, now we're just going to maintain this for a while.

Leo: We've finished. We wrote the perfect browser. We'll never have to fix this.

Steve: Yeah. So I'm glad for that. But it's going to be bloody. There's just no way around it. It's a bunch of new code. There'll be mistakes in it.

Leo: Sigh.

Steve: So in security news, I mentioned a couple times this botnet which was using a previously unknown security hole. Well, it turns out that the bad guys are apparently even further ahead of us than we knew. We've talked about the Stuxnet, S-t-u-x-n-e-t, the Stuxnet worm, before because that was the one which surprised us back in July with the shell remote code exploit, the .LNK. Remember the shell shortcut, .LNK, that's the one that was spreading through these SCADA systems. SCADA is the acronym for Supervisory Control And Data Acquisition, which is used unfortunately in high-automation factories and plants like nuclear reactors and in other major, like power control, power-generating factories and things. It's sort of like the underlying technology for doing factory and large plant automation. And for whatever reason, the Stuxnet worm had - it was, like, going after these SCADA systems.

And what was found was that, unfortunately, the way SCADA's architecture was set up, they had hard-coded passwords in these systems, which was known to this worm, this Stuxnet worm, which was using the hard-coded passwords for its own purposes. So what Kaspersky Labs folks recently discovered was that not only could it spread using this shell shortcut vulnerability, this .LNK vulnerability, but it was also able to spread using that previously unknown printer sharing vulnerability that Microsoft just patched two days ago, and no one knew about it. Except this worm knew about it, and it was using it.

And what's also creepy is that they found as they analyzed this worm that there was the ability for it to use a vulnerability which has been known for more than two years. It was back in 2008 there was a vulnerability. And what was interesting is that, if the worm used that in corporate networks, that is, this worm was not only - it's a Windows-based worm, but it's targeted at these SCADA systems because unfortunately Windows is the interface to these Supervisory Control And Data Acquisition systems. So Windows was sort of like the door into these SCADA systems.

But the worm was smart enough to know if it was on a corporate network that would probably sense its attempt to use this '08 exploit. And if it sensed that, it would not use it. It only used it if it could tell, through its own analysis, that it was on an older network, like one of these SCADA networks, that would probably be vulnerable to that and wouldn't be updated and able to sense that it was doing that.

So Siemens, the manufacturer of this particular brand of SCADA systems, has said that at least 14 operational plants located in the U.K., in North America, and Korea, and also by far the largest number of infections located in plants in Iran, had been infected. And Joe Weiss, who's a managing partner at Applied Control Systems, ACS in Cupertino, that's a major supplier of this kind of automated control technology, has said that, as troubling as the Stuxnet worm has been for Windows, the real target appears, as I was saying, to be these SCADA control systems which are interfaced to Windows, and that's their way in.

Symantec reported that this Stuxnet has the ability to take advantage of the programming software to also upload its own code into what are called PLCs, the Programmable Logic Controllers, in industrial control systems that are typically monitored by the SCADA systems. In addition, said Symantec, Stuxnet hides its own code blocks so that, when a programmer using an infected machine tries to view all the code blocks on this PLC, not even a Windows thing, so this is crossing out of Windows into equipment automation systems, they will not see the code injected by Stuxnet. Thus, this Stuxnet isn't just a rootkit that hides itself on Windows, as we know it does, but is the first publicly known rootkit that's able to hide injected code located on these programmable logic controllers, these PLC systems.

And finally, Joe Weiss, this guy at ACS, was quoted saying the mechanism that the Stuxnet worm uses to install the Siemens payload comes at the very end, which means - that is, the end of what the worm is doing - which means this isn't a Siemens problem and that they could have substituted GE, Rockwell, or any other manufacturer's PLCs as the target of the worm. And he says at least one aspect of what Stuxnet does is take control of the process and be able to, for example, whatever the programmer wanted - opening and closing valves in the plant, turning pumps on and off, or speeding up a motor, or slowing one down. He says, "This has potentially devastating consequences, and there needs to be a lot more attention focused on it." So it's frightening stuff, Leo.

