GRC's eCommerce System

Description: Steve and Leo delve into some of the non-obvious problems encountered during the creation of a robust and secure eCommerce system. Steve explains the hurdles he faced, the things that initially tripped him up, and the solutions he found when he was creating GRC's custom eCommerce system.

High quality (64 kbps) mp3 audio file URL: http://media.GRC.com/sn/SN-109.mp3
Quarter size (16 kbps) mp3 audio file URL: http://media.GRC.com/sn/sn-109-lq.mp3

INTRO: Netcasts you love, from people you trust. This is TWiT.

Leo Laporte: Bandwidth for Security Now! is provided by AOL Radio at AOL.com/podcasting.

This is Security Now! with Steve Gibson, Episode 109 for September 13, 2007: Steve's eCommerce System.

It's time for Security Now!, everybody's favorite security podcast, the number one tech podcast in the nation.

Steve Gibson: [Trumpeting] Thanks to our listeners, yeah.

Leo: Part of the Podcast Awards. You're going to go down and get your award in a couple of weeks.

Steve: I absolutely am looking forward to it.

Leo: I will be in Vancouver, so I can't come congratulate you. But it's ironic. You'll be in Ontario, California. I'm going to be in Vancouver, Canada.

Steve: That's true.

Leo: But congratulations. I think that's just really wonderful. Really great news.

Steve: Well, again, we owe it to our listeners because I didn't even know the competition was happening. And then we pushed out that quick little quickie short podcast saying, oh my god, you know, everybody please vote for us, I would love to win this. And our listeners made it happen, so.

Leo: Well, you know, I make a point of not, you know, most podcasts are always lobbying to get, you know, voted up in Podcast Alley and stuff. And I - we did it once just to prove we could do it, and then I made a point of not mentioning it ever again because, you know, we know you love us. We don't need to win any popularity contests.

Steve: I want a whole shelf full of trophies, Leo.

Leo: It's nice to have the awards, though. That's different, you see. That's - I think that's nice. And, you know, I've got my little shelf. Not a big shelf, but I've got my little shelf, and I'm satisfied. So now it's your turn, Steve.

Steve: Okay. Well, I'll happily do that. And our listeners, if they keep supporting us, then I'll be, you know, super delighted, so.

Leo: So now it's time to talk about security and all its many guises. Today we're - actually this is going to be kind of fun. I'm looking forward to this. We're going to talk about something Steve did. Steve is a developer. I mean, that's really what you are first and foremost. You're famous for GRC, not only for your commercial product, SpinRite, but for all the little free programs you write.

Steve: Oh, Leo, when I can arrange my life so that I can just go hide in a corner and code, I am never happier than then. You know, it's just...

Leo: I'm sympathetic. I feel the same way, really.

Steve: Oh, it's just - I love coding. And, yeah, so...

Leo: I just never do it, so I'm so rusty it's not worth the energy anymore. It takes me a week just to figure out what I forgot.

Steve: Oh, yeah.

Leo: You've got to do it all the time. Do you code every day?

Steve: No. I wish. I mean, I really, really wish. But there are, like, days I'm - at the moment I'm working on implementing a round of sort of internal updates to our eCommerce system. Sue, my operations gal, had said hey, you know, it'd be nice to be able to search on email domain names, not just email addresses. And, you know, I had just - I thought, oh, okay. And she'd asked for that a while ago because, for example, people at IBM want to know if, like, who else at IBM has purchased copies of SpinRite.

Leo: Oh, interesting, yeah.

Steve: And she had no way to do that because I'd written everything, you know, to do specific types of searches that we thought would be useful. Anyway, so I'm doing a bunch of those. And I thought, you know, in going back into the code, I saw some things that I had sort of forgotten that I had done that I really liked a lot. And also someone once asked us, and I had referred to this in an earlier podcast, like, you know, what's the thing I'm most proud about in my eCommerce system, and what's the, you know, were there any sort of anecdotal mistakes I made. So I thought that would really be a fun thing to talk about since I'm all sort of back into it at the moment.

Leo: Cool.

Steve: But we've got a bunch of feedback stuff I want to cover first.

Leo: Let's talk about addenda, errata, and anything else you want.

Steve: Miscellaneous flotsam and jetsam.

Leo: Flotsam and jetsam.

Steve: Okay. First of all, one of the more popular things on our Security Now! page is that feedback form down at the bottom of the page. I think it was Episode 101, so about two months ago I commented to our listeners that, if it were easier for them to just send the email to the same email account that the web form sends it to, that they could, and that it was securitynow@grc.com. Well, about 24 hours later we began getting spam on that.

Leo: Really. Well, that's because spammers listen to the show, I guess.

Steve: Well, okay. I actually think spam comes from a couple sources. For one thing, Elaine, our fantastic transcriptionist, dutifully transcribed the email address, which of course went onto the web in four different fashions, you know, HTML, TXT, and PDF. And so maybe it was scraped off of there. But the other thing we've seen through the years is that we'll get spam, for example, on our corporate accounts, presumably because people get some infections on their machines, and these trojans will suck up all the email addresses in their email client and then start sending spam to those. So it may have been that securitynow@grc.com ended up being in people's email outboxes and got picked up that way. Anyway, one way or another, yeah, one way or another it was a disaster.

Leo: But a feedback form wouldn't be in their email address, so that's something else.

Steve: Right, right, no, it was the fact that I told people the direct address.

Leo: Oh, you gave them an address, okay.

Steve: Exactly. Anyway, so I've had to shut that down. Anybody who does try to send email there will get a polite bounce message saying, well, we're sorry, but the bots found this account, so we've had to close it. What I did, what I've decided to do, though, because that Security Now! page has gotten quite long, is I decided to move that feedback form to its own page. So I just wanted to let people know that there's a new page. It's actually only linked at the moment from the bottom of the Security Now! page in case people are used to going, digging all the way down to the bottom, scrolling down to the bottom of that long page. And I've got to do something about that. That's part of the project I'm on now in sort of fixing a whole bunch of things that have been loose ends on the GRC site is I'm going to reorganize our Security Now! page because it's just, you know, right now it's a huge, long page of 108 individual podcast notes. So I'm going to do something to fix that. But GRC.com/feedback. I know you like the nice, short URLs, Leo.

Leo: I'm not the only one.

Steve: So in your honor, GRC.com/feedback is a feedback page which replaces the form that used to be at the bottom of the Security Now! page. And I did something this time different, and that is I added a subject line to the form. It used to be that there was just sort of a big box where people could write what they wanted to send, and then optionally provide their name, their location, and email address so that we were able to say, hey, you know, Bob in New Brunswick has written to us, has a question and so forth, to allow people to put that stuff in if they wanted to. But in the email that that form sent me, I had just a sort of a default Security Now! feedback subject line. Now the user can put that in. And oh, I tell you, I mean, I've been getting the feedback, and it's so nice to sort of just be able to scan the subject lines and see what people are writing about. So it's an improved feedback form in addition to being a little more bot-obfuscational.

So I experimented actually with putting reCAPTCHA on there. And the reCAPTCHA API turns out to be very easy to use if you've got scripting enabled. But it sort of fights you if you don't have scripting enabled. And I didn't want to require our users to have scripting enabled. So I ended up just sort of abandoning reCAPTCHA for the time being. And it doesn't seem to be a problem with form spam here anyway, so...

Leo: No. As you and I both know, CAPTCHA doesn't [indiscernible]. So it's just a nice - I still use it just because I want people to help write books, type in books. But it's not, you know, it's not the best way to fight spam. I like what you've done.

