PROFESSOR: All right, let's get started. So welcome to another exciting lecture about security and why the world is so terrible. So today we're going to talk about private browsing modes, something that a lot of you probably have a lot of personal experience with. At a high level, what is the goal of privacy? When security researchers talk about privacy, what are they talking about?

Well at a high level, they're talking about the following goal. So any particular user should be indistinguishable from a bunch of other users. In particular, the activity of a given user should be non-incriminating when viewed in light of activity from a bunch of other different users. And so, as I mentioned, today we're going to talk about privacy in the specific context of private web browsing.

And so there's actually no formal definition for what private web browsing means. There's a couple different reasons for that. So one reason is that web applications are very, very complicated. And they're adding new features all the time like audio and video capabilities and things like this. As a result, there's this moving target in terms of what browsers can do. And as a result, what information they might be able to leak about a particular user.

And so what ends up happening is that in practice, like with many things involving browsers, there's this living standard. So different browser vendors will implement different features, particularly with respect to private browsing. Other vendors will look and see what vendor X is doing. They will update their own browser. So it's like a moving target.

And as users grow to rely on private browsing more and more, they end up a lot of times actually finding bugs in private browsing mode, as I'll discuss a couple minutes later in the lecture. And so private browsing at a high level you can think of as an aspirational goal. But we as society are continually refining what it means to do private browsing and getting better in some aspects-- worse in some aspects-- as we'll see a little bit later.

So what exactly do we mean by private browsing? It's tough. But the paper tries to formalize it in two specific ways. So first of all, the paper talks about a local attacker on private web

browsing. This is someone who is going to possess your machine after you've finished a private browsing session. And it wants to figure out what sites you looked at in private browsing mode.

And the paper also talks about web attackers. The web attacker is someone who controls the websites that you visit. And this web attacker might want to try to figure out that you are some particular person John or Jane as opposed to some amorphous user that the website can't tell who they are. And so we'll look at each one of these attacks in detail. But for now, suffice it to say that if the attacker can launch both of these attacks-- both a local and a web attack-- that actually really strengthens their ability to try to dearm us.

So, for example, a local attacker who, for example, maybe knows your IP address can actually talk to the website and say, hey, have you seen this particular IP address in your logs. If so, aha! You're looking at the user whose machine I control right now. So it's actually pretty useful from a security perspective to consider these local and web attacks. So they are separate things and then to see how they can possibly compose.

So let's look at this first type of attacker, which is the local attacker. So as I mentioned, we assume that this attacker is going to control the user's machine post-session. And so by post-session, I mean that the private browsing activity has already finished-- the user has perhaps gone off and done something else.

It's not at the computer. And then the attacker takes control of that issue and wants to figure out what's going on. And so the security goal is that well we don't the attacker be able to figure out any of the websites that the user visited during this private browsing activity. Now, the reason why the post is actually important there is because if we assume that the attacker can control the machine before the users private browsing, then basically it's game over, right, because the attacker can install a keystroke logger-- the attacker can subvert the binary that [INAUDIBLE] the browser. The attacker can subert the OS.

So we don't really care about this pre-session attacker. And also note that we're not trying to provide privacy for the user after the attacker has controlled the machine. And that's for the same reason. Once the attacker gets to the machine, he or she can do the same thing that is mentioned-- key logger.

So, basically, once the user leaves the machine, we don't assume any forward notions of privacy. Does that make sense? It's pretty straightforward. And so you can imagine that

another goal that you might want to try to satisfy here is you might want to try to hide from the attacker that the user was employing private browsing mode at all.

Now the paper actually said that's very difficult. This property is often called plausible deniability. So your boss comes up to you after you use private browsing, and says were you looking at mylittlepony.com? No, no, I certainly wasn't. And I certainly wasn't using private browsing mode to hide the fact that I was looking at mylittlepony.com. So as I said, the paper said it's difficult to provide this property of plausible deniability. I'll give you some concrete reasons why this might be the case a little bit later on in the lecture. But that basically is an overview of the local attacker.

So one question we might want to think about is what kinds of persistent client-side state can be leaked by a private browsing session? And by persistent, I just mean stuff that will end up getting stored on the local hard disk, the local SSD or whatever. So what kinds of state might be leaked if we weren't careful when someone is doing this type of private browsing?

So one thing you might be worried about is JavaScript accessible states. So examplees of this includes things like cookies and DOM storage. Another thing you might be worried about-- and this is what most people think about when they think about what they want to say in private browsing-- is maybe the browser cache. So you don't want someone to look in the inner cache and figure out here are some images or HTML files from websites you prefer people didn't know that you visited.

Another important thing is your history of visited sites. So many of your relationships have been broken when the other goes to the browser-- start typing something into to the address bar and all of a sudden it auto-completes to something very embarrassing. So this is one thing definitely you don't want to leak outside the private browsing session.

You can also think about configuration states with the browsing. And so here you could think about things like client certificates. You could also think about stuff like bookmarks. Maybe if you logged into a particular site and the browser offers to store your passwords in another type of configuration state that you might not want leaking from private browsing mode.

Downloaded files-- as we'll discuss, this one's a little bit interesting because downloading a file actually requires explicit user action to download that file. Maybe we do actually want this stuff to leak outside of private browsing mode. Maybe if you download something in private browsing mode, it should actually be accessible when you open the browser or use the

machine after that session. So we'll talk about this a little bit in a second.

And then, finally, during private browsing mode, you might install new plug-ins or browser sessions. That's another type of state that you might imagine you don't want to leak outside of private browsing mode. So, basically, current browsing modes typically try to prevent one, two, and three from leaking outside of private browser sessions. Right? So there shouldn't be any cookies or DOM stores to get out of there.

Anything you put in a cache during a private browsing session should be deleted. And you shouldn't have any history of the URLs that you're using. Typically, four, five, and six private browsing modes allow to leak outside of a session. And there's some good and some bad reasons why this might be the case.

And as we'll discuss later, we'll see if you allow anything to leak from the private browsing session, that actually radically increases the threat surface of private leaks. So it becomes much more difficult to reason about what the security properties are for private browsing mode. Does that all make sense? Anyone have any questions? It's pretty straightforward.

So the next thing we're going to talk about very briefly is network activity during private browsing mode. And what's interesting about this is that even if we cover all this stuff-- we don't allow private browsing to leak anything from there-- the mere fact that you're issuing network packet connections leave evidence of what you were doing.

So imagine when you want to go to foo.com, the website, your machine actually has to issue a DNS resolution request for foo.com. So even if you don't leave any of this type of persistent state up there, there may be records in your local DNS cache that you, in fact, tried to resolve the hostname foo.com.