Leo: Yeah, no kidding.

Steve: I mean, it's getting out into the real world, not just messing with people's email, but now using Windows as the entry point to get to process control systems and muck around with pulling the reactor core out of things.

Leo: You know, I think this might be a new surface, attack surface, because I'm sure these embedded systems and these specialized systems like this are not anywhere near as tested and tried and true.

Steve: No.

Leo: Maybe they are. I don't know. It just seems like...

Steve: No. From my exposure to them, they're much simpler. They're much less sophisticated. They're much simpler. And they don't have nearly, well, for example, there's no sort of antimalware, antivirus technology for them. They're very simple-minded. And so it's entirely feasible that something could load code in and just sort of link itself out of the file system very easily, and just - and then be in there running, but not seen.

Leo: Yeah, amazing. Ugh.

Steve: I did run across an interesting add-on for Firefox that does work for me, that I wanted to mention, sort of under my Errata and Tips. It's called BetterPrivacy. And I ran across it a couple weeks ago, and I've been using it for a couple weeks and can now recommend it. So it's BetterPrivacy for Firefox. What it does is deal with these supercookies that we've talked about, the LSO, the Local Shared Object cookies which, for example, Flash installs. The first time I ran it, it said, when I was shutting down Firefox, it said, oh, you have 142 supercookies installed. How would you like me to delete those? Well, it surprised me because I know I had gone over to the Adobe site, which is still Macromedia is where this thing is configured, and turned them off. But they turned themselves back on again.

Leo: Oh, yeah, because we've talked before about turning off Flash cookies.

Steve: Yes. And then just today, when I was restarting Firefox, actually to install this gPDF thing that I talked about, there were 14 more. And I thought, wait a minute, I'm sure I had these disabled. So I went back over to the Flash configuration. And just using that UI, I disabled these again. I went through the various tabs, noting that there were more of them now than there used to be. And when I went back to the first tab where I turned it off, it was already turned on again. So I'm really annoyed by this. I don't know, I haven't tracked down what's going on. But I'll just tell people, you think you've turned this off, check back the next day and see if it stays off because something is turning it back on.

And so if nothing else, this BetterPrivacy add-on for Firefox, I mean, it's not about browser cookies at all. It's just sort of a nice place for this little piece of code to live, such that when you're shutting down Firefox, it will, on the way out, it will delete these Local Shared Objects. And then what that does is that prevents your regular browser cookies from being reconstituted next time you bring up a page for some advertising site that is deliberately using the Flash cookies to reconstitute your browser cookies behind your back, which is the way this super-tracking happens.

Leo: And this LSO program will find all Flash cookies, all of these supercookies?

Steve: I know that it does it for Flash. And it sure found 142 for me.

Leo: That's a lot of them, wow.

Steve: Yeah.

Leo: So do we now change our recommendation that you turn off the cookies using Flash's own tools? You might want to find a third-party tool to check.

Steve: That's a good point. I haven't found one yet. But I'm very disappointed that just using the tool that you're directed to for managing Flash cookies...

Leo: Right. This is Adobe's own recommendation is to use that tool.

Steve: Yes. And it seems to be not working right, in a way that's not in the users' best interests.

Leo: Yeah. And you think this is a flaw in Flash or - which certainly there's none of those. Or is it, I mean, what's going on here?

Steve: I don't know because the tab, there's, like, there's now about six or seven tabs. And the second tab is the one where you say do not store anything on, don't ask me again, and disable. And I did that. And then I marched through all the tabs, one after the other, to see if there was anything else I wanted to turn off. And there were various things I deleted and turned off. And then by coincidence I just sort of went back to the second tab again, and it had turned itself back on. So it's like, oh.

Leo: I hate that when it happens.

