Heather:

Welcome to the Hurricane Labs Podcast. I'm Heather, and today we're going to be talking about how ads impact your page's security and how attackers can use cross-site scripting attacks to deliver malicious JavaScript to site users. Here to help with that, I have Hurricane Labs, pentester, Dennis. Dennis, why don't you go ahead and tell us a little bit about what it is that ads do that allow them to give attackers access to the pages.

Dennis:

So ads, sadly, are one of the few ways to make money on the internet, probably the most popular way to make money on the internet, because it doesn't involve a user actually having to type their credit card information into some website. So they're pretty popular, especially on news organizations and those kinds of things. Ads can kind of look at where they are and keep an idea about who you are through session information and other tricky things in order to find out what appeals to you in order to give you the correct ads. People that are concerned with privacy dislike this, because it's a way of tracking people, but it creates an interesting situation. Ads are a great source, as it turns out, for cross-site scripting.

Heather:

Okay. So what is cross-site scripting and why is it bad?

Dennis:

Websites have a JavaScript that they can run on a webpage. If you ever see a website that like... things start moving around in the page, JavaScript did that. There's server-side language and there's client-side language, and the only good client-side language, I'm sorry to say, is JavaScript. JavaScript's the best that we could do so far. Developers are relying on JavaScript because it's kind of free computation. You can... Every visitor is running the JavaScript on their own browser. It doesn't take away from the server's computation. So when you visit a website, you are running code for the website that it gave you in JavaScript. So if I can get you to run bad JavaScript, I might be able to get you to do something bad. This is where cross-site scripting comes in. Cross-site just means on a different website. Scripting means JavaScript. Can I get you to run JavaScript on someone else's website? Can I get my own malicious JavaScript on someone else's website? This happens because of injection attacks, which are kind of facilitated by the craziness of JavaScript. Usually, this happens because the JavaScript or the website, the server-side code is trying to put something in the webpage. It's trying to make something pretty. For example, it's trying to welcome you and say, "Welcome, Dennis, to our website." But it only knows my name because I gave it my name. And if, instead of giving it my name, I gave it HTML, it would say, "Welcome..." and then the HTML stuff, which if the browser read as HTML, because it didn't know any better, then it would manipulate the HTML of the page. In the same way, I can say welcome script tag, and if the browser doesn't know any better and it treats it as a real script tag, it allows the attacker, me in this case, to put arbitrary JavaScript into the page, which then can do anything that the JavaScript in the page that was meant to do. So all the fancy pretty things of stuff moving around that JavaScript does, cross-site scripting can do that. But more importantly, sending requests to the website, JavaScript can do that. So if there's a request that deletes your user, JavaScript can do that. If it's... If you have to click a little HTML popup that says, "Are you sure?" JavaScript can click that popup for you. Anything... Just about anything that you can do on a website, JavaScript can do. The only stipulation though is JavaScript cannot do something on someone else's website. This is a violation of what's called the same-origin policy. I'm not allowed to like ask Google dot... or Gmail... From HurricaneLabs.com, we can't send

a request to Gmail using your session token and find out what your name is and find out, oh you're logged in as this user. So we can't use that to like track you in that way. That would be a violation of same-origin policy, unless for some reason, Google explicitly allows us to do that and gives us some feature by which we can do that. So cross-site scripting and ads, they go together quite well. Ads are in a strange situation. Ads are pretty much injected into a webpage, and usually injecting code into other code is kind of tricky. And it's hard to get your bearings if you're writing the code, that's being injected. The ad doesn't know what website it's going to be put into. It doesn't know what framework it's going to use, those kinds of things. It doesn't know, is it going to be on the left side of the page or on the right side of the page, the top or the bottom, those kinds of things. And the second thing that ads are primarily concerned with is ads want to know when were they loaded and what website loaded them. That's the two big things for ads. An ad wants to know I've been seen, and I've been seen here. That gives the company producing the ads, or the ad network, some information about who is interested in the ad, and they can change their marketing... what they're trying to do with marketing. That offers an interesting input. The URL bar is often injected or put into the ad in some way or another. And if it's not put into the ad in a safe manner, it might cause cross-site scripting.

Heather:

How would you go about finding a cross-site scripting vulnerability?