Steve: Yeah, I think it's going to be good. So we'll see how it goes.

Leo: You've cleverly crafted this so that - because I get a lot of form spam, comment spam. But if you cleverly craft your form, you don't get that.

Steve: Well, yes. And it's worth mentioning to people what I did. I don't think that we're going to have any listeners who are going to be spamming me. But, for example, in the prior form I labeled with text, I used text to label the fields. And I thought, well, that's a problem. So now I'm using images to label the fields. I've reorganized them, and there are some dummies in there, also. So, for example, it's going to be much less easy, much less automated for something to come along and fill out the form and have it send email to me. So anyway, so I did a little, I mean, I didn't want to go overboard. But it's a little, you know, it's a little bot-hostile without making a big deal about it and without being at all user hostile, which I'm really happy about.

Leo: That's great, yeah. No, I think that's neat. In fact, at some point it'd be great to - did you make this stuff up, or was there somewhere you went that documented some ideas about this?

Steve: Oh, Leo, it's assembly code. I just wrote...

Leo: But you thought about the ideas of moving forms around and fields around and obfuscating and so forth.

Steve: Yeah.

Leo: You should document that at some point. You don't have to give away your particular secrets, but I think it'd be a good idea to help others do this.

Steve: Yeah. And it is nice to have something that doesn't run the users through the hassle of CAPTCHA, yet still is moderately resistant. On the other hand, I really do think that this is a function of, as we talked about before, it's a function of the value of cracking the form. Certainly this is completely crackable, if somebody really wanted to go to the trouble of doing it.

Leo: Yeah, but they don't do that, right?

Steve: Exactly. Because it's not like I'm giving away Yahoo! email accounts. That's just, you know, send me your comments sort of thing. So it's a relatively low-value thing for someone to try to crack.

Leo: Right, right.

Steve: In talking about defragmentation, we talked - it may have been a comment last week, in last week's podcast, where a listener asked what was the free defragging tool that I had mentioned. I said, well, the one I had mentioned wasn't free. I had mentioned Vopt at v8, which I really like. Wait, I think it's at v8.2. But the one I'm using now is PerfectDisk.

Well, it turns out that, first of all, I got two things. Someone calling himself BotSing posted in the GRC newsgroups that there was something that Mark, our old friend Mark Russinovich, had done at Sysinternals, who of course has now been purchased by Microsoft, so it's over on the Microsoft site. He's got something, a piece of freeware, called PageDefrag. So if you just put PageDefrag into Google, the first thing that comes up is Microsoft's instance of that. And it does, for free, does a defrag of the swap file, your registry hive files, event log files, hibernation files. And basically those things which are in use at the time that you're defragging which would not otherwise be defraggable. So if you were to use that and any other defragger, for example Vopt, then you sort of get the best of both worlds, even though PerfectDisk, which is really my current favorite, does that as well, and does a bunch of other things also. Then we got a note...

Leo: So that doesn't - it doesn't do your whole drive. It does the key things, though.

Steve: It just does those main system files which are normally not movable while the system is in use because they are in use. And so Mark came up with a way to make that happen.

Leo: That's free. Microsoft offers that for free. If you do a search for Microsoft and Sysinternals, you'll find that there's, like, a lot of programs there.

Steve: Oh, yeah. Those are all of the things that Mark has done over...

Leo: Great stuff, yeah.

Steve: ...the years.

Leo: So I presume it's there as part of that package.

Steve: Yup, it is. And then we had a listener named Donn Edwards, who - I guess he's a longtime defrag fanatic, is the way I - I don't think he would mind me describing him that way. He is also a listener. And he made a blog posting whose title was Mr. Hard Drive - meaning me - "Mr. Hard Drive Uses PerfectDisk." Which turns out to have been his favorite defragger after extensive defragger analysis. I'm mentioning this, and we will have a link in our episode notes, because he's got extensive reviews of 20 commercial defraggers and 20 freeware defraggers.

Leo: Wow. I didn't know there were that many.

Steve: It's unbelievable. And a little bit of the history here, I sort of browsed around trying to understand, sort of to get an overview so I could explain this to our listeners. He also was a PerfectDisk follower, that was, with all of his reviews, I mean, like, seriously, 40 defraggers, both commercial and freeware, as I understand it, PerfectDisk was his chosen defragger. Then he heard me mention Vopt and took a look at Vopt. And he decided, wow, I like that, too, because Vopt is faster, whereas PerfectDisk is more thorough. So his solution...

Leo: He uses both.

Steve: ...is to use, exactly, is to use Vopt, like, more frequently to keep things under control, and then PerfectDisk occasionally to do major deep cleaning sort of defragging.

Leo: Before we get too much farther, do you really think this is necessary?

Steve: No. I just...

Leo: Okay. I've been saying for years defragging is overrated. There's some people who need it if you're, you know, if you're digitizing video and you have big, large files and you need a lot of space, okay, that have to be read and write very quickly, okay. Maybe every six months, every quarter you can optimize. But people do this daily.

Steve: Well, I've got to say, Leo, with what Microsoft is doing now with their continual patching, they're patching system files all the time that would normally not be moving. And so it really does create a much greater level of fragmentation on your drive than in the old days, where you installed the OS, and it just sat there happily. And of course Microsoft's OSes are getting bigger and bigger and bigger. I don't know. For me it feels good to have it defragged. You know, there have been benchmarks that sort of come out either way, like, yes, it speeds things up; or no, it really doesn't; or you're having to do it, you know, they tell you defragging is more wear and tear than just letting the frags sit where they may.

So I don't know. I just like it. I like the idea of pulling things together. What I normally do is I will do a defrag before I do an image because I like the idea of sort of creating a nice, compact image, even though the images are obviously smart enough to handle fragmentation. It's like, well, why do I want to make an image of a fragmented drive? I'd rather make an image of a nicely, all set to go, kind of cleaned out drive. That's also an opportunity to delete, you know, this huge amount of cookies that tend to accumulate, to delete all the junk out of your temp directories, and just sort of do a little housekeeping before you do a - and then you do a defrag, then you do an image. So...

Leo: Okay.

Steve: That's my approach.

Leo: Yeah.

Steve: And then, lastly, I found out that our podcast has a lot of reach relative to all the talking we've been doing about multifactor authentication. It turns out that PayPal felt a huge event from our talking about my original discovery of the PayPal Security Key. And similarly VeriSign. Apparently we just drove them right off the charts when we...

Leo: Sorry. Sorry.

Steve: No, I mean, they're really happy about it. But, I mean, we came to their attention so much so that they're playing snippets of the podcast mentions of VeriSign in their upper-level management meetings.

Leo: Really.

Steve: Yeah.

Leo: Wow.

Steve: They're interested in somehow using, sort of leveraging the podcast to sort of extend the reach of their VIP Security Key tag thing. And they're going to sort of put together a deal where, if our listeners will go to their page and sort of fill out a questionnaire, in return they'll make the VeriSign fobs available at a reduced price.

Leo: Oh, that's great Now, all they have to do is buy a little advertising, and we could put it network-wide. I guess I should call them.

Steve: Yeah. Or next time I talk to them, I mean, I want to keep our and my enthusiasm for...

Leo: Well, it wouldn't run in yours anyway. You're sold out. So it would have to run in other podcasts anyway. But...

Steve: Okay, yeah. I just wanted - I want to make - I want to keep us, you know, clear in terms of, like, what's being paid for and what's not. I just think...