Steve: This is not good.

Leo: That's terrible.

Steve: Yeah. Yeah.

Leo: All right. Do you have a SpinRite story before we...

Steve: Just a very quick little mention from a listener of ours, Jeff Crews, who wrote - I loved his little comment. It just started off, "SpinRite Works!" And I thought, well, yeah, Jeff, I could have told you that. He said, "I've been listening to the steady testimonials on the Security Now! podcast. I bought SpinRite 6, and a couple of months later" - I like that because he bought it just to support us and because he wanted to have it ready if he had a problem. So he said, "And a couple of months later, my home computer would not boot up XP in safe mode or any other mode. I ran SpinRite 6 on Level 2, Recovery, and in 60 minutes it had marked one sector as not fully recoverable, but repaired the rest. I rebooted my computer, and XP then booted. XP ran its special CHKDSK on its own and has been working fine ever since, for over a week now. It would have taken hours to reload all my software and reconfigure everything again. So thanks, Steve, for saving a lot of time and lost data." And thank you, Jeff, for the report.

Leo: Yay.

Steve: Yay.

Leo: All right. Now, I know a lot of people have tuned in, because I tweeted about this, and I think there's a lot of interest in this. In particular, I mean, OAuth is such an important part of the Internet these days. It's used by almost every site for authentication.

Steve: Yes.

Leo: And I think it'd be great to know how it works and also what the potential problems are.

Steve: Well, yes. My sense is they're getting smoothed out. It had a bit of a rough start. It started back in '06. And just to back up a little bit, to recap a little of what we were saying at the top of the show, the issue is that users want a means, well, actually need a means these days to allow other websites, and maybe even desktop applications, to act on their behalf. So, for example, you might have all of your photos located on some photo-sharing site, like Flickr, and you might want to use a printing service, to allow a printing service to gain access to some photos that are over there for printing them out.

So rather than you being the middleman, you just want to use direct bandwidth for one site to have access to the other. Well, in order for that to happen, the printing service might need, well, it needs authentication, it needs authorization to have access to those files. So what we would have once done would have been to, unfortunately, to give the printing service your username and password. But that makes everyone uncomfortable these days, as well it should because, as we mentioned before, if you did this a lot, then there's no means for revoking those privileges that you've given to just one of the services to whom you've given your username and password. You'd have to change your password, which then breaks all the authorization that you've given, and then you've got to go back and fix it all.

So what we needed was - oh, and for some reason this is called the "password anti-pattern." I ran across that phrase a number of times while I was pulling things together and doing the research. And it doesn't have any particular meaning for me, but password anti-pattern is the standard term used for giving your username and password for one service to a different service when you want that second service to have access to the first. So that's the problem that we're solving here. And notice also that, if you had a service where you authenticated using one-time passwords, then there's no way you can give, even if you wanted to, you could give a varying one-time password to some other service because, as soon as you use it, it's no longer valid again.

So we needed some way to solve this problem. So back in '06, about four years ago, sort of a loosely confederated group began to try to figure out how to make this work. And a perfect example, Leo, of third parties that want access, I looked at my own Twitter account for SGgrc.

Leo: Oh, I bet you have - if you're like me, you have dozens of them.

Steve: Yes. I had Seesmic Desktop; Bitly; Seesmic Web; Twitlonger; something called ManageFlitter that I didn't even know, I was like, what's that, that I'd given permission to at some point; Twitterrific; the Visitor Widget, which again it's like, huh, and that was part of TwitterCounter that I did go look at a couple weeks ago; TweetDeck; Twittelator; Twitpic; and Twitter for Blackberry.

Leo: You really did get into Twitter [laughing].