That's very interesting. So you can imagine that browsers could try to flush the DNS cache somehow after the private session was over. Now, in practice, that's actually tricky to do because on many systems, you require administrator privileges to do that. So it's not clear if you want the browser running as root because browsers, as we've seen, are somewhat untrustworthy individuals. And also too-- a lot of DNS flush commands-- they don't actually act per user.

They flush the entire cache, which is typically not what you would want if you're implementing private browsing mode. You'd want to use a type of surgical thing where I only want to get rid

of foo.com and things that were visited during this private browsing sessions, but not delete other things. So in practice, that's kind of a tricky thing to handle.

And another tricky thing to handle, which the paper mentions-- are these things that I'll call RAM artifacts. So the basic idea here is that during private browsing mode, that private browser has to be keeping some stuff in memory. And so even if the private browsing mode doesn't issue any direct I/Os to disk-- user rights. The RAM that belongs to that private browsing tab can still be reflected into the page file, for example. It can still be reflected into the hibernation file, for example, the laptop.

And so if that state gets reflected into persistent storage, then what may end up happening is that after your private browsing session is over, the attacker can look in your page file, for example, and find, for example, JavaScript code that was reflected to disk or find HTML that was reflected to disk.

So we're going to have a little demonstration of how this might work. So if you see up here on the screen, I basically loaded up private browsing tabs. And so what I'm going to do is I'm going to go to some website. So this is for the PDOS group here at CSAIL.

I've loaded up that page. And then what I'm going to do is use this fun command called gcore. So, basically, I'm going to take a memory snapshot of this running page. And so I will do the following magic. So basically there's going to be some work that my terminal is doing to generate that memory snapshot.

So this takes a little bit of time sometimes. Now, what's happening here. So now we've basically generated the core file for that private browsing image. So what we're going to do now is we're going to look inside of that image and see if we can find any mentions of PDOS.

And so what's interesting is we see a ton of instances of the string PDOS in that memory image for the private browsing mode. And so what is interesting is we actually see various prefixes for things. If we look further up, we can see things like there's full URLs here and things like this. You also find HTML code in there.

So the point here is that if we found all this in the memory of that page, then if this-- if any of those pages got put to disk in the page file, then he attacker could basically just run strings. So they could do what I just did over the page file and try to find out what sites that you visited in private browsing mode. So does that make sense?

Basically, the problem here is that private browsing modes don't try to obfuscate RAM basically or encrypt it in any way. And that seems like a pretty fundamental thing because at a certain point, the processor has to execute on clear text data. And so this is actually a pretty big challenge.

So does anyone have any questions? Yeah?

AUDIENCE: So one thing is I don't expect my browser to do that. One thing is that these browsers-- the guarantee that they give you through private browsing-- the example they give is if you're shopping for something, your layman friend can't go on the computer and see the things. So can you talk a little bit about what guarantees they give and if they had to change anything as a consequence of this paper?

PROFESSOR: Yeah, it's very interesting. One thing you can look at is when you open up a private browsing tab, typically there will be a little blurb that says, hey, welcome to incognito mode. Here's where we'll help you against. We won't help you if someone is standing behind you with a rubber hose about to beat you. And so the browser vendors themselves area little bit cagey about what guarantees they provide.

And in fact, after the Snowden incident, a lot of the browsers actually changed that splash page because they wanted to actually make it clear that we're not actually protecting from strong ways with the NSA or something like that. So long story short, what guarantees are they providing you? In practice, they're providing that weak thing that you mention there.

It's like a lay person who wanted to see what you were doing afterwards couldn't figure out what you were doing. And we're assuming the lay person can't run strings on the page file or things like that. Now, the problem-- there's actually two problems though. One problem is that first of all, because browsers are so complicated, they often don't even protect against the layperson.

I can give you a personal example. So a lot of times when you see those ridiculous ads from "Huffington Post," like, oh, my gosh. It's like puppies trying to help small puppies go down stairs and things like that. Right? Because I'm weak, I will sometimes hook on those things. But because I don't know want people to know that, I'll sometimes do that in private browsing mode.

So what will happen sometimes is that sometimes I'll see those URLs will leak into my URL

history like my regular, public mode browser, which is precisely what this stuff is designed not to do. So one problem is that sometimes these browsers don't provide protection against the layperson attackers.

The second thing is I think that there are actually a lot of people who would like for private browsing mode to provide something stronger, particularly with the whole Snowden thing. I think there is a lot of people increasingly who would like private browsing mode to protect, for example, against these RAM artifact attacks, even though they may not be able to technically articulate that goal.

And so actually one of the things I've done while I was here, I got to do some research in a stronger private browsing mode protection. So we can chat about that after all. One of the things we learn about all professors is that we will talk about our research endlessly. So if you want to talk about that for three hours just send me a calendar request. And we can do that.

So, anyway, this is basically a demonstration. Oh, you had a question?

AUDIENCE: Yeah, about the RAM. So I'm not familiar with how it works exactly. How come a browser can't at the end of a session, just ask the OS to flush those parts around that he was using?

PROFESSOR: So we're actually going to get to that topic in a couple of minutes. But you are correct. At a high level, what you can imagine is that maybe the OS when it, for example, killed a process, would actually go through all those numbered pages and write zeros to all those pages. Or you could also imagine that maybe the browser tried to pin all the pages in memory to prevent anything from getting flushed out at all.

So there are some solutions that can do that. So hold onto that question for one second. This is basically an example of how data from RAM can leak onto disk through paging activity. But note that data lifetime is a bigger problem than just in the context of private browsing.

You can imagine that any programs that deals with, let's say, cryptographic keys or user passwords will have this problem. Anytime you type in your password to a a program, the memory page which holds that password can always get reflected to disk.

So let me show you another example of this. So let's say that we looked at the following program, which is pretty simple. It's called memclear. So you see here at the bottom and main, we're just going to read in some secret text file here. And then we're just going to sleep

forever. So what is that Read Secret do? Basically, it reasons from file.

It's going to print out the contents of that file. And then it's actually going to clear out the buffer that was used to store that secret information. So getting back to your issue. So one can imagine the browser, for example, would try to just memset to zero all the secrets that it encountered when it's just in private browser.

So if we look at the secret files, it's not very fun. It just says, my secrets of in a file. And then if we run this program, in the background-- so what did it do? So like I said, it just printed it out. It read that file in, printed out the secret value-- cleared the memory buffer that it used to print that stuff out. Now it's just sleeping in the background.

So once again, if we use this fun gcore command, we can take a memory dump of the memclear program that's running in memory right now. OK, and then if we do-- let's see which ones we're going to look at. So then if we look at-- this guy is the one we want. And then we do a grep for a secret.