Leo: You know I do, too, yeah.

Steve: Yes, yeah. Anyway, what I found out was...

Leo: So they purchase no advertising; they're getting this plug absolutely free. Aren't I happy.

Steve: So it turns out that some people who've been writing in have been unhappy that the VeriSign fob is only available to domestic purchasers. So I asked the gal who I've been talking to over at VeriSign what's the deal with that. And she said, well, it just - it's only temporary. It's just that their current fulfillment system only runs in dollars, dollar-denominated transactions, and they just haven't put together everything they need in order to sell these things globally, although the VIP system is running globally. And, I mean, their infrastructure is ready to go global.

She was down doing some meetings in Brazil. And what she told me was sort of interesting. She said that it was much - they quickly achieved a lot of traction in Brazil because the consumer protection laws that we have in the states and that we sort of - I actually sort of take them for granted. For example, we know that if your credit card gets compromised, you have a very low liability. I think it's $50, and normally even that is waived. That is, if you challenge charges on your credit card, and you validly charge them and can show that these were not your purchases, they're just taken off your credit card. Not so in Brazil. In Brazil you are liable for those charges on your credit card. So you can imagine that consumers are substantially more leery of eCommerce fraud and really want all the additional protection that they can get.

Leo: Right, right.

Steve: So anyway, good things are happening with VeriSign. Obviously they're set up to be an OpenID provider. I had asked the question, is there any way that a smaller guy can also take advantage of that VIP system. And so that's what we're going to pursue. And just to sort of - I think it would just be fun to be able to, like, have a page where people could play with their tokens on GRC. So I'm going to see if I can do that. And also they want to - they're working to move forward with other avenues like PayPal. In fact, it turns out that your favorite company, Bank of America, has just added, and just announced a couple days ago, that they will be offering multifactor authentication, no longer just your favorite SiteKey technology, Leo.

Leo: Yeah, which drives me up the wall, yeah.

Steve: I know it does. But and they've chosen a different form factor. They're not going to be using a fob. Apparently you can also do some sort of a credit card form factor, where it's got a little LCD in a window, and a button that you press in a credit card form factor. I guess they think it's nicer that it kind of goes in your wallet and looks like a credit card.

Leo: Right. And apparently will also send you this number via cell phone, which is what's most interesting to me. They said the card is delayed till next month. I just got an email.

Steve: Oh, okay, cool.

Leo: That's going to be very interesting, yeah. I'm glad. Thank you. I mean, that's really an improvement.

Steve: That's going to be really nice. And again, that's - the whole cell phone thing is another one of VeriSign's means for doing authentication. They are planning to come up with something even better, which is a software implementation, a pure software implementation of essentially the crypto that's in the fob, so that it could be installed on cell phones and just be built in at zero cost. So, I mean, this - I'm just excited about this notion because I think all of this provides a tremendous amount of leverage for the need for secure authentication, which is obviously not going away. I really believe it's just going to be something we need more and more in the future.

Leo: Very cool, yeah. As soon as I get mine, I'll give you a report on it.

Steve: Cool.

Leo: Yeah, yeah. Do you have any mail that you want to - I guess we're kind of responding to mail right now, aren't we.

Steve: Yeah, and I do have a fun SpinRite success story. Actually the subject was "SpinRite Success Story With a Linux Twist." I try to find testimonials that are a little bit interesting and different from sort of the, well, SpinRite did it again sort of thing. This is Matt actually from New Brunswick. I think that's why I used New Brunswick earlier as an example, it was in my head from finding this one. Matt from New Brunswick, Canada dropped us an email. And he said - and the subject was, you know, "...With a Linux Twist." He said, "I've been a long time Security Now! listener and have heard the numerous reports of SpinRite success from your other listeners. I recently ran into uncorrectable error hard drive problems on my Linux machine." He says, "I was more annoyed at the expense of replacing drives than worried about data loss. Every night at 3:00 a.m. everything's backed up to an external drive, and I've got full monthly backups on DVD-Rs." He says, parens, "(I'm the paranoid type)." And he says, "Naturally I turned to SpinRite to fix the errors, knowing that it supports Linux partitions and correctly assuming that it would fix the problems. The twist is that I found out the SpinRite executable works perfectly under the Win32 Wine emulation program."

Leo: Oh, that's interesting. Did you know that?

Steve: I think I remembered that during our beta testing, but it's not something I had ever mentioned before.

Leo: Wow.

Steve: And so he said, "I thought I'd share this in case other Linux users are also running into hard drive problems and are on the fence over buying a non-native Linux program."

Leo: So you run it in Linux under Wine.

Steve: Exactly. So when you run SpinRite, the same SpinRite EXE that you normally boot and run, when you run it, it sees that it's being run under Windows or in a Windows clone environment. It pops up its little graphical dialogue, and that allows you then to tell it to burn or to create for you an ISO image. Then you use your regular Linux DVD or CD, I'm sorry, CD-R burning software to burn the ISO onto a CD, and that gives you a bootable CD that allows you to then run SpinRite on Linux, needing no Windows anywhere.

Leo: So it's not for running SpinRite, it's for running the installer so that you can burn the CD, then you can run it.

Steve: Right. Because SpinRite does not run under any OS. It brings a little copy of FreeDOS along for its own environment. But you don't need Windows in order to run SpinRite. And that was the point that Matt here was making was hey, you know, I ran it completely under native Linux, under Wine, and it allowed me to make my bootable CD. Or for that matter you can make a bootable floppy, too, which is a lot easier for people who still have floppy drives on their machine. And I do because I can't - I don't know, not having a floppy drive just really always makes me uncomfortable.

Leo: Right. Although you're going to be out of luck soon.

Steve: I know.

Leo: I don't think anybody - in fact, I wonder how many machines are sold with floppy drives at all.

Steve: Yup, it's certainly endangered at this point.

Leo: Yeah. Okey-dokey, let's talk about your eCommerce system.

Steve: Well, it's funny. When I was getting going, many people said, wait a minute, you're writing an eCommerce system.

Leo: You know there's a thousand prepackaged eCommerce systems out there.

Steve: Yup.

Leo: Plenty.

Steve: And so first and foremost was, yes. But, you know, they're not mine. I mean, I wanted to write one just because I never had before.

Leo: And I should say it's not just because you don't trust the other ones. You may not. But it's also, as you just said, you enjoy programming. And I think you enjoy learning about the problems that need to be solved to do this. It's not, well, you reinventing the wheel.

Steve: Well, okay. To be fair, there was some of that. But we were, you know, being involved in security, I was seeing constant reports of flaws being found in the so-called "shopping cart" systems. So my question was, how can I trust, I mean, how can I trust and expect my customers to trust some software that I did not write?

The other thing is there are third-party services like Digital River. And unfortunately Digital River are just horrible people. I've had bad experiences with them. They pursued me for years to let them sell SpinRite. They want to take a huge percentage of the retail sales, like 30 percent they take. They also - I've had to buy other software through them, and then I start getting spammed by them because I have to give them my email address when I use them to buy somebody else's software.

Leo: So you're protecting your users, too, really.

Steve: Well, exactly. Exactly. I don't want my - there's no way that I want to use a third-party to do any of my, you know, to capture my data or have anything to do with order fulfillment because how can I trust what that third party will do? And I wouldn't be at all misunderstanding of users who said, gee, Steve, we wish you didn't have to bounce us over to this horrible third-party site in order to buy SpinRite. That just doesn't feel like the right thing. And then, finally, I wanted it to work my way. That is, I really dislike this thing that so many eCommerce sites make you do, which is the, quote, "create an account" with them. You know, it's like, wait a minute. I'm buying a widget of some random type. I will never be back here again, I promise you. Why do I have to create...