Steve: Yeah. So those are all things that - those are other third-party agents of various sorts that I've given, that I've permitted to access my Twitter account. Okay. So how does this work? The user's experience is pretty nice from a standpoint of using a web browser. You're at a site that wants access to some resources elsewhere on the 'Net. So it bounces you - what the user sees with their web browser is they get bounced over to the site that you're wanting to provide permission for. You agree to do that, and you're taken back. I mean, so it's seamless and nice from a user's experience. So let's talk about from a security protocol, from a security standpoint, how that works.

We have to name the endpoints. First of all, the client is the service that wants access. So, for example, the client would be the photo printing site in the examples that I've used that wants access to resources on the server. So the server would be like Flickr, or would be, for example, Twitter, that has the services or resources or whatever which we're wanting to provide the client service access to. And then we have the user, who has the credentials, who is the one who controls the resources on the server side.

So the server makes it known that they support OAuth, that is, they're an OAuth provider, meaning that, if the client supports OAuth, then it's possible for there to be this exchange of credentials and authentication. Now, prior to this, the reason OAuth is good is that, before there was a standard that people could agree upon, there was still this problem, that is, this need to - in fact formally you would call it "secure delegated access to protected resources." Secure delegated access to protected resources. That's sort of the problem we're trying to solve is we want to delegate to a third party some access, controlled access, to resources that we don't want everyone to get to. They're protected in a way that gives us, for example, the ability to revoke that access at some later time, and also to control it.

I saw in some of the pages that talk about this, they talk about the valet key that some higher end cars have, where the main car key is able to do everything with the car, but the valet key, you're able to start the engine, but for example not unlock the glove compartment and the trunk. And in fact, I didn't know this, but apparently there's like a limited distance you can drive with a valet key?

Leo: I didn't know that either. I have one. That's interesting. That makes sense, yeah, how far should he have to drive? You'd hate for him to conk out a couple miles away.

Steve: It's scary, the idea that the car might just decide, well, okay, you've driven me too far on my valet key. I mean, I don't know what it does. But anyway, I did run across that. I thought, well, I didn't realize there was that limitation. But anyway, so a valet key is sort of an analogous way of saying that this is something you could do limited things with. So, for example, in the Bitly case, where I was using Bitly to shorten links and post on my behalf through my Twitter account, well, that's all it can do. So it's able to do that, but it's not able to do all the other things that I can do as the fully credentialed, fully authenticated owner of this account.

So a service provider, the so-called server, it says it's going to support OAuth. Oh, what I was saying was that, prior to OAuth, we still had these needs. But there wasn't a single standard for doing it. So people, you know, Facebook had some need for automated access, so they just invented a random protocol that was theirs. And various sites would create their own. And the problem was, from a developer's standpoint, there were no libraries for doing this. It required constant code to be written all the time. And the client sites, especially if they wanted to offer their services to multiple providers, they'd have to individually support what other, I mean, like the arbitrary, invented-there protocol that each different service was offering in order to allow some sort of - this delegated access to their protected resources.

So the dream and the goal of OAuth was let's all agree on a single standard, for all the reasons that doing so makes sense. Just we're going to solve this problem once. We're going to all bear down and focus on it and solve it right, solve it in a way that provides a solution for everyone's needs, and then we're done. And we're solving something that's really important for the 'Net. So the service provider says, okay, we're going to support OAuth. It also needs to make available to developers the so-called endpoint URLs, that is, the URLs, the web links where its OAuth API is accessible.

There's been, there's some talk in the current 2.0 spec, the OAuth 2.0 spec, of an OAuth discoverability where discovering when a service supports OAuth and discovering what these URLs are would be automatic. But apparently that doesn't exist yet. This is all finally now in the hands of the IETF. So it sort of, again, it's had a spotty past. There was a 1.0, and that had actually some security flaws in it. And then there was a 1.0a. And then, when it wasn't really being well adopted, there were people who were complaining that the spec wasn't really clear, that they were having problems getting their OAuth to interface with somebody else's. There were some ambiguities that kept it from working, error codes that were sent back, didn't really explain what the service had a problem with. It just said, oh, it didn't work.