So once again, we see that if look in the RAM image of that running program, we found instances of both the file name that was read in and also some prefixes of the string contents of that file, even though we wiped the buffer in the C program itself. So you might say why did this happen? This seems very, very strange.

And the reason is that if you think about the way that I/O works, it's like a layer type thing. So by the time that the contents of that file get to the program, it's already gone through, let's say, the kernel memory. It's already gone through maybe like the C Standard Library to do I/O because that library does buffering and stuff like that.

And so what ends up happening is that even if you memset the application visible buffer, there are still instances of secret data lying in many different places throughout the system. And this is looking at the user mode portion of this application. So there's probably still data sitting around in maybe like the kernel I/O buffers or things like that.

So getting back to your question, if you want to do what they call security allocation, you can't just rely on mechanisms at the application level because there may be other places where that data lives. So what are some examples of other places where this data might live? So, for example, it might live in a process memory.

So these are things like the heap and the stack. So when we did that memset inside of

memclear.c, we were basically trying to address this. But what we found out is that that is necessary, but insufficient to actually clear all instances of that secret from memory. So where else my RAM artifacts live or secret data persists-- so all kinds of files-- backups-- SQL write databases.

If at any point, an application takes something in RAM and writes it to one of these things, then once again, the attacker may be able to recover that after the attacker controls the disk . As I mentioned, a kernel memory is another common place where RAM secrets may live because, once again, applications typically do layered I/O in which each piece of data goes through multiple parts of the stack.

Think of like network transmission, for example. First, the data has to come to some network buffer that's probably inside the kernel. Then once again, it probably goes through some buffers inside the C Standard Library. And then finally it will go to the user mode-- the part of the application that the developer wrote him or herself.

So that can actually be a big problem. You can also think too of freed memory pages as being a place where data can leak. So imagine that your application allocates a bunch of memory using whatever [INAUDIBLE] or whatnot. And then that process dies. And the kernel sends out another process but hasn't actually zeroed out all the physical RAM page.

So what could happen is that when that new process spins up, it could just do a walk through all this physical RAM pages and use a bunch of memory and just do the same thing-- do the strange thing-- see if there's anything interesting there. And then they might be able to get secrets that way.

So there's a lot of ways information is leaked from the kernel. You could also think about I/O buffers and things like a keyboard from things like the mouse. There's just a bunch of different factors that data can leak through the kernel.

How might an attacker try to get some of this information? Well, in some cases, it's just as simple as reading the files-- so just read the page file. Read the hibernation file and just see what's in there. Some file formats actually embed different versions within themselves. For example, the way that Microsoft Word used to work is that a single Word file would actually contain versions for old pieces of data.

So if you could get access to that Word file, you could just sit there through either format and

so step through all the old versions. And so as we have been discussing in the last couple minutes, security allocation is also a problem. It cannot supported a full stack. So for example, an older Linux kernel-- when you would create a directory, end directory, you could leak up to four kilobytes of kernel memory.

Only Zeus knows what's inside that memory. And that's because Linux wasn't actually zeroing out kernel memory that had been allocated, deallocated, and then allocated to something else. So as I mentioned before too-- if the kernel doesn't zero out pages that are given to user mode processes, you can also have user mode secret leaks through those types of menu pages as well.

Another thing is that-- SSDs-- many of them implement logging. And so in other words, when you send a write to an SSD, oftentimes you are not directly overwriting data, you're actually writing to a log. And when a piece of data becomes invalid, it lays away your claim.

So what that means is that if you as the user get unlucky. And you've written a bunch of data that hasn't been reclaimed by the SSD, then maybe the attacker can look at that hardware and say, oh, OK, I understand the log format. And even though technically speaking, this data may be invalid, I can still recover because I understand how the Flash translation layer works or something like that.

And at a high level, you can also have this problem with stolen or discarded hardware as well. If you don't use encryption, then a lot of times, you can just take some disk that you found in a dumpster somewhere-- you understand what the physical layout is and recover data like that.

So anyway, there's a lot of problems with these RAM artifacts getting stuck in persistent storage somehow and then being available for an attacker later on. So how can we fix these data lifetime problems?

So we've already discussed one solution, which is to basically zero out memory when you're done with it. So whenever you deallocate something, you just go through. You write a bunch of zeros or some random thing and then essentially hide the old data from someone else who might come along later.

So does anyone see potential any potential problem with that? One problem you might imagine is that as with all things in security, people always complain about performance. And so when you say that you zero out memory, maybe this isn't a problem if your program is I/O

bound. So you're waiting on some slow, mechanical part of the hard disk or whatnot.

But imagine if your program is CPU bound. And maybe it's very memory intensive too. So it's always allocating and deallocating data. So maybe zeroing out memory might be performance cost that you don't want to pay. Typically this isn't a problem in practice. But as we all know, people love performance. This is sometimes an objection that you'll have with this approach. Another thing you can imagine doing is that instead of zeroing out memory, you always encrypt data as it goes to stable storage.

So in a system like this, basically, before the application ever writes anything to disk, it's actually going to encrypt it before it actually hits that SSD or that hard disk. Similarly, when the data comes back in from stable storage, you're going to decrypt it dynamically before you put it into RAM. And so what's interesting about this approach is that if the key that you use to decrypt and encrypt data-- if you throw it away, then once you throw it away, you've effectively made that data on disk unrecoverable by the attacker, assuming that you believe in cryptography.

So this is very, very nice because it gives us this nice property that we don't have to remember per se all places where you've written this encrypted data. We can just say why drop the keys? And I'll just treat all that encrypted data as it's something that I can allocate again. So, for example, if you look at Open BSD, they have this option where you can do swap encryption.

So you can basically associate keys with various sections of the page file. So it does this very thing I mentioned. So every time you group the machine, it'll generate a bunch of new keys. And then when your machine goes down because you shut it down or you reboot it or whatever, it will basically forget all the keys that it used to encrypt that swap space.

And then it can basically say now all that swap is available to be used again. And so because those keys are forgotten, one can assume that the attacker can't look at the stuff that is in there.

AUDIENCE: What is the [INAUDIBLE]?

PROFESSOR: Ah, yeah, that's a good question. I'm actually not sure what sources of entropy it uses. Open BSD is pretty paranoid about security. So I imagine it does things like it looks at let's say the entropy pool gathered from user keyboard input, for example, and other things like that. Yeah, I'm not actually sure how it drives those keys.

But you're exactly right that if these sources of entropy that it uses are predictable, then that basically shrinks the entropy space of the key itself, which then makes the key more vulnerable.

AUDIENCE: So with the memory it's capturing [INAUDIBLE].