Leo: I know, I hate that.

Steve: Agh.

Leo: I just want to buy this. Even if I, you know, maybe I am going to be back. I don't want to create an account.

Steve: Yes, that whole notion of, like, needing to go fill out all this paperwork and

questionnaire on the web to create an account. It's like, ugh. So there are sites that are - that give you the choice. You can create an account, and they'll, like, hold onto some of your stuff to make it easier if you come back later. Or they'll allow you to, like, bypass the create-an-account phase. And I'm always so glad when I'm given that option. Well, I thought, my system isn't going to have any of that nonsense. You provide only the information that we need in order to process the purchase transaction and nothing more. So that's the way I designed it. Because I was writing it myself, I was able to do this exactly the way I wanted to. So that was the final reason really for doing this. And the problem has been also that people have liked my system and have wanted to buy it. And...

Leo: That's interesting.

Steve: ...it's not for sale. I mean, I didn't...

Leo: Then you know you're going to get a spate of requests after this comes out.

Steve: Well, yeah. That's why I'm saying upfront here that it's not for sale. Because it does some cool things that I have never seen anyone else do. And I'm going to explain it all and, you know, give them to the world. So anybody else who wants to write their own eCommerce system to do this is welcome to. But mine's not for sale because, frankly, I can't charge enough for it to make it worth my while to package it for sale. I mean, this is not the business I'm in. I'm in the business of doing these podcasts and developing software, you know, like for individual end-users. That's really my focus, not for other companies that want to sell stuff.

So anyway, I did, in thinking about how I wanted my system to work, there were a couple very cool things that I came up with. For example, my system doesn't use or require cookies, and doesn't use or require scripting, and has no messy URLs. That is, as you're using it, the URL is just sort of a generic, I mean, it looks sort of per-user, but I think it's an eight-character little bit of junk at the end of the normal URL, which actually is just sort of the ID for the page.

The question then is, how am I saving state? And let's step back a little bit from before we answer this to remember what the model is for so-called Web 2.0, that is, the idea that browsers are no longer just receiving information from people. You're no longer just putting in a URL which is the address of a page, which the browser then sends you, and you read that, and maybe you click a link and go to some other page. Now there's much more interaction between the user and the remote server, much more like an application where you're doing things.

But inherent in the browser, sort of the client-server model is the notion that there's a transaction, meaning that you're looking at a page, you do something, maybe you fill out some information and then you submit it. Well, that all goes back to the server. But due to the heritage of the way the web works, even when you're sending information back to the server, it's actually in the form of a query because the web was built on browsers querying servers. That's all they were able to do, really. So when I fill out a form and submit it, it's actually sent as a query, and I'm waiting for a response. That's why, you know, users will be so used to seeing in IE the little globe spinning, or the flag waving, or whatever. Or in new IE, the little sort of blue disk spinning deal. Or, you know, whatever mechanism the browser has for saying I sent the query on to the Internet, and we're now waiting for the response.

The model is that a connection is created to the server, the query is sent at the beginning of that connection, and then the connection stays open while the browser is waiting for the response, which is the next page. So there are several ways that, in this model, in this query/response model, that people have come up with for maintaining state, meaning - and by "state" I mean that there's, like, a memory of what you have done before. Now, the classic

means is cookies. And it's because of the need for maintaining state that Netscape originated the notion of a cookie, the idea being that the server would send back to the browser literally a cookie, which is - well, I don't mean literally. I guess that's, you know...

Leo: Here's a chocolate chip cookie.

Steve: Chocolate chip or, yeah, or molasses, sugar and molasses or something. A cookie in this case is just sort of a chunk of text. It could be English readable, but most often it's some sort of a token that has meaning only to the server. And that's the point, and the reason it's called a cookie, is it doesn't have to have any meaning to the user. It's not visible normally to the user. The idea is that the browser will accept it from the server. And then any subsequent queries, that is, any subsequent connections and submissions of a query for a next page that the browser gives to the server will automatically include just an echoing back of that cookie.

So the idea is, in an eCommerce model, you go to some page, and you fill out a form. And then so you've got the form filled out. You then press the Submit button. And the form content is sent back to the server as a query. So what the server's going to do is it's going to look at the form data. And maybe, you know, often you'll receive it back in response to submitting it, sort of as a, okay, here's what we got from you, check out your information to make sure that it's correct before you commit to buying this.

So now you've got that information. Well, say that you say yes. Okay, if you say yes, then the question is, yes to what? Because you no longer have a form. You just have a button saying yes. So the server needs to know who among all the people maybe purchasing things at the same time is saying yes. So what that requires is that the browser have cookies enabled so that you're able to - when you say yes, the cookie goes back. The server then looks up what user is associated with that cookie and gets the data in some sort of a temporary database of, like, purchase transactions that are someway along, and then processes it further.

So the point is that you need to have cookies enabled in order for this to work. Well, as we know, some people just don't like cookies at all. They've got them turned off. So rather than that breaking eCommerce, and actually sometimes it will, then scripting has been used in order to provide this same sort of functionality, where you've got some sort of scripting, you know, code running on the page which is involved in providing cookie-like behavior, even though you're not actually using the cookie mechanism in the browser. But as we also know, scripting is sometimes turned off. I keep recommending that people not use scripting, although, again, it can be the case that turning scripting off will break things.

So one of the other mechanisms that is used for basically tracking users is what's called "URL munging," where - and this is what we'll often see. For example, if you go to Amazon, bang, you know, you go to www.amazon.com, what you see almost immediately is some bizarro URL. I mean, it's got all this junk in it. And so what's happening is, basically they're saving state in the URL. That is, all the links on the pages will be relative to that wacky URL so that, when you do something, you're sending back a different URL, even if you're on that page, than somebody else who's looking at the same page. Because Amazon, after logging you in, is basically giving everybody custom pages for everything they're doing.

Well, I looked at all those mechanisms. And I said, you know, I don't like any of them. I don't want to require my users have cookies enabled because I discourage cookies, although actually I think first-party cookies are fine, that is, cookies that you're sending back to the browser that you're visiting, not to some third-party advertising site. Third-party cookies I see no valid purpose for other than tracking. First-party cookies are okay. But if people have them turned off, I don't want that to break my eCommerce system. I clearly don't want to require people to have scripting because here I am saying to everybody, oh, turn off scripting, it's evil, it's bad, except when you want to buy SpinRite, then you have to turn scripting on. It's like, no, that won't fly either. And I really just sort of dislike the whole URL munging problem because you

can end up with cached pages in your browser that end up having URLs in it, or you might mark the page and give the link to somebody else, and so they're basically capturing something that was like your cookie in your URL. So that's a problem, too.

So, and the overarching problem is this notion of the server having to know something about incomplete transactions, meaning when you start to purchase, the way most systems work is they'll create sort of a record or a database entry or something for this not-yet-purchased event while the user moves through the web forms and provides the information that confirms their email address, however many stages this purchasing transaction is. The user is interacting with the server over time. And the problem is, what if someone goes a couple stages in and changes their mind? And they hit their home page, and they disappear.