And so then there were people saying, well, we don't want to try to sign these requests because signing is hard. So we're just - we're going to rely on SSL to provide the security wrapper for the protocol. And then people who actually understood security said, no, no, no, you can't do that. So there's been just sort of this evolution of this protocol going back and forth with people tugging it in different directions. There was something called OAuth Wrap, W-r-a-p, for a while, which ended up being what OAuth 2 was based on. So it's over the last four years this thing has had a bunch of ups and downs.

But fundamentally the protocol works as follows. I'll use the Bitly example that I talked about at the beginning. I'm over at Bitly. Now, Bitly, a client of Twitter, has to have identified itself before, that is, sort of like when it's going to set itself up as an OAuth client of Twitter. The Bitly owners, the owners of the Bitly site, send email to the Twitter folks and say, hey, we want to be a client of OAuth, so send us some credentials.

So the Twitter folks just sort of make up what looks like a really good password. It's just a bunch of random characters and numbers, two of them, actually. There's one that's used to identify the client. And then there's a secret token which is never put on the wire; it's never sent out. It's used as a signing key which that client uses to sign its communications with the server to prove that it is who it says it is, so that the server is able to verify the signature. It's basically just a digital signature, as we've talked about many times in the past.

So ahead of time, a site like Bitly, or anyone that wants to be a client, has to have had a relationship, has to have established a relationship to get these permanent credentials which identify it statically, permanently, to the server. So I'm over at Bitly, and Bitly says, hey, I'd like to act on your behalf to be able to post compressed-link tweets to your Twitter account. So if that's good for you, we need to authorize me to do that, says Bitly. So I say yes, I say, that's what I want to do.

So Bitly redirects my browser to the site that is going to be serving this content, that is, the site that knows me, that I have an authenticated relationship with. Part of that redirection is in that redirecting URL, essentially. It can be headers in the HTTP request for this authentication page over on Twitter, in this example. It can be strung on the end of the URL, or it can be in the body of the request, the way posted data is sent to the server.

One way or another, a bunch of information is provided by Bitly, this client, over to the server. Among them are, for example, this very random-looking, sort of pseudorandom-looking string of numbers and letters, which is this client identifier, which identifies to Twitter who it is that is requesting authentication, in this case Bitly. And then a timestamp and a number of other sort of like one-time things. There's a so-called nonce, a one-time usage string. And also the URL, which the Bitly URL, which Bitly wants the user to be sent back to once they have authenticated. So the user comes to Twitter with their browser and all this additional information which provides a digitally signed request from Bitly for - essentially it's asking for authentication.

Now, behind the scenes, Bitly has also had a communication with the server getting temporary credentials. Basically it says I want to open a session. I have a user here at my site, Bitly, that wants to authenticate, so I'm going to send them to you in a minute. Give me a temporary credential, just sort of for this transaction. So those are also sent. So Bitly's credential, along with this temporary credential, are sent over with the user to Twitter.

So Twitter has everything it needs in order to understand what it is that is being asked of it. For example, it knows about Bitly because they have established a relationship before. It sees this temporary credential that it issued probably moments before. It's like, okay, yup, here's this guy coming back. We've been expecting him. It knows who the user is, assuming that they're still logged into Twitter. If not, the user would need to authenticate with Twitter in order to say, yes, this is really me who wants to provide this permission.

And so if all of those criteria are met - oh, and Twitter will also say, hey, we know about Bitly. Here's their logo. Here's the level of authentication we want to provide. And it could be whatever's been negotiated. It could be full access to Twitter so that it's able to pull and read and change settings. It might just be, like, posting-only access. It can be whatever level of granularity makes sense for that service to be issued by Twitter. The user agrees and says, yes, this is what I want to do.