PROFESSOR: Yeah, so basically, what this model assumes if all we are doing is looking at the swap encryption, It assumes that the RAM pages for the keys, for example, are never swapped out. And that's actually pretty easy to do if you're the OS of if you just pin that page to memory. And this also doesn't help you with someone whose got pins with the memory bus or someone who can walk the kernel memory page or stuff like that. So you're right.

AUDIENCE: In terms of browsing, it helps of attackers that come after the fact because if you have to throw away the key, then after the fact, there is no key to memory.

PROFESSOR: Yeah, that's exactly right. So what's nice about this is that it essentially doesn't require modifications to applications. Like you said, you can just put any old thing atop this and get this property for free.

AUDIENCE: Going back a bit-- if you look at the data before [INAUDIBLE] to RAM. How does that avoid the RAM artifacts [INAUDIBLE]?

PROFESSOR: OK, so if I understand your question correctly, I think you're worried about the fact that, sure, data is encrypted when it's on disk, but then it actually can sit in clear text forms somehow in the actual memory itself. So this gets back to the discussion that we had here. So ensuring that data hit the disk encrypted doesn't actually protect against an attacker who can look at RAM in real time.

So basically what we're saying is that if you're only worried about this post-session attacker who can't, for example, look at your RAM views in real time, this works fine. But you're exactly right that this does not provide, for lack of a better term, encrypted RAM. And there actually are some research systems that try to do something like that. It gets a little bit tricky because at some point when you look at your hardware, your processor, it has to actually do something on real data like if you want to do an ad and you have to pass a clear text operands perhaps.

There are also some interesting research systems which actually try to do computation on encrypted data. This is mind blowing like "The Matrix." But suffice it to say that protections that

people have for in RAM data are typically much weaker than what they have for data that lives on stable storage. You got a question?

AUDIENCE: Yeah, but does that [INAUDIBLE] because even though the attacker has post-session access, that's just post-private mode access. So there could this could still be a public mode session going on. And the attacker would have access to the machine, right?

PROFESSOR: So you're worried about if a concurrent--

AUDIENCE: So if you have a public mode tab and you have a private mode tab. You close the private tab and the public mode tab stays on-- the attacker could still dump the memory. And the RAM artifacts would be problematic. Is that right?

PROFESSOR: Yeah, interesting-- so we will talk at the end of lecture about an attack which is somewhat similar. So most of the threat models of private browsing due not assume a current attacker at all. In other words, they assume that when you're doing private browsing, there is no other person who have a public mode tab open or anything like that.

But you are in fact correct that the way that private browsing modes are often implemented-- let's say you open up a private browsing tab, you close that tab. You immediately run to go get a cup of coffee. So one attack I will describe is that Firefox, for example, still keeps statistics about, let's say, memory allocation.

So if the memory for your private tab is actually laid with the garbage collected and I can basically go to about.memory or whatever and actually see URLs and stuff in your tab. But yeah, but the long story short, most of these attacker models do not assume a concurrent attacker at the same time that you're privately browsing. Make sense?

So this is one that you do-- do swap encryption like I mentioned. This is nice because this gives you some pretty cool security properties without having to change the browser at all or any of applications running on top of this. And in practice, the CPU cost of doing this kind of thing is much, much lower than the actual cost of doing I/O in general, particularly if you have a disk because with disk you're particularly paying C cost. That's a mechanical cost. This is all processing cost-- pure computational stuff. So typically this not that big of a performance hit.

Oh, god there's physics here. This is always an adventure. So the next attacker that we're going to look at is this web attacker that I mentioned at the beginning of lecture. So the assumption here are that the attacker who controls the website that the user is going to visit in

private browsing mode-- how the attacker does not control the user's local machine.

And so the security goals that we want to have against the web attackers are two fold. So first, we don't want the attacker to be able to identify the users. And by identify with, we just mean we don't want the attacker to be able to distinguish the user from any other user that happens to be visiting the site.

And you also might imagine that perhaps we don't want the attacker to tell whether or not we're using private browsing mode. So the attacker can't tell the user employees private browsing. And so as the paper discusses, defending against the web attacker is actually pretty tricky.

So what does it mean, for example, to identify different users. Like I said, at a high level, as you could imagine, the user looks no different than any other users that visits this site. So you can imagine a web attacker might want to do one of two specific things. It might want to say, OK, I see multiple people who were visiting my site in private browsing mode.

You were visitor five, seven, and eight. So in other words, identifying a particular user within the context of multiple private browsing sessions. The second the attacker might want to do is actually try to link a user across public and private mode browsing sessions. So I go to Amazon.com once in public browsing mode. I then go to it again in private browsing mode. Can the attacker actually figure out that I'm actually the same person through those two visits. Yes?

AUDIENCE: This is all a module of the IP address.

PROFESSOR: Ah, yes, that's exactly right. That is excellent foreshadowing. So right now I'm assuming that either user employs Tor or uses something like this. So yeah, we're punting on this whole issue of IP admittedly for now. That's right. So yeah, this segues very well. So what's an easy way to identify the user, as you suggested, the IP address.

So it's a pretty high likelihood if you see two visits that are sort of close in time relatively speaking with the same IP with high likelihood that's probably the same user. And this in fact the motivation-- one of the motivations for stuff like Tor. And so we're actually willing to discuss Tor next lecture. So in case you haven't heard of Tor, it's basically a tool which tries to obscure things like your IP address.

And you could actually imagine layering Tor-- having Tor be the foundation. And then you put private browsing modes atop that. And that might give you some stronger properties then you would if you used private browsing modes at all. But, anyway, so the thing to mention about Tor though is that Tor does provide some sense of IP anonymity. But it doesn't actually address things like the data secrecy lifetime issues or things like that. So Tor-- perhaps you can think of it as maybe necessary, but insufficient for a full implementation of private browsing mode.

And so what's interesting too is that even if a user employees Tor, there are still ways that a web server can identify the user by looking at the unique characteristics of that user's browser. So this is our final demo for today. So let's see here. So going to get rid of this guy. And then let's see. I am going to go to this site called Panopticlick.

Some of so you heard of this. It's run the EFF. The basic idea is it is going to try to identify you the user by looking at various characteristics of your web browser. So I'll show you exactly what I mean. So I want to go-- the URL is very long. This is very stressful for me to type in. So please don't just if it doesn't go through. Let's see. Panopticlick-- did it work? Yes, OK.

So I am going to go to this website. And it's run by the folks at EFF. And I say, OK, test me. So what this is doing is it's basically running a bunch of JavaScript code, maybe an applet-- maybe some Java. And it's trying to fingerprint my browser. And it's trying to figure out how much unique information does it have.

And so-- let me increase the font here. So, for example, one thing it looks at is it looks at you see here what are all the details of the browser plugins that I'm running. So basically it'll run code in it's web page that looks and sees do I have Flash installed? What version of Flash? Do I have Java installed? What version of Java?