Dennis:

This wasn't something I was intended to do. I developed, a while back, a Firefox plugin called a Eval Villain. It makes finding DOM cross-site scripting, a particular type of cross-site scripting, particularly easy. Every time I would update a Eval Villain, I'd just go to random websites, usually with ads because that is like a stress test for a Eval Villain, to see how fast it's running and just start clicking around and seeing what happens and making sure Eval Villain is running and doesn't die and doesn't break the website. But as it turns out, I found, seemingly by accident, multiple vulnerabilities and some well-known websites that are reported to HackerOne. So the first one that I found was Imgur, and this is a good illustration of how to not do ads. I'm not talking bad about Imgur. They changed their ways, and they're doing ads in a much better way. Things aren't... They're doing the proper thing now, I believe. But previously, they injected their ads directly into their webpage. This means that the ad is in the same origin as the webpage hosting the ad. All the JavaScript running in the ad can do anything in that webpage that the webpage can do, that the user can do. So a malicious JavaScript that realizes, hey, I'm running on Imgur could delete your user if it wanted to. It could check, are you this user I don't like? And delete the user. So in theory you one user that's angered another could try to buy an ad that seems safe up until it's loaded on the other user's session in Imgur or something like that. Something ridiculous like that. So Imgur is hosting these ads, and while the ads may be nice, the ads might have cross-site scripting vulnerabilities in them. In which case, even if the ad isn't trying to be malicious and make giant annoying popups or something like that, it may accidentally introduce a vulnerability into Imgur's website itself, which could allow another person that didn't create the ad to delete user accounts that they don't like or to post content for another user that is inappropriate. Or in essence, anything that a user can do on the website. So I visited Imgur with Eval Villain on just to see how well it would work, did the website break, those kinds of things. Shortly after I created Eval Villain, I wanted to find cross-site scripting so I could brag out it, in essence, and be like, "Eval Villain's the best. You need to try it out." So I looked around on Imgur for a while, and I found an obvious cross-site scripting injection, where you could make galleries, I think is what they were called, so I could make a... like here's a bunch of pictures that I like, and I can share this gallery with other people. And the gallery would have a name, and the name would be put into the page. That's just like what we were talking about earlier. So that name, if

we put in script tags, might cause cross-site scripting. That's the general idea, and I found a vulnerability there. And it was pretty you straightforward. So when I reported it, I was kind of surprised I even found it that like, this is a pretty easy thing to find. And after I reported it, I was notified that I'm not the first to report this. And Imgur, the way they do bug bounties is wonderful. When there's a duplicate... At least in this case. I don't know if they do this all the time. When there's a duplicate, they included me on the original poster's finding so that one, I know for sure they didn't lie to me, that it is a duplicate, it does exist, and it was done before me. This is the same finding. Number two, it helps me to find that like this is the same finding as me, and so I can... I don't have to worry that they'd made a mistake. And number three, I can add any of my information that might help Imgur solve this problem faster. So that was really cool of Imgur. Not long after that, not long after the duplicate was found, I was just browsing Imgur, testing of Eval Villain, and a finding popped up in Eval villain that disappeared when I refreshed the page. The reason it disappeared is because the finding was actually in an ad hosted on Imgur. So the vulnerability only appeared on Imgur when the correct ad loaded. So I had to reload the page a ton. It was at least 100 page reloads before the vulnerable ad appeared again. At which point, I had an opportunity to audit the situation, look at the source code. The vulnerability was in JavaScript, so I could look at the source code, try and construct an actual payload that'll work. And then I had to reload the page another 100 times in order to see if my payload actually worked so I could get a screenshot of the vulnerability. While I was reloading the page then, I found another vulnerability in a different ad. I ended up with three different ads that have vulnerabilities in Imgur. It really stinks as a website owner to host ads and just import vulnerabilities from whoever wants to host them on my website.

Heather:

What can you do? How can you help keep that from happening?

Dennis:

This is how you should do it. Imgur fixed this by sandboxing ads, and you should sandbox ads with an iFrame. Does that make everything great? Makes things much better, but not perfect.

Heather:

How have you seen sandboxed ads still be vulnerable?

Dennis:

So another time, I was browsing internet radio stations, and Eval Villain popped up and said that an iFrame running on the page had a cross-site scripting vulnerability. It notified me that tpc.google.syndication has... is loaded on the page, and my user input was being put into that iFrame and an unsafe way. At that point, I found cross-site scripting in a Google domain. Now Google does bug bounty, so I hit up Google's bug bounty as quickly as I could with shaking hands, thinking I'm a cool guy now. And then the next day... Of course, I couldn't sleep. The next day, I couldn't get out of my head. Next day, couldn't get it out of my head. No response. I think Google is busy dealing with a bigger, much more important closure vulnerability. And in that time period, when I'm just thinking about what is Google going to say? Are they going to like my vulnerability? Do they think I'm a cool guy? All those kinds of silly things... I realize I maybe should read the disclosure program well. When I read it well, I realized that Google doesn't count bugs for sandboxed domains, and everything about tcp.googlesyndication.com looks to be a sandboxed domain. But the bug bounty system does say that if you can prove that it's still a vulnerability, even though that it's a sandboxed domain, they will accept it. Now, I don't want to look like an idiot in front of Google, so I went about doing my best to prove that it's