Well, over on the server, the server's waiting for them to proceed. It's waiting for them to press the next button, move to the next stage in the purchasing transaction, give them back the cookie or the URL or whatever it is that the server is using in order to deal with it. So that requires some sort of temporary storage at the server end. And it requires that you do something about it at some point in the future, that is, you expire it somehow. You say, well, this got to be an hour old, and we don't think this user's ever going to finish with their sale. If they happen by some bizarre chance in the next, you know, at some point in the future to press the button, then we're going to say, we're sorry, we don't know who you are anymore because we expired your purchase. The point is it's just messy to sort of have to have this data sitting there in an incomplete state and then make a determination about when you're going to have it expire. And most people think, well, okay, how else could you possibly do this? How could you not require that the server have any storage about this potential purchase as it's moving forward? Well, I solved the problem.

> **Leo:** Of course you did.

Steve: My system has no storage on the server side. Instead, what I do I've never seen anyone do before. But it's worked out really well, and I think it's really neat. And that is that we'll start, for example, at the first form page. The user fills out their information and submits it to the server. So that goes to the server. The server, because I'm requiring no scripting, the server runs through the data that they've submitted, makes sure that their email address looks like an email address, makes a quick check of their credit card number to make sure that it passes the Mod 9 credit card number check, which we've talked about a long time because it turns out that there's a simple formula you can run on a credit card number that will catch digit transpositions, and basically that's pretty much all it's useful for. But it does, if you just had a typo in the credit card number, it would catch that. Basically it's sort of a first pass to make sure that the fields we need to be filled in are filled in and that the user has, you know, is on their way, as far as we can tell, to a successful transaction.

Okay. Upon submitting that data - remember that that's a query that goes to the server. And the browser's little globe is spinning, or whatever that it's doing, saying, okay, we're waiting to hear back from the server in response to our query. Well, in my case what I ask for is - I want to show them everything they submitted and ask them again for their email address. Because it's frankly surprising, Leo, I'm sure you've seen all these systems that ask your email address twice. And it's annoying. It turns out people really don't give you the same email address twice. It's unbelievable how necessary it is to ask for it a second time. And I didn't want to put it on the same page because I know what I do, I type it wrong once, then I cut, copy, and paste. I mark and copy it and then paste it into the second field because I don't want to have to type it again. But if it's wrong twice, then it's really wrong. So I wanted to put it over on the second page.

So I show the user everything that they filled in in the forms, sort of in a confirmation. And I say, you know, please provide your email address again to confirm. But that's all I'm asking for is the email address. So what I do, which really worked out well, is I take essentially a binary

image of all their data, of their purchase data. And I encrypt it using a secret key on the server. Then I digitally sign it using a cryptographic hash, as we've talked about before, tacking that on the end. Then I turn it into ASCII so it's all just characters and numbers. And I send that back to the web browser as a hidden field in the next form the user is going to fill out.

So all I'm asking for on the second page is their email address. And that would be one field. But then there's a hidden field which is this ASCII blob which is completely secure because it's encrypted, and it's digitally signed. Basically what I'm doing is I'm handing back to the user all of the data that would normally be kept temporarily on the server. And I'm keeping nothing on the server. I have no record at all that this user has ever done anything because there's...

Leo: Do you think that's typical? Is that how other commerce systems work?

Steve: No, I've never seen this done.

Leo: They're all keeping it on their own site.

Steve: Well, yes. If you turn off scripting, and you turn off cookies, basically they'll just break. They will tell you you have to have cookies enabled in order to buy something from this site.

Leo: That's right, yes.

Steve: And I'm sure you've probably seen that before. Because the point is they're relying on cookies, and they're having to maintain the user data in some temporary form at their end. Well, I just don't like that because, first of all, there's a more clever way to do this, and that is to basically hand back the current state to the user's browser and make it be responsible for sending it back to you as you move to the next step in the process.

Anyway, so that's basically what I do. And it's worked perfectly. When they give me their email address, and they press yes, I want to - I'm confirming my purchase, here's my, you know, gosh darn it, here is my email address again. I'm typing it again for you correctly. Then what I do is I receive their email address and this blob of data. I verify the digital signature. Then I decrypt it back into their purchase record. So I have not had to store it in the meantime. Meaning if they unplug their computer, they change their mind, they wander off somewhere, then that's fine because I'm not needing to hold that, wondering what happened to them. I don't care at all. If they do go to the next step, then they're providing me with all of the data for their purchase as that next step. And I just pick it up from there.

Now, I do have some things like a timestamp in what I call the "envelope," which is this binary thing. And so just for the sake of not purchasing two hours later, because that seems like an unlikely thing to have happen, I will expire the envelope. If I receive it back, for example, three hours later, it's like, eh, for the sake of making sure we're doing the right thing, I return them a screen saying, hey, this is now three hours old. This may not be what you intend. Let's start again. Sorry for making you do so.

Leo: But you'll show them what they ordered three hours ago, or...

Steve: Yes, exactly.

Leo: See, I like that. It gives you the choice.

Steve: Yes, exactly.

Leo: Because a lot of sites just throw it out. Or they keep it for days, you know.

Steve: Yeah. And what's very cool then is that, from my standpoint, basically I'm using the client-server model, and I'm using the browser to save the state of the purchase as we move forward, never needing to store anything at all on the server. And just it really has worked out well.

Now, one of the other things that was a bit of a runaround was I'm sure you've seen on many eCommerce sites these warnings on the final purchase confirmation button about only pressing this once and then waiting, or you may be double-charged. I know I see that all the time. Well, you know, this was my system. I didn't want to have to have that.

Leo: It happened to me the other day with Apple's system. I bought two copies of something.

Steve: Yup. Now, the first thing I did was I said okay, I'm not wanting to admonish my purchasers to only click once. So I'm going to take responsibility if they double order. And so from the beginning I had a - I have an intercept page which comes up and basically warns them if it looks to me like they're about to purchase something that they may not be intending to. So, for example - and I grabbed this off the screen earlier so I could share it with our listeners. If you, for example, were to get back on your browser a couple times, and then you deliberately tried to purchase the same thing again, you get a screen from me that says "Your card has already been charged," is the title. "Our records indicate that this credit card was recently used to successfully purchase one or more of the same products from us, less than..." and then I fill in, in this case it says two hours and four minutes ago. "While we certainly do not wish to discourage the purchase of our products, we absolutely do want to eliminate any inadvertent purchases and unintentional charges to your account. If some glitch in our system prevented you from receiving web page confirmation of your previous order, please do not proceed. Instead, check your email for your purchase confirmation receipt and software download instructions. If you are deliberately purchasing additional copies of our software, we thank you very much for your honest patronage and support of our work. Please click the Confirm Similar Order button below to proceed with the processing of this additional order. If this reorder is a mistake, please click the Do Not Repeat Order button below, then choose among our other site locations."

So I deliberately intercepted anybody where I was seeing another purchase come in after, you know, within I think it's two days. I think within 48 hours I decide, you know, let's make sure they're really intending to do this. Okay. Even with that in place, every so often we would get a double order. And it was, I mean, for the first year it just drove me nuts. Sue, my operations gal, would say hey, Steve, we double-charged somebody. And that's like, how can that be happening? I mean, before I'm accepting a second order I check the database to see whether that credit card has been charged. But every so often it was happening.

Well, I figured out what was going on finally and ended up fixing it. And as far as I know, nobody else again on earth has ever done this. Because it is really hard. It's really tricky. It turns out that some people double-click a button that you don't have to double-click. Just, I mean, a lot of us get into...

Leo: My mom does that. She thinks everything on the computer must be double-clicked.

Steve: I was just going to say, yes, you often see...