So you can see that these are all-- they can't even fit on the tree at one time. These are like all the various plugins and ridiculous formats that my browser supports. Now, the high level-- this should be troubling to you if you're a security person. Am I actually actively using all of these things at a given time? This gives me nightmares.

So what ends up happening is that web servers-- this web attacker-- they can hunt code like this. And they can figure out what are all the plugins that you're looking at. Now if you look at these two columns to the left, what are they? So you see up here. It says bits of identifying information. And then one in x browsers has this value.

So, for example, if we look at a plugin, it's saying there is basically-- it's probably this is the number that's more interesting. It's no longer right. It's saying that 1 in approximately 280,000 browsers has this exact set of plugins. So that's actually a pretty specific way to fingerprint me. It's saying very, very few people who have this exact set of plugins and configurations.

So as it turns out, they're right. I am quite unique. But this a problem from the security perspective. So look at this. The screen size and the color depths for my machine-- 1 in-- what is this? 1.5 million. That's actually pretty shocking. So there's only one person in a sample of 1.5 million people who have this particular screen image.

So these things-- they are additive in some sense. So the more fingerprints you have, the more easy it is for the attacker to figure out exactly who you are. And so note this was done purely from the server side. I just went to this web page. And I just did this. And this is what it got to. One second-- I want to show one more thing. This was done in private browsing mode.

And let's see here. I will open up a regular version of Firefox. Then I run this up again. So note that now I'm in a public mode browser. Before I was in private mode. Now I am public mode. So what you'll see is that when we look at the browser plugins, the extent to which I can be fingerprinted is essentially the same.

So it's going to be a few plugins that may or may not load depending on the vagaries of how privacy mode is implemented. But still, look at that. I'm still very easy to fingerprint. And in fact, if you look back at this guy again-- that screen size and color depth. I didn't change that actually between the two-- between public and private browsing mode.

So that ability to fingerprint there is basically the same. This is one reason why it's so difficult to protect yourself against this web attack because browsers themselves reveal so much information about you just from their configuration.

AUDIENCE: I am curious the screen size and color depth thing. How does it do that? How is that unique? How many screen sizes and color depths are there?

PROFESSOR: Well, I think it's actually hiding some of the magic that it's using to figure out what that is. So at a high level, how do a lot of these tests work? So there's some parts of your browser environment that are testable purely by JavaScript code. So you can imagine that you can essentially have JavaScript code, which looks over the properties of the window object, which

is like a global JavaScript manuscript and sees how do you define this weird widget? How do you define this weird widget?

And if so, that my count your plug-ins, lets say. Pages like this also typically take advantage of the fact that Java applets and Flash objects can look at all kinds of more interesting stuff like the fonts that are available on your machine and things like that. So as to the particular screen size and color depth thing-- I think-- don't quote me on that. But I think what ends up happening is it will try to run an applet, let's say, that will actually try to query your graphics card or whatever are the graphics interfaces in Java and poke for different aspects of it.

So I think it's actually more than just screen size and depth. They condense it for size as that. So at a high level, that's how all these tricks work. So you see a bunch of information-- you can snarf up through JavaScript. Then you run a bunch of plugins, which can typically access more stuff and see what they can snarf up. And then you see what's going on.

Does it all make sense? Yeah, this is basically why it's very difficult to protect against a web attacker. And in particular, getting back to the discussion we had about Tor, right, even if I had gone through Tor-- so you'll note the IP address-- you don't see it up here. And so you can imagine that yeah, maybe this thing would actually look at your IP address.

But the thing is like even if I didn't know what IP you were coming from at all, I can do all these things. It's pretty maddening. It's pretty insane. So there are some products out there that tried to do things like imagine that you had a proxy out in the cloud that all your web traffic went through. And then imagine that proxy tried to present a canonical version of a browser runtime.

And imagine that it would always try to emulate, let's say, Firefox v 10.7. Then it would try to send back the data that it rendered as Firefox v 10.7. So some people would try to attack this. It's sort of tricky.

AUDIENCE: [INAUDIBLE].

PROFESSOR: I am not--

AUDIENCE: Is that Tor distributions? Is that paired with virtual machines? [INAUDIBLE].

PROFESSOR: I see-- so the basic idea-- is it a similar idea to what we were just talking about?

AUDIENCE: Yes, [INAUDIBLE].

PROFESSOR: Yeah, so I never heard of that one. I have heard of some of these other projects. I'm imagining there's actually some trickiness in getting systems like this to be efficient a lot of times because particularly imagine if you have something that's interactive. It's like you want to play a game or something like that. It's a little bit awkward to send my mouse click to some proxy. That proxy is then somehow going to [INAUDIBLE].

AUDIENCE: Let me clarify the first station virtual machine actually runs [INAUDIBLE] Firefox. In the proxy it's known as a Tor.

PROFESSOR: Ah, it's just a Tor proxy. So if it's a Tor proxy, sure, that's one thing. Then the only overhead there you have to pay is the regular Tor overhead of going through all the onion route. Yeah, so I was talking there are systems-- let's ignore the IP anonymity for a second because they basically try to say you have your own very fingerprintable browser on your own machine. You don't want to expose that to the web server.

So essentially you go through a proxy, which you think of it all the time like a headless Firefox let's say of some canonical version. The web server thinks it is interacting with this thing. So if I go load this site, I am perceived by the web server as Firefox 10.7 or whatever. If you go there, you are also perceived as Firefox 10.7. Then behind the scenes its' spitting out HTML and stuff like that it collected from the proxy. So those two things are orthogonal.

AUDIENCE: But it seems like you don't need a proxy for this. You could have browser support for this, right? Meaning the Tor browser does this already by trying to appear as the most generic version of Firefox.

PROFESSOR: Yeah, so this is true. Although, I think a problem with a lot of those things that even if you try to lock yourself into one version, there's still a lot of things that can fingerprint it. So I think with the Tor distribution, what they often do is they say, we control what's in the Tor distribution. So if we all go down to the Tor distribution, then forshizzle, we're both going to get Firefox with the same Java version-- the same so on and so forth.

AUDIENCE: Well, it's more than that though. They return screen sizes that are the most common screen sizes whenever you clear the screen.

PROFESSOR: That's all true. Yeah, so one thing that's interesting to look at though-- the Tor team that also put out-- the people who do the bundle-- they'll often put out reports about what data still gets

leaked. So stuff does still get leaked out. But you're right. If you could-- the high level of that goal is very reasonable.

It's saying that if we all agreed to download the same distribution and to then not trick it out by adding plugins or stuff like, then you're exactly right. That'd work great. Any other questions? Yeah, so that is it for demo time. And there's more physics. This must have been a riveting previous class. So we will ignore that for the moment. Let's see here.

