All right.

So just a couple of announcements real quick.

Can you guys turn this down a little bit?

So, for tomorrow, you guys should all be ready to give these short presentations on your design projects in recitation.

We posted a one-page guide talking about what it is that you should prepare for next time in class.

These are short presentation.

You shouldn't need to put too much time into getting them together, but please look at the Web page if you're not sure of what it is that you're supposed to be doing in class.

Today what we're going to do is wrap up our discussion of security.

And what I want to do is start off with a fun little diversion which is over the past couple of times that you guys were in class we captured a log of all of the network traffic that was going on while people were in class on their laptops.

And, for the most part, it turned out that you guys were pretty boring.

Nobody was actually doing anything exciting, but there were a couple of interesting lessons.

In case you're curious as to what we actually did, we ran this TCP dump command.

The pseudo command just says run this command as though you are a super user, as though you are SU.

And then TCP dump just says dump out all the traffic on the specific network interface.

In this case, it is dumping out all the traffic on EN1 which on this computer is the wireless network.

So we are just dumping out all of the packets that were sent out over EM1. And what you see here is the yellow part is the sort of header of the packet saying this is an IP packet and specifying who the destination and recipient of this packet are.

And then there is some body text, some sort of payload that is associated with this packet.

In this case, this first packet is some binary data.

We cannot make any sense of it, but if you look at some of these other packets, for example, this is a packet which is an HTTP request packet.

What you see is that there, in fact, is English text.

The HTTP protocol uses English text to transmit information, and so this is specifying some information about this is the actual HTTP request going out.

In this case, it was going out from this laptop.

I just set this up as to illustrate what TCP dump does.

We ran a command like this while you guys where in class and captured a trace of what you were doing.

And we ran a little Perl script that went through this trace of packets and sucked out all the Web pages that you guys had been looking at.

And so I have pictures of some of the Web pages.

A lot of it, as I said, is pretty boring.

There is somebody who is looking to buy a micro drive so they searched for Micro Center cruiser USB high-speed on Google.

We saw this page coming back from Google which was the search request.

And, in fact, you sort of poke around some more and see this person actually found the Micro Center Website and went to Micro Center and apparently was trying to purchase this thing.

There were also a couple of people looking for movies, people who were looking for a movie to see.

In fact, we actually saw this on both days.

It wasn't just one day that somebody was looking for movies, but somebody keeps coming to class and looking for movies.

The next one, though, is much more interesting.

There are a couple of Web pages where somebody is looking at a tool called Gaim-Encryption. Gaim is the open

source AOL Instant Messenger client, and Gaim-Encryption is a version of the open source Gaim client that encrypts data that gets transmitted over the network.

And the way it works is when you install this tool it generates you a public/private key pair.

And then, when you have a chat session with somebody else who is also using this Gaim-Encryption tool, it sends that person your public key and then it uses public/private key encryption and signing in order to protect information that gets transmitted over the network.

So there was somebody who, during class, was looking at this tool.

We see they go to the project page, sourceforge, where this tool is hosted.

And they are clicking around looking at what the tool does and reading about it.

And then you see that they download this tool.

And, in fact, after that, at some point, there is a whole lot of AOL Instant Messenger traffic that is going out on the network.

I am just surmising that this traffic actually belongs to this person who downloaded this tool.

And you can see that, in fact, the tool appears to be working.

So this traffic is not intelligible.

And, in other cases, when you see this tool actually goes across the network in plain text, the Gaim normally runs in plain text and you can just see the entire transcript of the chat happening if you run this tool.

There is a very interesting security lesson here which is that sometime later, in this particular trace of a chat, we see this same person.

Person one, their name is transmitted in plain text.

You see this same person even when using the Gaim tool.

You don't see their messages in plain text, but you see this person now actually is talking, having a conversation.

And they sort of say oh, I download this Gaim tool, now I can talk encrypted to other people.

And yet there is this conversation where it is not encrypted, and this sort of dialogue goes on.

They are talking about somebody who is in class and say why doesn't this person go to class anymore?