Leo: And that's not her fault because it's not clear. Some things are double-click; some things are single-click.

Which should it be?

Steve: Exactly. And it's very much the case that you'll see sort of, like, neophyte users double-clicking on things. If I'm watching someone use a computer, and like they double-click the Start button, I think, ugh, no. But again, it's not clear at all which it should be. So what was happening was people were double-clicking the final purchase confirmation button.

Now, here's what happens when you do that. The first time you click, because this isn't a button that requires double-clicking, we send the final purchase confirmation transaction to the server. The server sees, okay, we're ready to go, we're actually going to perform this purchase. Now, the server takes the information and sets up a communications connection with the credit card verification company. It happens that I'm using VeriSign. I wasn't always using VeriSign. I was using something called Payflow Pro, and I can't quite remember the company that used to own that. But Payflow Pro is like the back-end provider. I have a merchant relationship with them. And so they're the people who actually perform the credit card processing with the electronic funds transfer system in the U.S. And so the server, while it's got the connection open to the user, waiting to confirm their purchase, it turns around and opens a connection to our transaction provider and begins processing the credit card.

Now, what would normally happen is we would get a response, yea or nay, from that back-end provider. And then the GRC server turns around and either says we had a problem processing your card or, congratulations, you just bought yourself a copy of SpinRite. But if the user clicks again prior to that page coming back to them, what browsers do is drop the current connection and send the query off again. So they abandoned that open connection they had on which they were waiting for a reply, and they just started up again. So what would happen would be, this purchase confirmation comes in, and we go, oh, okay, this guy wants to purchase. Well, what do I do? I check the database to see whether we charged that card. But we haven't yet because we're still waiting for the first response to come back from the credit clearinghouse to see whether this card can be charged or not. So even though I'm looking to see whether there's already a charge, because this double-click happened so quickly and the back-end credit processing is relatively slow, I would look at the database, not see a charge, and launch a second purchase.

Leo: Okay.

Steve: And this is what everybody does. So the first one would clear. The second one would come back. And I would respond to the customer that they had purchased SpinRite successfully, when in fact a double charge snuck in. And that's why all these sites say make sure you only click this button once.

Leo: It's intractable. It's a tough problem to solve.

Steve: Well, and I did solve it.

Leo: Oh. I guess it's not so intractable.

Steve: What I do is, I have what I call "in-flight orders." That is, I basically - I separated the user-side link from the back end. So what happens is I'm tracking the orders which are in-flight, meaning from our server to the back-end credit processing server. And I'm tracking any connections from the client. So what I ended up - the way I ended up solving the problem is you say that user clicks first of a double-click, and the order is launched. I then log this in a server data structure saying, okay, we have an in-flight order to the credit processor. Then the user clicks a second time. Well, now I can see that we've got an in-flight order already, that is, I'm now looking for anything pending.

The problem is, and here was the tricky part, is that in the second click I dropped the first connection, the one that initiated the order, and now I'm waiting, I have a new connection to the server, and that's the connection that needs to receive the response. So what's funky about this is that the query that initiated the order to our server is different from the one that needs to get the response. And so by separating the relationship between the user's browser and our server, and the set of in-flight orders that may be underway, once I get confirmation back from the credit processing, I then have a list of potential queries. A user could, for example, click 10 times, bang bang bang bang bang bang bang, and that would set up 10 connections to my server. Because I'm now checking for duplicate in-flight orders, only one order goes back to VeriSign for credit card approval. When the response comes back, I don't know for sure which connections are still alive because the browser will only have the most recent one. So what I do is, I attempt to give all of them the good news or bad news, knowing that all but one will fail. But that's okay because I want to get the answer back to the user. And the whole system works, and we've never had another double charge in the last several years.

Leo: That's very clever.

Steve: Yeah. It ended up really working well.

Leo: You're doing this all in assembly, right?

Steve: Yup. Although, you know, just because that's who I am.

Leo: It's your language.

Steve: You could do this in any other language that gave you the kind of control you want.

Leo: And it gives you an EXE which IIS can run.

Steve: Yes. Well, in fact, that's one of the cool things that I did. The other thing that I wanted was anytime you're using some third-party processing, you need to somehow separate your program download from, like, the registration, typing in a code kind of thing. I mean, virtually every time I buy software it's, you know, go download the file, and then we're going to email you your passcode or your registration information or whatever. Because those things are always separated because you're just not - you're not downloading the code that is, like, already set for the user.

Well, that's one of the things I wanted to achieve with my own eCommerce system, and I did, meaning that when you successfully purchase SpinRite, you are given a download link. And that link is absolutely unique, and it's already customized for you. When you click on it, the version of SpinRite you receive knows your name, has your serial number and other data already built into it. So it is yours, and it's essentially sort of prelicensed for you, which I just really like. There is no secondary rigmarole about receiving your registration code and needing to fill in some form. That's all done.

Now, one thing did catch us right off the bat which I never saw coming, and that is, I wanted to give people a link which would be good forever, that is, I want it in their email receipt, which we do send them, here is a link you can use to download SpinRite. And the idea was there would be a funky-looking cryptographic token embedded in the link that would forever allow them to download SpinRite. And actually we ended up with a system that allows that. That's one of the other things I wanted was, for example, if somebody was traveling, they didn't have their copy of SpinRite with them, but they got into trouble, they're like, hey, I need SpinRite. Well, if they bought it, I wanted them to be able to have access to it. And so our system will allow anyone who has ever purchased SpinRite, you know, five years after they've purchased it. There's no expiration. I mean, it's like we have this relationship with our customer that we want to preserve.

However, something messed me up in the beginning, and that is it turns out there are parallel downloaders, you know, so-called "download accelerators" that not only open multiple connections, which I have no problem with, they also spy on their users and send the URL of anything the user downloads back to the mothership. And the idea is they build - they're trying to say, oh, this is value-added because you can go to our website and see the most popular things being downloaded. What was happening was this link, which was meant to be private and never, ever shared, was being published behind the user's back by their parallel downloader, by their downloader accelerator. And so a couple very early copies of SpinRite escaped onto the Internet. And, now, we know - we knew which users had this happen because their names were in the program. So anybody else who clicked the link got their copy of SpinRite. Which was not a good thing.

Leo: Something you want to avoid, yeah.

Steve: So what happened is we quickly contacted those people. We invalidated their licenses, which I already had the facility to do built into...

Leo: See, you were smart. You built this all in ahead of time. The person's name was in there, and their license was there.

Steve: Right. Right. Although of course they weren't happy that their copy had gotten loose. But we said, well, it's not your fault, it's your download accelerator is spying on you and basically reporting to this third party every single thing they downloaded. I mean, basically their privacy was being violated by this. So anyway, what I did was I realized, okay, I can't ever allow a download link to be used more than once. That is to say...

Leo: Ah, of course, yes.

Steve: ...if they're going to be spied on, then they click it, and it downloads. But in the act of clicking it, it has to expire it right then. The only way to avoid this. So I quickly did a rev of our eCommerce system. And now the links it issues, they are - literally, you can click it once, and it's dead. Now, you're able to ask for as many of them as you want. That is, if you want to, you

can just refresh the page, and it gives you another one, and then you can click it once. But nothing you have ever clicked can ever be clicked again. That is, no URL for downloading SpinRite can ever be used twice. And I also expire them. The actual funky-looking URL that I embed in there has a timestamp in it. So it self-expires, I think it's after, like, 15, maybe 30 seconds. I mean, I went a little overboard. But after this experience with our customers being spied on by their own download accelerator, I thought, well, better safe than sorry. And that was all right at the point that SpinRite 6 was released. Oh, no, I guess it was SpinRite 5 because we were initially selling SpinRite 5 through the new system. So it was, like, three years ago, and it completely solved this problem, and we never had that happen again.