So using the URL that, for example in this case, Bitly provided, Twitter sends the user and their browser back to Bitly with an additional token which is the authentication. So that goes back to Bitly, who then receives that as the data in the returning request that the user's browser has made to Bitly, in order to go back there. Bitly has now the temporary credential that was first issued. It also has the authorized credential which the user has just received from Twitter, having authenticated with Twitter and agreed and brought back to Bitly.

So Bitly then makes one final contact to Twitter, saying here's everything I've collected. Here's the temporary credential; here's the authentication which I've just received back from the user. Please now issue me permanent formal credentials on the user's behalf that I'm going to use from now on. And so there's a web server to web server interchange. Finally, behind the scenes, Bitly gets that credential and then stores it with the account that it's established with the user, which then in the future permits it to make the agreed-upon requests on the user's behalf.

Leo: I got it. I followed every step of that. Every step of the way. No, no. But it's a token system.

Steve: Yes, yes, it is a...

Leo: Exchanging tokens instead of passwords.

Steve: Yes. You're basically - you're exchanging tokens. And the client ends up with the credential that then allows it to make queries, API queries, to the Twitter API on the user's behalf. The thing that's nice about this is that, much as I did when I went over and looked at my connections, I think it's called "connections" under Twitter, here are all the different applications which I have at one time or another done that with. I have permitted them to access Twitter on my behalf.

And exactly as you said earlier, Leo, what Twitter did recently, which is really what brought this to my attention, is they shut down their traditional username and password base, so-called "basic authentication," and they said - I mean, and they've had the OAuth there for, like, more than a year. So there's been plenty of time for applications to switch over and start using it. And they said no more username and password. We're tightening things down. We're going to do it the right way from now on.

Now, here's the problem, is what I've just described really works well. But there are a couple of reasons it works well. One is that, when I'm dealing with one website that wants access to another, the websites are talking to each other as needed behind the scenes. So, for example, I never - there's no way for me to have access to the client's secret key which it uses to sign the conversation, these packets of information going back and forth. The beauty of that is that that's its secret key which absolutely verifies and validates its identity to the server. All I'm doing with my browser, as we bounce from one site to the other, is we're just shuttling, we're just sort of carrying the information back and forth that has been signed by these two parties. And the beauty is the user is sort of the shuttling mechanism from one site to the other, providing the authentication and then bringing it back. So that aspect of it is very elegant.

Okay, so here's the problem. And this came up, this was the main thrust of this Ars Technica attack, which is really what it amounts to, by this Ryan Paul. He's a FOSS developer, Free and Open Source Software developer who has a Twitter client. And this OAuth technology, it begins to fail when you move it from the scenario I just painted to websites where instead you want applications on the desktop. Because the lesson all of our Security Now! listeners know very well is that nothing on the desktop can be protected. That is, it's very much like we've talked about how you fundamentally cannot keep DVDs encrypted because the DVD player in the end user's living room has to be able to decrypt them. In fact, Leo, you may have picked up on the news that HDMI, the master HDMI key...

Leo: HDCP, yeah.

Steve: Yeah, HDCP has leaked out onto the Internet. It hasn't been confirmed yet. But, if so, it means that Blu-ray encryption is in bad shape right now. So the problem is we know that it's impossible for applications to keep secrets. Well, it's even more impossible for open source applications to keep secrets. So here's the problem, is something like Seesmic or Twitterrific or TweetDeck, any of these things that are running on the desktop, what we're saying is that for them to use OAuth means that they have to have their token, and they have to digitally sign their transactions with this signature, with their shared secret key. Except that it's got to be in the application. It's got to be there on the desktop.

So even though we're wanting to allow this third-party application access on our behalf, and although it seems like a very similar thing, the fact that it's running on the desktop fundamentally breaks this model. And so the problem is, Twitter has formally said, Twitter has said that, if some application starts spamming Twitter accounts behind their users' backs, that application's OAuth credentials will be suspended.

Leo: Isn't that beautiful.

Steve: Well, it's a nice idea.