still a vulnerability. Now, I should say upfront, this is a kind of a stretch as to whether this is a vulnerability or not. It's not a very great vulnerability, but Google did consider this a vulnerability. But they'd already known about it. It was a duplicate again. We have cross-site scripting inside of an iFrame that is a Google domain. This is... This iFrame, only hosts ads. It does not host any... It doesn't host any other content. The whole point of iFrames, of sandboxing the ad... In this case, they're called SafeFrames, the idea that Google was using... is that you're removing the ad from interacting with anything important. So if the ad is absolutely malicious, it can't do anything really bad. So if there were session tokens in that iFrame, then that's... then I would have a vulnerability.

Heather:

So if you're going to put ads in your site and you're using SafeFrames, then...

Dennis:

Don't put anything else, don't put any session tokens in there, nope, no nothing. It's just the ad by itself. Don't give the ad something to break that isolation. That is what Google did, access to other origins through cores. If you want to go down that rabbit hole, you can Google it. That iFrame didn't seem to have any privileges, but still I don't want to look like a fool in front of Google, so I had to figure something out. I mean, this as a lesson, that if you're going to do SafeFrames or iFrames, it turns out you should either not accept arbitrary content. Google, it turns out, the reason that this vulnerability existed, the reason I could inject content like this was Google was accepting the name, the window name, as the ad. Ads loaded through the window name, and you can just... you can iFrame... To this day, you can iFrame googlesyndication, that URL, with a name that will inject arbitrary JavaScript into the page. And I was isn't even the first to find this. Years ago, someone figured this out, that you can inject arbitrary content into the googlesyndication frame. But I didn't want to look bad. So how can I make this bad? How can I force this into something bad, so that at least I don't look stupid? I didn't get anything amazing. This isn't amazingly terrible. I'm not trying to brag or something like that. But if you are going to make SafeFrames, if you're going to put ads in your website, if you're going to put Google ads in your website, there's a better way to do it, and then there's a less good way of doing it. The better way to do it is to use a random domain. So a random origin, I should say. In this case, the way that Google fixed this problem is they randomized a subdomain of googlesyndication. So now by default, if you put ads into your page, Google will randomize its subdomain. You can opt out of this, but if you do, then you get this attack that I'm about to describe that I told Google about. So we have an iFrame, and it doesn't have any important information in it, except for maybe the web name of the website that's hosting it or some of the information about the website that's hosting it because the ad wants that information in order to do analytics and stuff like that. We could, by putting JavaScript into an iFrame, potentially steal that information. Additionally, we can also replace what's in the ad. So those are the two things that we can do. Once we have our malicious code running inside of Google's iFrame, we'll have same-origin access to any other iFrames of the same origin. So if I got an ad on your website in one iFrame, and there's another iFrame that's the same origin, my malicious ad can crawl around and look for any other iFrames of the same origin, finding someone else's ad on the website, my malicious ad can change that ad into its own ad or it can replace the content of that ad. Like if it's a rival company, it can place the content of that ad with something objectionable in hopes that the user will click the little button that says, "Hey, this is offensive," and report the ad as being objectionable. So that's the first way that we can use this in a malicious way, is by if we're an ad, a malicious ad, we can manipulate other ads on the same page. But we can take this a step for further. There's another channel of communication that we can use called the window.opener. When one tab opens another tab, depending on how it opens it, a channel of communication is open between the two. This is kind of an old technology. This isn't used very often