So where were we? So what is the high-level goal of privacy? And you can think of it as what's your anonymity set if you're a user? So in other words, how many-- what's the size of people-- the number of people that you could be confused for-- you could be mistaken for by an attacker. And so what the browser fingerprinting stuff shows is that oftentimes a web attacker can narrow you down to a very, very tight demographic without controlling anything on your local machine whatsoever.

So that's actually little bit frightening to know. So you might want to think about how can a web attacker determine if you're using private browsing mode? Maybe that's [INAUDIBLE] for some reason. So in the paper they actually describe an attack that uses link colors. So remember, in private browsing mode, the browsers isn't supposed to keep track of the history of the sites that you visit.

And so in the paper, the authors describe an attack in which essentially the attacker-controlled page creates an iframe to some URL that the attacker controls and loads that inside the attacker page. And then it basically looks at the link colors. It creates a link to that page-- that iframe it just created-- and then sees that the link color for that link is the visited color.

So see it as purple versus blue. And the idea that if you do this test in private browsing mode, then presumably the link colors should stay like the unvisited color because the browser is supposed to be forgetting about all this kinds of stuff. So that's the attack they describe in the paper. What's interesting is that this attack actually doesn't work anymore.

So we actually discussed this a couple of lectures back. So this attack that the paper describes is the browser history sniffing attack. So as we discussed a couple of lectures ago, JavaScript code now does not expose correct link colors basically to JavaScript . And it's precisely to prevent these types of attacks. So that particular part of the paper is outdated.

AUDIENCE: What does it point to that browsers now also show links as purple in private browsing mode

and turn blue again when you exit.

PROFESSOR: Yeah, it's a bit weird, yeah. They implemented that attack-- the defense-- I think before a lot of the private browsers like a popware. So now they do this additional thing too. The long story short, the attack they describe in the paper doesn't work because of some of these browsers sniffing defenses. But you can still imagine that there may be ways for the web attacker to figure out if you are using private browsing mode.

So for example, when you do private browsing mode, any cookies that you got from public mode should not be sent during private mode. So in other words, if I go to Amazon.com in public mode, I generate some cookies. Then I go to Amazon.com in private browsing mode. When I contact Amazon.com in private mode, those public mode cookies should not be sent. That can actually act as the sign to the web attacker that you actually are using private mode.

AUDIENCE: This is also now you're using the canvass in both of these events, right? So you need to know the IP address.

PROFESSOR: Yeah, that's right.

AUDIENCE: So that link that you were targeting with the link color would be on a per IP basis. And you would have to rely that the user first visited it as a public mode, and you protect it.

PROFESSOR: Ah, so the link-- so the link attack you can actually do in the context of a single page. So imagine that I, the web attacker, construct single page. I, the attacker, have JavaScript that creates an iframe to foo.com like this. So that iframe will load the contents of that page. And then I, the attacker, in the parent frame can then create a link element and then try to look at the color. This worked four years ago.

So in that case, it doesn't rely on the user having explicitly visited that iframe page at all because I, the attacker, can create that in the context of the page. I have gotten [INAUDIBLE]. Any other questions? So yeah, so you can maybe think about how cookies can reveal public and private browsing modes and things like that.

So one thing we might think about is how we can provide a stronger privacy guarantee for private browsers? And for the sake of this discussion, let's just ignore IP addresses for now because as we'll discuss next lecture, we can used Tor to maybe help with some of the privacy of IP addresses. So one thing you can imagine doing is you can imagine using VMs in some

way to help provide stronger private browsing guaranteed-- so VM level privacy.

And so the basic idea is that you want to run each private session inside of a separate VM. And then when the user is done with that-- so is finished with the private browsing session, you basically delete VM after that session is done. So what's the advantage of this?

Well, what's nice about this is presumably you can get some stronger guarantees about what privacy properties you can provide to the user because, presumably, the VM has a pretty clean interface to the I/O path of the underlying post-OS. So you can imagine that maybe you combine this VMs into let's say some type of a secure swap solution like Open BSD has-- give us another encrypted disk type thing.

So you can imagine, OK, we have a very clean separation of VM up here and all the I/Os that are generated down here. And so that gives you stronger guarantees than what you can get from the browser, which wasn't designed from the ground up to think very carefully about all the I/O paths and what secrets might leak when it was in storage.

So yes, this provides what's nice about this-- strong guarantees. And, also, what's nice is it doesn't require any changes to your applications-- that is to say to the browser. You take your browser, put it inside one of these VMs-- then everything gets better all magically. It's not location change. So what's bad about this-- I'll use an unhappy face to demonstrate that. So what's bad is first of all, it's heavyweight.

And by heavyweight, I mean that time you want to spin up one of these private browsing sessions, you have to spin up a whole VM. And that can actually be pretty painful. So perhaps users are going to get upset because it's going to take them long time now to launch these private browsing sessions.

And the other problems to is this solution actually has bad usability. And the reason I say that is because now it's actually difficult for users to do things like take files that they've saved in private browsing mode and then take them to the rest of their computer-- any bookmarks that they generate during private browsing mode that who they actually do want to persist will be difficult to get those at the end. It can be done. But there's a lot of friction here. So that's the bummer.

So another thing that you might imagine doing is something that looks like approach number one. But we actually implement it inside of the OS themselves instead of in a virtual machine.

So the basic idea here is that you can imagine that each process could potentially run in a privacy domain.

So basically, the privacy domain access the collection of OS global resources that process uses. And so the OS tracks all that kind of stuff. And then once the process dies, essentially the OS goes through and looks at all the things that are in that privacy domain set. And then purely deallocate all those resources. And so the advantage of this over the VM is that it is lighter weight because if you think about it, the VM is essentially agnostic to all the OS state and all the application state that is actually being used to run.

So the result-- it probably does more work than the OS would have to do because the OS presumably knows all the points at which the private browser would be touching I/O, and talk to the network, and stuff like that. So maybe it even knows things like you can actually clear the DNS cache selectively, for example.

So you can imagine that it's much easier to spin these things up-- these privacy domains-- then to tear them down. However, the sad thing, at least with respect to the virtual machine solution, is that it's harder to get this right. So I just described the VM approach as being headway because it's essentially agnostic to everything that's running inside the container.

But what's nice about that is that allows the VM approach to only focus on a few low-level interfaces. And it can focus on those things. For example, the interface the VM uses to write to disk, then you can have high confidence that it's actually managed to contain everything.

Whereas with the OS-- if you think the OS is going to interpose on individual files with system interfaces-- perhaps individual network interfaces and stuff like that-- it's much more complicated to find all of those points at which data can leak if you're going to do that at the OS level. So does that all make sense? Why is this physics everywhere? Ah, god, I'm being tested.