Leo: Yeah, because it's just you can - you have a kill switch.

Steve: You have a kill switch. The problem is these applications can be reverse-engineered.

Leo: Oh.

Steve: Yeah. And in fact what Twitter has said is, well, developers, even of open source software, need to go to any reasonable measures to hide the so-called consumer key and consumer secret, that's this client key and client secret, from exposure. Well, this Ryan Paul, who writes for Ars Technica, was miffed at this whole thing because the problem would be he's got a Twitter client, and he's an open source guy. This is a Linux client. He's an open source guy. And he receives this key and secret from Twitter.

Well, in the first place, it's fundamentally antithetical to his ethic of an open source guy to have anything that needs to be kept secret. But it's open source, so arguably it can't be. So out of curiosity, he took Twitter's own Android client, which Twitter produces for the Android platform, just did a string search through the executable and found "Consumer key: 3nVuSoBZnx6U4v," and then he put in his article, "XXXXXX."

Leo: He didn't want to publish it.

Steve: Right. And similarly, the consumer secret, same thing, here's the secret shared key which Android's app uses in order to sign its work. And again he left the last six digits as X's because he didn't want to expose it. On the other hand, everyone now knows that this thing is available by doing a string search.

Leo: In the clear.

Steve: In the clear. So here's the dilemma. Obviously Twitter's Android client is authorized to do things on behalf of the user. So all a bad guy has to do is create an evil client that looks the same as the Android client, but sends spam behind people's backs and uses the same key and the same secret, and Twitter cannot tell the difference.

Leo: Oh, that's not good.

Steve: No. And so as soon as...

Leo: So there's no certificate authority or anything like that. There's no way of verifying...

Steve: Well, here's the problem, is there's a fundamental breakage in the model when we go from a web-based client to a desktop client. And that's where this thing falls down.

Leo: I see. So there are certificates on the web that prevent this.

Steve: Sure, yeah. Well, and in the web model, remember, if I'm, for example, using Bitly, I have no access to its secret.

Leo: Oh, yeah, yeah, yeah, that's right, Bitly has it, but not me. Yeah, yeah, yeah, I get it.

Steve: Right. It's on its server, and it uses it to sign things.

Leo: I get it. Right.

Steve: But if I have a desktop client that must use the secret to sign something, I can watch it do so. I can debug it. I can reverse-engineer it. I can go, oh, look, it's signing what it's about to send. There's the signing key.

Leo: Maybe it would be better if these clients went to the web to get the key, you know, and did a triangular thing. Or something.

Steve: I had exactly the same thought. That's not in the model at this point, unfortunately. And so just to finish this thought, once the evil client starts using the Twitter Android client key and shared secret, Twitter has no choice but to disavow that client. It has to disable that client, which breaks every Android client which has been downloaded and forces all the users to update their client.

So, I mean, it's a fundamentally broken thing because, unfortunately, this is not a problem we can solve. I mean, there isn't - we have to think through, like, adding another leg to this. Like if the client went to a web server to have it do something, I still don't see how an evil client couldn't do exactly the same thing. It's the problem of having complete access to the client running on the desktop. So...

Leo: It's like the analog hole. It's just a flaw with clients; whether it's DVD players or Twitter clients, it's just a flaw. You have to have the key.

Steve: Yes. You have to have the key. And if you can have it, then everybody else can, too.

Leo: And even if you encrypt the key in the code, I mean, obviously this is really bad because it's a plaintext searchable string. But even if you encrypted it, you have to decrypt it and store it in memory at some point.

Steve: Yes, exactly. You have to access it in order to use it. And in the act of accessing it, we know how clever hackers are. They're just going to chew into this. I mean, now that it's become a big issue. This was something that hit the news about 10 days ago is where I first saw this posted in Ars Technica. And I thought, okay. We've got to talk about this.