And they are having this little chat session back and forth about why this person isn't in class.

And the dialogue goes on talking about their design project and they're worried about it.

It is sort of a nice example of a security problem or a potential security problem, which is that this person actually did something that was sort of nice.

They came to security class, they are learning about security in class, they go download this tool and think, OK, I've got this tool installed and now my conversations are protected.

And then, as soon as they try and talk to somebody who isn't also running this version of the tool, their conversations are no longer protected anymore.

This is sort of a nice example of why security is actually a really hard thing to get right.

Because this person may have even believed that this person they were talking to had this tool installed as well, but you sort of don't know that it is not working until somebody points out that it is not working, somebody violates, goes in and is able to access this on unprotected data.

This is a nice example of why security is hard and why sort of providing protection is essentially a negative role.

You never really know that you are protected until somebody points out that you are not protected.

That was sort of a little lesson about security, and I hope that it sort of pointing out a reason in which why it is that security is something that is both difficult and something that you might want to be conscious of.

Often times when you're sitting there using your laptop, you have this perception that is my information I am sending, nobody is going to be able to overhear this information.

It is going so fast and being transmitted so instantly that, of course, I am private.

You feel very sort of isolated when you're having this communication until somebody points out that it is actually very easy, I mean this is a tool that is standard on all Linux distributions.

And it is one line that we type into our computer to get this information.

It is actually very easy to do this, so it is something that when you are transmitting sensitive information out over the network over an email or in an Instant Messenger you might want to actually think about whether or not you want to be sending this information out in the way that you're sending it out.

What I want to do now is come back to something we were talking about last time, finish up our discussion a little bit of authorization and then talk about how we can start thinking about designing protocols that actually provide -- How we can reason about these security protocols that we are using.

In particular, how we can reason about authentication protocols within networks.

If you remember last time, we talked about this idea of building up a secure communication channel and then we talked about doing authorization on top of that channel.

Remember, authorization is this process of determining whether a given user should have access to a particular resource that they want access to.

We talked about two main techniques for doing this, ACLs and tickets.

An ACL is an access control list which is just a list of users who are authorized to access a particular resource.

And so the way typically an access control list works is that the system first authenticates the user to determine what their identify is and then it checks to see whether that identity is actually on this access control list and the user is authorized to access this particular resource they want access to.

Now, ticket has a slightly different flavor which is typically if a user presents a ticket you do not need to actually check and see what their identity is.

Simply having the ticket is sufficient to allow somebody to have access to a resource.

The analogy here is very similar to a physical ticket in the real world.

You go to a concert, you hand them a ticket and they let you into the concert without checking your ID.

A common example of a ticket in the real world is on the Web you get these cookies which are like tickets.

And the reason you use cookies instead of checking access control list every time is that when you have a particular cookie the system doesn't have to re-authenticate you in order to allow you to have access to the system every time you revisit a page.

I have a cookie for Amazon dot com on my computer when I am in the process of purchasing some item from Amazon dot com that identified my identity and my shopping cart and I don't have to sort of re-log in every time I request a new secure page from Amazon.

Specifically, what we are talking about is the Web.

And we had a browser and a Web server.

And the browser and the Web server were talking over one of these secure communication channels.

And we said this channel has both been authenticated and is confidential.

Authorization, we said, happens where there is some guard process in charge.

This is our guard.

And it sits in front of the services on the system.

When a request comes in to access a service, the Web server passes that request onto the guard module which does the authorization.

And then, if the request is authorized, it goes ahead and passes it onto the service.

We talked about this notion of how we establish a secure communication channel.

Typically, we do that by exchanging, one way that we talk about doing this is where B, for example, would discover W's public key somehow and then use that public key in order to exchange with W a shared secret key that they could use to encrypt all their communication.

I will show that protocol again in a minute, but the question, and the question that we are really going to get at today, we want to focus in more on this question of how it is that B actually discovers W's public key in a public key system and how it is that B can convince itself that, in fact, it is actually talking with this service W.