Those are basically some approaches we can use to provide potentially stronger privacy guarantees than what's implemented in private browsers right now. So one question you might have is can we still be an anonymized user if the browser-- sorry, if the user is employing one of these more powerful solutions-- if the user is surfing through VM or surfing one of these privacy domains in the OS-- can we still figure out who they are? And the answer is, yes.

So maybe the VM is unique for some reason. And so similar to how we were able to fingerprint

browsers using that Panopticlick website, maybe there's something unique about the way that the VM would be set up that allows to fingerprint it. And it may in fact be the case that maybe the virtual machine monitor or the OS itself is unique in some ways. That would allow a web attacker to figure out who the user was.

And so one cute example of this is TCP fingerprinting. So what's the big idea behind this. So as it turns out, the specification for the TCP protocol actually allows some of the parameters for the protocol to be set by the implementation of the protocol. So, for example, TCP allows implementers to choose things like initial packet size-- the things that are sent out the first part of the TCP connection-- it allows implementers to choose things like that initial time to live in those packets.

And so you can imagine, and in fact, you don't have to imagine that this is actually the truth. You can get off-the shelf tools like InMap, for example, that they actually can tell what operating system you're running with high probability just by sending you packets. They'll send these very carefully crafted packets. And they will look and see things like here's what the TTL was or here's what the packet size distribution was-- here's what the TTP sequence number was. And they basically have a database to fingerprint.

And they say, OK, if the return packet has this, this, and this characteristic, then the table says that you're probably running for some reason Solaris. You're running Mac. You're running Windows or whatever. So even if we use one of these stronger approaches for private browsing with a VM or an OS, you still may be able to run one of those TCP fingerprinting attacks and learn a lot about a particular user.

And one thing that's also interesting to note is that even if we use one of these more powerful techniques to try to protect the user, the user is still shared across both the public and the private browsing session. Still uses-- visibly using the machine. So why is it interesting? Well, it's interesting because you yourself by way that you use computers, may leak information about yourself.

So, for example, as it turns out, users have unique keystroke timing. So if I look at-- if I give everyone in this class the same thing to type in -- the quick, brown fox-- whatever that nonsense is-- and I actually look at the inter-key press timing, we'll all have these unique distributions that can potentially be used to fingerprint us.

Another thing that's interesting is that users have unique writing styles. So there's this branch

of security that is called stylography. And the basic idea here is to figure out if I am an attacker, can I figure out who you are just by looking at writing samples from you? So imagine that for whatever reason you're hanging out on 4chan-- don't hang out on 4chan-- and I want to figure out if you've actually, in fact, been hanging out on 4chan.

So perhaps what I can do is I can look at a bunch of different posts from 4chan. Maybe I can cluster those posts into sets of posts that I think look stylistically the same. And then what I can do is I can find things that you've written publicly where you're actually attributed as the author. I'll look at you homework assignments or papers that you've written or things like that. And I'll see do you match any of these clusters from these 4chan comments.

And if so, them maybe I can say maybe send you a stern note. Talk to the parents that their kid has gone off the beaten path. Get off of 4chan. So the reason is I would like to look at this thing called stylography. It's actually quite interesting. Does anyone have any questions about that? Excellent.

So we discuss how we might be able to use VM or modified operating systems to provide private browsing support. And so you might wonder, OK, well, then why don't browsers require users to do one of these things-- to have one of these tricked out VMs or tricked out OSes? So why do browsers take it upon themselves to implement all this stuff?

And so the main reason is deployability. So in fact, browser vendors typically do not want to ask their users to do anything special to use the browser besides install the browser binary itself. This is similar to the motivation of the native client. So if Google wants to add these cool future to end users' computers. But it doesn't want to force users to install some special version of Windows or Linux or whatever. So Google basically says, we'll take care of this ourselves.

Then another reason is actually usability. So a lot of these VM and OS-level solutions in private browsing-- as we've discussed, they make it more difficult for users to persist state from private browsing sessions that they do actually want to persist like downloading files like bookmarks they create and things like that.

So basically the browser vendors say, well, if we implement private browsing modes ourselves, we can actually allow users to do those things. We can allow users to take downloaded files from private browsing mode and take them to the rest of the machine. So that seems nice at first. But note that, of course, that allowing users to export some type of private state actually

opens up a lot of security vulnerabilities. It makes it very difficult to analyze security properties that result in private browsing modes actually provide.

And so in the paper, they try to characterize the different types of browser states that can be modified and how current private browsing modes actually handle the modifications at stake. So the paper describes this taxonomy of browser state changes. And so there are four things in the taxonomy. So one type of state change is initiated by the website itself. And there's no user interaction.

And so examples of this type of state change think about stuff like when a cookie gets set-- when something gets added to the address history of the browser-- maybe within a browser cache or something. And so from this type of state, basically, private browsing mode says this state is a private browsing mode session.

But it basically is going to be destroyed when that private browsing session concludes. And so the intuition behind this is that because there is no user interaction in creating this state, then perhaps the right thing for the browser to do is assume that the user wouldn't want that to persist. So another type of browser state change is initiated by the website that the user is visiting. But there is some type of user interaction involved in the state change.

So an example of this might be the user installs client certificate or maybe there's a safe password. So the user tries to login to something. And the browser says very helpfully would you like to save this password? And then if the users says, yes, then these types of things, say passwords, can actually be used outside of the private browsing mode.

And so it's a little bit unclear in principle what the policy for this should be. So what ends up happening in practice is that browsers typically allow statements in this category that set in private browsing modes to persist outside of that private browsing mode under the intuition that the user did have to say yes or no. If the user said, yes, then maybe the user is smart enough to understand that they save some password for some unsavory site and someone comes on later and figures that out, that's the users fault-- not the browsers fault.

So it's a little unclear what the best policy is here. But in practice, this type of state change is allowed to persist outside of private browsing mode.

So there's another type of state change, which is purely initiated by the user. And so here you can think about things like setting a bookmark or maybe downloading files. And so the story for

this state is similar to the story for the state up here. So basically because the user was explicitly involved in the creation of that state. Private browsing modes typically say, OK, it's OK to persist these types of changes to the outside world outside of private browsing mode.

Then there's some sets of state which are actually unrelated to any particular session at all. So this is stuff, for example, like an update to the browser itself-- the actual binary that constitutes the browser. And so the way the browser vendors think about this state is this state is essentially assumed to be part of the single, global state that's available to both public and private browsing modes.

And so eventually, if you look at it, there's actually quite a lot of states that will actually potentially leak outside of private browsing mode, particularly if there's user volition involved. So it's interesting to think about is this the right trade-off between security and privacy? So what's interesting is that-- so the paper actually says that it's difficult to sort of prevent a local attacker from detecting whether or not you've been using private browsing mode. And the paper was a little bit vague about why this might be the case.