So the good news is, for web-based solutions, where the client is not on your desktop, but is like Bitly or is like any of these web-based systems, OAuth is a beautiful solution. It's at v2.0. Now we're beginning to see libraries which allow programmers using Python and Ruby and so forth, standard toolkits, there are libraries that do all the heavy lifting, perform the crypto, do the work that's necessary. And so this kind of delegation among online services makes sense and works. The problem is, I mean, maybe it's better than nothing on the desktop. I mean, I don't want to give these clients my username and password. I mean, that's just a bad idea. So the fact that we sort of have a solution is better than none.

I would argue, I guess, that there ought to be no attempt to identify a client by using a key and a secret, which is fundamentally vulnerable, as it is if it's on the desktop. We ought to just say, look, that's not something that OAuth is able to do, even though it's trying to. And that's sort of where it falls down. If we don't ask it to do that, if we don't make assumptions that it can, then Twitter has a problem because there isn't a means for Twitter to shut something down that goes awry. On the other hand, there's no good way for it to do that. It's just - it's them wanting to do something they really can't do, unfortunately.

Leo: Yeah. Gosh, I hope this isn't fatally wounded. I mean, we really need it to work. But wouldn't any system be similarly hobbled?

Steve: Yes. Yeah, I mean, it...

Leo: That's the nature of it.

Steve: Yes. There is no way to protect this on the desktop. The only way to protect it is in web-based systems where there is a piece that you cannot get to. You cannot get to the server. All you're seeing is what the server wants to show you. So that provides the integrity that OAuth was originally designed to provide. It's when they extended it into this desktop model that it's like, wow, this is a bad idea.

Leo: Yeah. Very interesting.

Steve: So we do have a protocol, OAuth, which works beautifully, and very seamlessly. I mean, I loved, when I got bounced over to Twitter, I thought, wow, this is the way it should be because I'm not telling Bitly anything about me. I'm saying to Twitter, this is me, and I want you to allow those guys to do the following things. And I've got a list of those things that I've given permission to, and underneath each one it says "Revoke," "Revoke," "Revoke." So at any time I want to, I can click on those and remove them from the permissions list, and they're no longer able to act on my behalf. This works. When it's web-based, it works. Unfortunately, it's got problems when it's desktop-based.

Leo: Almost seems like we should have made a different way of doing it for clients and just made it required that it's the web only. Then we'd have at least a working web system.

Steve: Well, and one thing you could do would be that users themselves could say, okay, I'm not going to use a desktop client. I'm going to use web-based clients because I know that they're going to be robust against these kinds of threats.

Leo: Right. Really great stuff, as always. I'm glad you tackled this one. Makes me wonder if I - now, the new Twitter page, you don't really need a desktop client anymore. So maybe I'll just use the new Twitter page and avoid that.

Steve: Yeah.

Leo: Steve Gibson is at GRC.com. That's the place you can get SpinRite, his hard drive recovery and maintenance utility. It's fantastic. Also lots of free stuff at GRC.com, including ShieldsUP! and all of his free security programs. And this show: 64KB version for those with lots of bandwidth; 16KB version for those without. Transcriptions for those who like to read as well as listen; show notes, too. We also have all of that at TWiT.tv/sn. And our TWiT wiki now contains all the show notes as we go along. I've been putting them in there, which is great. And we have some nice people, volunteers who go in and clean it up because I just dump it in there. They format it all out: wiki.twit.tv. We do this show live at live.twit.tv, every Wednesday at 2:00 p.m. Eastern time, 11:00 a.m. Pacific, 1800 UTC, live.twit.tv, so you can follow along and be in the chatroom while we do it live. It's always great.

What else? I guess next week a Q&A, so if you've got questions for Steve or thoughts about this or any other topic in security, go to GRC.com/feedback, pose the question, and we'll use about 10 of them next week. Steve, have a great week.

Steve: Thanks, Leo. It's great talking to you. Talk to you next week.

Leo: See you next time on Security Now!.