Leo: Interesting. Interesting.

Steve: And then my final little quick story of the thing I messed up on that was, well, I thought it was humorous, but a couple customers didn't. I had this generic hacker lockout, that is, I thought, you know, no matter what anything else is going on, I want my system to be a little - because it's an eCommerce system we want to take it seriously. I want it to be intolerant of people screwing around with it, you know, like filling out the form, guessing credit card numbers, you know, finding a stolen credit card number and trying to guess the street address, you know, that kind of thing. And I recognized that valid users might need - might trip themselves a little bit. It turns out that people often use the wrong address. They'll use their home address rather than the billing address, if the home address different than the billing address and so forth. And since I'm not physically shipping any product, I'm only electronically delivering, I don't do that whole billing address/shipping address thing. All I want I the billing address because that's what the credit card companies are locking to as part of their, it's called AVS, Address Verification System, in order to reduce credit card fraud.

So what I decided was I would have some fixed number of transactions with the eCommerce system, that is, those being clicking a button when you've got a form filled out. For people who were upgrading from an earlier version of SpinRite, they receive an intermediate form where they provide the serial number of their down rev copy of SpinRite, which we then log in order to qualify them for the discounted upgrade price. Anyway, so that number, we've moved it around a bit. But I think it's at 12 right now. So 12 interactions with our system, and then I simply lock them out. I give them a page and I say - and this was sort of a stern page because I don't expect anybody to hit this by mistake. It's like, you know, your conduct is incompatible with this eCommerce system's policies or something.

Leo: They get spanked.

Steve: Exactly. I do a little bit of a swat. And so I figure 12, I mean, you have to really try to hit that by mistake. Okay, but get a load of this, Leo. I forgot to reset the counter when they successfully purchased SpinRite.

Leo: Oh, that's not good. So you're spanking people for buying too many copies.

Steve: No, no. Here's what happened, was they would try to fill out their information. Maybe they use their home address first, then they'd use their billing address, then there was no more - there wasn't enough headroom on that card, so they would need to use a different card. So they'd go back, and they'd start again with a different card. And then anyway, so here they're as exhausted as we are at this point. And on their final confirmation, they press the button, and it's like, yay, you've successfully purchased SpinRite [trumpeting]. Here's your download link.

Leo: Whoops.

Steve: So we've charged their card, everybody is celebrating, and we're exhausted, and now we won't let them have it.

Leo: Oh, that's cold.

Steve: Because, in fact, not only that, when they try to download it, I insult them by saying, you're a hacker, you know, you're abusing the system. And now that we've got your $89, you can't have SpinRite. So anyway, that happened to a couple people. And I thought, oh...

Leo: You heard about it very quickly, I'm sure.

Steve: I heard about it very quickly. And it's like, Gibson, I mean, again, it was trivial to fix. I fixed it immediately, of course.

Leo: This is what happens. It's hard to test a system like this. In...

Steve: Yeah. You have to put it into service, let everybody pound on it, and you quickly learn some lessons. And I did.

Leo: I want to say in vitro as opposed to in vivo.

Steve: Exactly.

Leo: Yeah, right.

Steve: Exactly.

Leo: Hey, you've got very nice users, and I'm sure they were very tolerant.

Steve: Yeah. We'd say, oops, sorry about that. And I quickly went in, I mean, it took me two seconds to change the code so that, upon successfully purchasing a product, I removed them from my hacker tracking queue, and then they're able to actually download the product that they just purchased.

Leo: Cool. There you go.

Steve: And anyway, I haven't been back in the eCommerce system for several years because, knock on wood, after getting those first little glitches out, it's just been performing flawlessly for us. And it was really - I'm really glad I wrote it. And again, it's one of the nice things about

having my own is that it acts exactly the way I want it to. I'm not having to beg some third party to add a feature or to fix something, nor are we having to go through a third party to, like, deal with somebody who doesn't recognize what this charge is on their card. I mean, it's just us, and we're a company that takes responsibility for making sure that everyone we've got and come in contact to ends up being happy. So...

Leo: That's great. That's great, yeah.

Steve: I thought our listeners would get a kick out of a little bit of behind-the-scenes of a real-world eCommerce system.

Leo: And now it's working flawlessly. When's the last time you had a bug you had to go in there and fix?

Steve: It's about, as far as I know, well, actually I never have. It's like SpinRite that hasn't had a byte change ever since SpinRite 6 was launched two and a half years ago. I haven't - I ought to tell people, the system's name is Bam Bam. I've been avoiding saying that all during this podcast, even though that's how I refer to it all the time.

Leo: Bam Bam.

Steve: Bam Bam. The reason is that our original management system that was written by a gifted programmer, I didn't write it, it was written in FoxPro for DOS.

Leo: Wow.

Steve: 20 years ago. For some reason we called it Dino. And I don't know why. But Dino Dinosaur, for some reason. And so when I was going to be, finally after - this would have been, like, after 16 years, because I think it's about four years ago that I wrote Bam Bam, I needed a name for the new system. Well, I didn't want it to be Fred or Wilma. Those weren't very catchy.

Leo: Pebbles isn't so good.

Steve: Pebbles is not that good. But I thought, hey, wait, what about Pebbles' boyfriend Bam Bam? I just sort of liked Bam Bam. So anyway, that's the name of the new system, the replacement for Dino, which believe me, four years ago we were still running FoxPro for DOS in order to manage our stuff. And so it really was a dinosaur by contemporary measures. And so anyway, yeah, I've not changed a byte in Bam Bam for, like, four years, ever since those first little glitches got taken care of.

Leo: That's truly amazing.

Steve: Because I love to code. And I love getting it right.

Leo: What fun. What a fascinating story. I think this is great. Someday I would hope, when you're old and gray and you no longer care, you might open source this, at least let people look at the source code. No.

Steve: Yeah. I think...

Leo: He's not going to do that. Doesn't want to get into that, I can tell.

Steve: Or maybe make it available on some terms, I don't know. I mean, the problem is, everybody else will want it to do their own different thing.

Leo: Well, you just say, I'm not going to support it. I'm not going to do anything to it. This is for your educational purposes only. You know.

Steve: You're right. I can just give it away.

Leo: Yeah. Or just leave out something so they can't use it.

Steve: Yeah. Although - yeah, you're right. No, I wouldn't do that. I mean, I guess they - I don't know.

Leo: It's not like you're selling it. And I know that you're not embarrassed, as I would be, at how bad your code is or how bad your commenting. Do you comment?

Steve: No, actually - oh, I comment like crazy. I mean, it's funny, too, because over the years I've learned how. It used to be that I would write comments that I was sort of writing because I felt like I should. But two years later I'd come back, and I'd look at it, and I'd go, okay, what? What am I trying to say here? I mean, this doesn't tell me anything. And so I really - I've been much better later in life in recognizing that I'm not going to understand what this is two years from now.

It's like, I don't know if you're this way, Leo. But when I need to take something out of the house, I now put it in front of the front door. I make it so I'm going to fall over this thing. Otherwise I just won't remember. I just - I know that I won't remember to do something unless I prevent myself from being able to forget. Because I'll just be in a different mindset. And it's about mindset. And so several years from now I will forget the mindset that I had at the time I was writing this.