So one reason why this might be the case is because some of this state that actually leaks from private browsing mode to public browsing mode, essentially it can actually contain hints the state was generated in private browsing mode. So for example, on Firefox and Chrome, when you generate a bookmark in private browsing mode, that bookmark has a bunch of metadata with it.

So for example, the time that it was visited and things like that. So in many cases, that metadata will be set to zero or some null value if that bookmark was generated inside of a private browsing mode. So then later on if someone controls your machine, and they look at your bookmark information-- if they see this metadata set to this zero and null value, they can say, aha, that bookmark was probably generated in private browsing mode.

So one thing to think about is typically we talk about browser security. We talk about, OK, what can people do with JavaScript or HTML or CSS. One thing you might want to think about is, well, what can people do with plug-ins or extensions? So in the context of private browsing, plug-ins and extensions are quite interesting because they're not constrained in most cases by the same origin policy. They can constrain stuff like JavaScript.

And what's interesting is that these extensions and plug-ins typically run with very high authority. Loosely speaking, you can think of them as like kernel modules. They implement

new features directly inside the browsers themselves. And so that's a little bit problematic because these plug-ins and extensions are often developed by someone who is not the actual browser vendor.

So what that means is that someone is trying to do something nice and provide you with this nice value add in this browser plug in or extension. But that implementor might not fully understand the context, the security context, in which that extension runs. So that extension may not implement private browsing mode semantics. Or it may try to implement it to do it in a bad way.

And so as I'll describe in a couple of minutes, that's actually bad from the security perspective because that means if we add some of these new plug-ins or extensions, you now can't strongly reason about what the resulting [INAUDIBLE] are. Now, one thing that's nice is that plug-ins are actually probably going the way of dinosaurs. So as you probably know, HTML5 adds all these new features like the audio tag and the videos tag, and stuff like that.

And so a lot of these new features were designed to allow people to get away from plug-ins-- to get away from Java-- to get away from Flash . So when people in the past wanted do things like have rich 2D or 3D graphics, they'd have to do something like Java or Flash. Now they can use things like Web GL. They can used things like the canvass tag.

So probably plug-ins are going away. In fact, the IE team for example, has said that in a couple years they don't think anybody's going to be using plug-ins whatsoever. It's all going to be HTML5 type stuff. In fact, if you go to YouTube-- I don't know if you've noticed. But a lot of times if you go to the video, the video is actually using-- it's called an HTML5 player. They've gone away from their standard plugin-based one.

So that's very interesting. You can already see sites trying to move towards this new plug-in world. However, extensions are probably here to stay for at least the foreseeable future. So it's still important to get those right. So, yeah, the last thing that I wanted to discuss is a paper was written in 2010-- that's four years ago.

So you might think to yourself what's changed about private browsing? And so at a high level, private browsing mode is still tricky to get right. And the reason why it's tricky to get right-- a couple of reasons. So first of all, because the browser [INAUDIBLE] is still growing because of things like this HTML5 stuff.

The interface, which needs to be secure with respect to private browsing mode, that frontier is always getting bigger. And also a lot of times developers-- they are more focused on to adding cool, new features. And then the privacy implications get taken up later on. And so in practice, it is still tricky to produce a private browsing mode which catches all potential data leaks.

So one example, there was a Firefox bug fix from January, 2014. And the basic idea is there is this extension-- it's called pdf.js is basically a way to look at PDF files using pure HTML5 interfaces. And so as it turns out, this extension was allowing public mode cookies to leak when it was being used in private browsing mode.

The idea is that let's say that you visit some websites in public mode. You want to download some PDF. Maybe you get some cookie that comes back. You come back in private browsing mode. You want to view another PDF from that site. And then pdf.js is actually sending those public mode cookies along with any private mode things that were set.

And so in the lecture notes, I actually have a link to the bugzilla discussion about the particular bug. So the fix was actually quite simple once they realized this was the problem. Basically they just have to add a check that says morally speaking, am I in private browsing mode? If so, do some things-- and one of those things is not from the cookies.

So the fix here is actually quite simple. But the challenge was that once again, people added this cool, new extension. But it hadn't really crossed their mind to do this full, invasive audit. And say where are all the places at which private browsing with semantics might be impacted by this particular plug-in.

There's another interesting one too-- this is actually the discussion we had about 30 minutes ago about what happens if you have private tabs and public tabs where you open at the same time or very close to each other. There is actually a bug in Firefox. I think that's from-- let's see here-- yeah, 2011, which is still unfilled. And the basic idea is that if you go to a task in private browsing mode-- OK, you go do some stuff. You then close that tab.

You then open a new public mode tab. And you go to about:memory. So as you probably know, a browser is defined as fake URLs and telling information about how the browser works. So you go to the private tab, close it up, then go to about:memory. This is going to tell you information about all the objects that Firefox has allocated.

So what would happen is that window objects are typically deallocated-- they are [INAUDIBLE]

in Firefox. So what ends up happening is that when you open up that new public mode tab, go to about:memory you can actually find information still about that private mode window such as things like a URL, for example, that will tell you how much memory to allocate and all that kind of stuff. And it's all in the plain text.

And so that's an example of how these very subtle interfaces and browsers that can actually leak a lot of information. And so it's very interesting. If you look at the bugzilla discussion, it's actually pretty interesting to see how these problems get resolved in real life. And I put a link it so there is a message that this book was deprioritized when it became clear that the potential solution was more involved than originally anticipated.

So that's a pretty long discussion about how do we fix this. And it involved changing the way that garbage collection is done. And it's very tricky because if you invoke it too often then it gets performance. So there's this long discussion about this. So they said, "It was deprioritized when it was clear the solution was more involved than anticipated." And then in response, a developer said, "That is very sad to hear. This could pretty much defeat the purpose of things like session store for getting about closed private windows."

So the developers about this stuff. Like in the case of the session store, this is storage feature for HTML5-- they had gone to a lot of trouble to make it delete things that belong to these closed private windows. But, basically, what this bug did-- it still-- it basically still left information about that stuff sitting around in memory somewhere.

So long story short, it's still very difficult to get private browsing right. And in fact, there are actually off-the-shelf forensics tools that you can download that will actually look for evidence of both public and private browsing modes. So if you're an attacker, you don't have to roll your own custom tool.

There's this one they call Magnet. I think it's an internet evidence finder. You just go get this thing. It'll do things like look through your page file for RAM artifacts. And it will give you a very nice GUI. It'll say here are the images I found. Here are the URLs. So in practice, these private browsing modes still do leak some information. All right, so next section, we'll talk about Tor.