If B wants to have a conversation with Amazon dot com, how does it actually convince itself that this public key that it has, in fact, belongs to Amazon dot com and that it is actually having a conversation with Amazon dot com instead of with somebody else who is pretending to be Amazon dot com or has lied about Amazon dot com's public key so that it can overhear Amazon dot com's traffic.

We are going to focus in more on that now.

Let's look at this simplified version of an authentication protocol.

This is going to be like the Denning-Sacco Protocol that I showed with the broken example or like SSL, for example, on the Internet also sort of works roughly in this way.

We've got our browser, we've got our Web server and we've got some certificate authority.

This is the kind of picture we drew before.

We said the browser sends to the certificate authority a request for, say, the public key of W.

The certificate authority sends a message back with has the certificate for B and the certificate for W.

And these are simply signed things that the certificate authority has signed that include the public key for B and W in them, so that is all certificates are.

And then B sends a message to W.

And there is some trickiness in getting this message right, but this message basically is a signed and encrypted message that tells W that it would like to initiate a conversation with it.

And perhaps it proposes a shared key that it can use in that conversation.

It is a little bit tricky to actually figure out and make sure we get the format of that message right, and that was there is bug with the Denning-Sacco Protocol, but you get the idea.

So W then sends a message back saying perhaps asking user to go ahead and trying to authorize the user saying, OK, I agree that that is the key, now what is your name and what is your password, for example.

And then, once it has exchanged that name and password, B is allowed to log into the system and access whatever these resources B should have access to on the server.

The question that I want to focus on is this question about how B actually decides the certificate authority that it should trust.

How does B initially establish the sort of authenticity of the certificate authority?

When we talked about authentication before, we presented this model that was sort of, well, you are going to decide that you trust somebody by, for example, having a conversation with the.

I am going to run into you in the hall and you're going to say my public key is X, and then I am going to be able to have a conversation with you.

But, clearly, that is not what happens in the case of the Internet.

For example, on the Internet, I do not have to call up Amazon dot com or call up my certificate authority and get

the certificate authority's public key from them.

Somehow I have already gotten this thing.

And so we sort of want to think about how it is that we can actually exchange these keys and how it is that these systems can actually decide that they trust each other and that they are willing to authenticate each other.

To do that, what we are going to do is look at something called authentication logic.

The specific version of authentication logic we are going to talk about is something called BAN logic.

B, A and N just stand for the names of the three people who proposed this thing so it is not a specific acronym you need to know.

And so what authentication logic is going to do is allow us to reason about these protocols for deciding whether or not we actually trust something like a given certificate authority.

Just to think about this a little bit more, let's look at a specific example.

Suppose that you receive a message over the Internet.

Suppose this message says it is some message m and the message says sign is signed m with some key k, call it k sub A.

And suppose that the contents of message m are m equals give A your credit card number.

This might be a valid request.

We might be purchasing something from Amazon dot com.

In that case, you might think it might be OK, we might expect to receive a request like this.

So this thing was signed with kA and the message said give A your credit card number, but how do we actually know, you know, maybe A, in this case, is give to Amazon dot com and we know that we are talking to Amazon dot com.

And so if this, in fact, were an authentic message, we might be actually willing to give Amazon our credit card number.

How do we actually know, though, that this message, that kA is the key that belongs to Amazon dot com?

We want to answer the question how do we know that or trust that kA, we are going to say, speaks for A?

This notation speaks for just means that if we see something that, for example, is signed with kA that claims to be from A, we might actually trust, in fact, that this thing came from A.

And we want to sort of determine, we are going to try and answer the question of how do we trust this?

One way that we might trust this is true is if we had heard, for example, some message, let's call it m2 -- -- from somebody B that had said that -- Or this message said kA is A's key.

One way that we might start thinking about, one way that we might trust A is if we had heard from somebody else or heard from some other place that kA was A's key.

But now this is kind of a slippery slope because now you say the question, well, how did you determine that you trust B's key?

There is sort of this infinite process that you could end up going through.

And that doesn't sound like a very good idea.