And in fact I've had this experience just recently because, as I was saying, I'm back in adding a bunch of features that Sue has asked for in Bam Bam for quite a while. I'm finally making a lot of her little dreams come true, extra stuff that she's figured out that it would be nice to have Bam Bam do. And so I've had to - now you can see why I've been avoiding calling it Bam Bam for the last hour.

Leo: You have to say Bam Bam.

Steve: Anyway, so I've had to go back in, it's like, okay, what was I doing here? Why was I doing this? It's like, okay. So, I mean, yes. Comments are absolutely crucial. Although I have to say my assembly code looks so different from anybody else's. In fact, I think I will stick - because I've been talking about this, I'm going to put a screenshot of a chunk of this eCommerce system on the episode notes page.

Leo: Oh, I'd love that. I'd love to see that.

Steve: So people can see what it looks like. Because I look at other people's assembly language, and it's this absolutely opaque-looking stream of op codes running down the left-hand edge of the page. And, I mean, nobody could figure out what this is. Mine actually looks much more like high-level language. I use long variable names, long subroutine names. I mean, it's really, I think, readable. And anybody who wants to see what readable assembly language looks like, take a look at our episode notes, and I'll put a screenshot of the way I edit. And I'm still in a DOS box. I'm using Brief as my editor, in a DOS box...

Leo: Brief, wow.

Steve: ...because I use WordStar keystrokes, believe it or not, for...

Leo: You could program Briefs to have anything, to do anything.

Steve: Yes. Super macro-programmable.

Leo: Now, what assembler do you use? You use MSM?

Steve: I use MASM, yeah. I use Microsoft's assembler.

Leo: And do you have - have you written a lot of macros so that you can just code stuff that you do over and over again in a few words?

Steve: Not really that much because it's one of the ways that I'm so different from Mark Thompson, for example. Mark Thompson describes himself as lazy. And you will never, as long as I live, hear me use that word to describe myself. I just cringe when I...

Leo: Well, he uses libraries a lot. In fact, really, writing code for him is assembling libraries together.

Steve: And that's my point is he's got this thing called Emucore, which is this amazing proprietary library of stuff that he's written over the years. And so all he does now basically is call into his library of functions in order to do stuff. But I'm just different. I love to code, so much so that I just - I don't mind writing the same thing over and over and over because every little instance is a little bit different. And it's just joy for me to write. But there are some macros that I've got, for example, the way a programmer will zero a register is you XOR it with itself, because it's very efficient. You could move, like, zero into it, but that ends up taking up more bytes of code than XORing, for example, the EAX register with the EAX register.

Leo: And you just have to know that. You just learn that as you use assembler after a while, you just know what works and what...

Steve: Yes. And in fact we've talked about XOR, and we know that an XOR will give you a one if either of the inputs is a one, but not both. So if you think about it, XORing something with itself always makes it zero, because if it's zero, then the result of it XOR'd with itself is zero. If it's one, then the result of it XOR'd with itself is still zero. So it ends up all being zero. Anyway, my point is that I have a macro called Zero, and that expands to XOR R1 with R1, what the - so for example, in my code, you'll see me say Zero EAX. Because that's what I'm intending. So rather than saying XOR EAX, EAX, I say Zero EAX.

Leo: So that's a macro you've defined, then.

Steve: Exactly. But the beauty of that is it's much easier for me to see, for my mind to get, that my intent here is to zero EAX.

Leo: It's semantic.

Steve: Exactly.

Leo: It's a code that does what it says it does, or says what it's...

Steve: Exactly. And if I didn't do that, I'd have to see whether I was using the same register both times in an XOR, rather than using it only once and saying the word "zero." Anyway, I'll stick a screenshot of some of my eCommerce system up on our pages. I think people would get a kick out of it. It just doesn't look like assembler. It's why I'm so comfortable writing this way.

Leo: Well, I've noticed that, even when I, ages ago, was writing in assembler, that that's where a good macro assembler can really help you. It can just make it look much more English language like. That's why I asked you if you used a lot of macros. So you do.

Steve: Yeah. I use a few, actually. I use a few, a lot. But I don't use a lot of macros. Although MASM has a very nice feature for allowing you to call the Windows API, and I use that. And it also has nice conditional instructions. I'll find a snippet of code that demonstrates all of this stuff and stick it up on the page.

Leo: Love to see that. Well, Steve, it's really been fascinating. I just - I want to thank you for sharing a little bit of insight into how you work, which is so cool, so interesting. I'm thinking that one of the downsides - I know you love your assembly language, and it's so efficient and so compact. How big is SpinRite?

Steve: SpinRite, because it's got the whole Windows side, it got a lot bigger. I think it's maybe 97K now.

Leo: I have images bigger than that. 97K. But you were about to say it's not portable.

Steve: Yes, exactly. It's not portable. And that's something that I have - I've regretted a little bit that I'm, well, I'm going to have a problem...

Leo: Yeah, I was going to say, because you're x86-specific.

Steve: Yeah, exactly. As 64-bit begins to happen more and more, it's like, agh. Although, you know, Intel's really good about making sure, and Microsoft is really good about making sure that older code continues working. So I think I'll be able to stay in 32-bit mode for, you know, at least for a long time.

Leo: What would going to 64 do? Would the op codes change? Would the language change? Or is it just you have to think in 64-bit registers?

Steve: Yes. You have 64-bit registers. The op codes do change. I've seen 64-bit assembler. I'm trying to remember now what they did. I think they use R. So RAX for Register AX.

Leo: You'd have to rewrite everything.

Steve: That would be - oh, yeah, yeah. I mean, there's no way I'm source-level compatible. Of course I never have been. When I went from 16 to 32, it used to be you had the AX, BX, CX, DX, for example. Then it's EAX, which is the extended, and then they went to RAX, which is the 64-bit register. So it's sort of the same architecture, but just everything's a lot bigger. And I've got to say, though, Leo, I mean, for the stuff I'm doing, I barely need 32 bits.

Leo: Right. [Indiscernible], yeah.

Steve: Yeah. Although it is really convenient to use 32 bits. But, you know, the world moves forward.

Leo: Well, I wrote some assembler, 8086 assembler, in the good old days. And then I moved to Mac and was writing 68000 assembler. And it was, I hate to say this, but it was so much cleaner because you don't have to worry about extended registers. You don't have to worry about the memory issues. And it seems like just such a nice assembly language. You could learn another assembly language. It wouldn't be that hard.

Steve: Oh, yeah. Oh, you know, I'm sure I can. I wanted to correct something I just said, too. I just checked the exact size of SpinRite. I was thinking of the DOS kernel portion, not the whole thing. Because I talked about, you know, with all...

Leo: You're talking about the actual program SpinRite, which is...

Steve: Well, it is only one EXE. But there's a Windows portion and a DOS portion.

Leo: Right, right, right.

Steve: Anyway, the entire thing is 173K.

Leo: Damn that Windows.

Steve: 173.

Leo: Talk about bloat. It doubled the sizes. Come on, Windows, come on. I still have images bigger than that. I think my home page on my website, my home page image is bigger than that.

Oh, Steve, you're great. I love - you know, what's fun is this is - it's a little old-timey, and I'm sure in time it'll seem a little archaic. But this is really how it got done for so long, and still gets done among the pros. This is real software kung fu here.

Steve: Well, I love it.

Leo: Yeah. Steve Gibson, we'll see you again next week for another episode of Security Now!.