anymore. But back in the day before tabs existed, when you'd open a new window, it would open a whole new window. Quite often, that would be a prompt to put in user information, and that user information needs to be sent back to the original window. And window.opener is the mechanism for doing that. In essence, it's a channel to the entire other window that allows you to do anything in that window, as long as you have same-origin access. So if we had two googlesyndication tabs with a window.opener that could talk to each other... But that's not a situation that's going to happen, no one ever visits, googlesyndication. It's always iFramed. But it turns out, if you have two websites that have a window.opener between them, two tabs that can communicate through window.opener, and both of them are hosting Google ads through googlesyndication, the ads in one page can talk to the ads to the other page and manipulate the other ad, changing its appearance, reading its information, so on and so forth, through the parent document that hosts the iFrame, then through the window.opener, and then searching the opener's webpage for the other iFrame. So it's a weird matter of crawling up and out of your iFrame across to another tab and then back down into an eye frame of the same origin. So using that we can make a malicious website that hosts a malicious googlesyndication iFrame. Googlesyndication allows ads to put arbitrary content in there through the window.name. Since we're hosting the iFrame, we can make a window.name of anything we want and host any code we want inside of the googlesyndication. That means we can steal googlesyndication's origin, so to speak, use that origin to crawl up and out of the tab and into the next tab and look around in that tab to see if there's any other iFrames of the same origin, any other ads. If there is an ad there, we can replace the content of the ad to be a jerk and put something malicious or objectionable or maybe to steal the ad to advertise on our own thing. Additionally, we can look through that ad and see if there's any interesting information in there. Two tabs, even with a window.opener, they're not allowed to interact unless they're the same origin. If they're the same origin, they're the same website, and so it's not a big deal. If they're a different website, that's a different origin, and they're not allowed to read the URL of the other tab. This would be a big privacy violation. If you were on a website looking for sensitive information about health records, about something you don't want people to know you're looking for, then you don't want to expect another tab to be able to see that information. And this is good because you would expect a website that hosts sensitive content would be aware that they're hosting sensitive content and wouldn't share your information with other people. This violates that in the sense that I can load an iFrame in one tab that's malicious, crawl up and look for a Google ad in the other, so any website that has Google ads, and there are a ton of websites that have Google ads. And from that, reading the Google ad itself, I can usually find the URL of the webpage that is hosting the Google ad. This means that in the case of Gfycat, for example, if you are browsing Gfycat while you have a tab open to my malicious website, my malicious website can do the silly iFrame tricks and check out what Gfycat image you're looking at. It can also read what you searched for when you search for something. Same with Zillow. It turns out Zillow, I could do the same thing and see what house you're searching for, the address of the house you're looking at, the price range that you're interested in if you're shopping, things like that. I didn't find... Again, this isn't a terrible vulnerability because it depends on the website. It's neat in the sense that it's widespread because, well, ads are everywhere, but it's not terrible in the sense that like it depends on what website you're at and whether it's sensitive or not. Additionally, you have to have this silly tab opener thing, which isn't necessarily easy to come by. Although, a hacker could potentially bring this about on purpose. I never found this situation, but some websites, it's considered a vulnerability in and of itself usually. The URL you shouldn't really consider as a secret space. So if you put in a session token into the URL bar, a session token that allows you to assume that user's identity, that's usually a vulnerability in and of itself because of things like this. But if you do do that on a website, then this tab-tracking idea using Google ads would allow an attacker to steal that information from the URL bar and then assume that user's identity and manipulate that user's session. But I share this information because

if you are going to put ads in your website, you should do it the right way. After I found the Google vulnerability, I found a very similar vulnerability and reported it to Amazon. Amazon was a simpler find. Amazon only had it on Amazon's website, and so I could track, using a window.opener, what you were searching for. And so if you're looking around on Amazon for medication or for something you don't want people to know about, you can usually trust Amazon to keep that information safe because if they didn't, then people wouldn't shop it on Amazon. But because of this... a very similar tab-tracking system, I was able to do the same attack on Amazon so I could see what you were trying to buy. Amazon was very smart though. They do not put ads on the checkout screen. Even sandboxed iFrame ads, they don't put it on the checkout screen. That's a very good idea because the checkout screen, maybe it has sensitive information inside of the URL bar. It's best just not to put any ads there just in case.Amazon fixed the vulnerability by... in a very different way than Google. Google fixed their vulnerability by blocking... by choosing a random subdomain to keep ads from being same origin. Amazon fixed it by not allowing arbitrary content. Amazon previously allowed any content into the iFrame. Now they don't. So it might be better to randomize the subdomain anyways, but I couldn't find any way around their problem. So to wrap this whole thing up, one, I'm not an expert on ads. This is stuff I just kind of fell into and wanted to talk about. So maybe look at the actual SafeFrame specification in great detail if you're going to do it. Two, use a SafeFrame. Don't just put the ad directly inside of your page because all kinds of bad things can happen. Next, in your SafeFrame, don't put anything except the ad. You're trying to segment the ad off of everything else, so just put one ad in one SafeFrame and that's it. And finally, if possible, randomize the subdomain of the SafeFrame so that SafeFrames can't talk to each other. If you use Google ads, this is the default behavior now. Alternatively, don't allow arbitrary content into the SafeFrame. So in short, if you do those, your website will be more secure and safe from vulnerabilities that might occur due to ads.

Heather:

All right. Well, thank you, Dennis, for coming in to talk to us. I appreciate it. That's all for today. And until next time, stay safe.