One way in which sort of we might bottom out this recursion, we might end this sort of infinite process of collecting keys is if we actually sort of, in the example I gave you, heard from one person at some point suppose there is a B and a C and D and E and an F.

And then eventually we get to F.

F is our friend who we had talked to, and F had said my public key is this and here is the public keys for a few people that I know.

So you might trust F and you might believe that F knows a few other people.

And then those people who F knows might in turn know a few other people.

And eventually we might be able to sort of have this web of inner connected people all of whom we trust.

So this would be sort of a "web of trust".

We might be able to build up a web of trust like this, but you could imagine that these relationships are pretty complicated.

For example, B might trust a few people and C might trust a few people and D might trust a few people.

And, every time we're presented with some message, we need to have some sort of principled way of thinking about whether we have a set of these keys, for example, that actually allow us to decide whether we should accept this message or believe this message or not.

If we had a web of trust like this, we would need some way of going about validating or verifying that, in fact, these messages were messages we should trust.

That is what authentication logic is really all about.

This is the simplified band logic that is presented in the text.

It is on page 11-85. If you want to copy it down, I will try and leave time for you, but in case you want to just copy it out of the text you might not need to worry about copying down the exact logic here.

There are three rules in this simplified authentication logic and they use this notation, this speaks for notation, and they are also going to use some notation that says notation.

What this rule says is if A says B speaks for A then B speaks for A.

So this sounds like sort of an obvious statement.

Saying that A says something means that if we hear A say something that means that A told us and we believed that this, in fact, was A who told us this thing.

So, in order for A to say something, we have to actually believe this cannot be a message, for example, that we received over an untrusted network.

This has to be I sat next to A in the hall and A actually said B speaks for me.

In this case, B might be, for example, some key.

So A might say my public key is kA.

If that is true and, in fact, we believe that A is the person who said this to us then we might be able to infer from that that this key kA actually speaks for A.

This is what the first rule is and this is the base case of you have a trusted communication channel with communicating with A and you can use that trusted communication channel to actually build up some belief about, for example, a key representing somebody.

Now, the next rule is one that sort of says given that we have some belief about one of these things like, for example, if we know that B speaks for A, we know what A's public key is and we overhear B saying that A says X then we might believe, in fact, that A says X.

A simple example of this is suppose we get a message that has been signed by this key kA.

If we see that then we might, in fact, believe that kA actually says that A says m.

And we might believe that kA says this because we believe that this sort of sign primitive actually means that if we see something that was signed with kA's key that, in fact, that thing was generated by kA.

Nobody else could have generated a signed message that worked with kA.

We can actually now receive a message from A that has been signed using kA and we can, in fact, believe that A said something.

Now, the final rule that we have is sort of a transitivity rule which is a way of expanding this web of who is related to whom.

This says if B speaks for A and A speaks for C then what we're going to believe is that B speaks for C.

This is just a transitive relationship.

For example, if we know that kA speaks for A and we definitely overhear B saying A speaks for B, if B delegates and says that A can speak for me then we might believe that key kA also speaks for B.

This is a simple transitivity relationship.

The idea with these rules is that we are going to be able to apply these rules any time we have a conversation to decide, again, whether we actually trust the people who we are communicating with.

And you may have noticed, when I was talking about this, that there were a lot of statements like nobody can actually fake this protocol sign X.

We believe that nobody could forge this message that was signed with kA for example.

If we believe that that is true then we can say kA says this thing.

This authentication logic is full of a lot of these sort of like if we believe this thing is true kinds of assumptions that we are going to be making.

My example was kind of confusing.

In this example, B is actually C.

kA is B.

It says if we know that A speaks for B then we might be willing to believe that kA actually speaks for B, given that kA speaks for A.

So we have this transitivity kind of a relationship.

Sorry that the notation was a little bit confusing.

I agree.

The challenge here is that we want to try and make, any time we're using this authentication logic, in order to actually determine that somebody definitely said something we are going to have to make some set of assumptions about what we trust and what we believe.

And one of the things that is important, any time you are using this authentication logic, is to actually make these assumptions as explicit as you can.

Let's look at a set of assumptions that we are making here.

I said something like when I see a message that says sign m, kA.

And then I infer from that that A says m, assuming that I already know that kA speaks for A, if I have something like this and I make this inference that A says m given that kA speaks for A then that's all a sign thing.

I am making a set of assumptions when I do this.

I am making a set of assumption, in particular, about what this sign protocol does.

I am assuming, for example, that sign is actually secure in some way.

I am assuming that this signature is not forgeable, that somebody else could not have generated a signature unless they actually had access, for example, to the private key or to the private key of A.

This signature is not forgeable.

I am also assuming, for example, that any private keys are actually private.

I am assuming that, for example, A hasn't gone out and broadcasted the private key to everybody else in the world.

Because if A had done that then I wouldn't actually know that A actually said m.

It could have been anybody who said m.

I would be in trouble if that were the case.

Any time that I sort of take this and make this inference from seeing a sign like this then that suggests that I am sort of making these two assumptions.

And it is not that these assumptions are wrong or bad, it is just that it is good to think about exclusively what they are.

Assuming that the signature is forgeable well, we all know that cryptography can sometimes be broken.

It is not that cryptography is totally foolproof, but we may have a relatively high confidence that this cryptography is going to be difficult for somebody to break.

And, while assuming that private keys are actually private, we might sort of know A personally and believe that A is outright upstanding individual and we are going to trust that she hasn't gone and disseminated her key everywhere around the world because that wouldn't really be in her interest and we take her to be a trustworthy person.

This is a bit of a leap of faith that we are making about this.

But if we believe those two things then we can, in fact, infer that A says m when we see a message that is signed with kA.

Let's look at another example that works with this sort of public key cryptography that we talked about in this example with this authentication protocol here.

Suppose that A tells me my public key is kA pub and this is done over a secure communication channel like in-person communication, I might then infer that kA pub actually speaks for A.

Now, suppose I see a message that has been signed with kA private, I might say, and this is an inference, that kA private says that A says m.

The question we want to ask now is should we actually believe that A said m?

I saw a message, I know what A's public key is and I trust A's public key maybe and I saw this sign thing.

Now I need to think about do I believe now that A actually said m given that I saw a message that was signed with kA private?

There is something obviously that we need to do which is that we need to verify this message.

Suppose we go and we run this verify step on this message and this verify comes out to be true.

I verify using kA pub that, in fact, this message was appropriately signed.

Verify says yes, the signature on this message checks.

Now should I believe that that A actually said m?

That depends.

There are a couple of conditions, again, sort of set of assumptions that we are making.

One thing we are clearly doing is assuming that this crypto system is secure.

We are also believing that this verification step actually gives us, that by taking something that has been signed with a private key and then verifying it with the public key that that is as good as A actually saying something.

One thing clearly that depends on is that the crypto system is secure.

There are a couple of other assumptions that we are making.

One thing is that it definitely, again, requires that kA private has actually been kept secret.

And, furthermore, it requires that A didn't lie to us about what kA public was.

Because if A had given us a wrong kA public then somebody else could have signed this message and then we might try and verify the message using the lied about kA public and then we might be in trouble.

Again, this is similar to this example with just kA but just showing how it works with public and private key encryption.

It is this case that, again, we sort of need to always think about making our assumptions as explicit as possible.

And that is really what it's about.

Deciding that we trust, for example, the signature is true is analogous to deciding that we have to sort of trust that we trust the cryptography.

It is possible to apply this authentication logic in other environments as well that aren't necessarily based on cryptography.

For example, instead of trusting that the cryptographic system works, I might trust that a particular communication channel is secure.

For example, if I pick up the telephone and call you and give you some message, you might trust that it's me because, for example, you believe that it wouldn't be possible for somebody to fake my voice.

And you recognize my voice because you've come to lecture so many times.

That would be an example of a way in which you might decide that you trust that something is actually true, and that would be an assumption that you're making about somebody not being able to fake my voice.

The point of authentication logic is that it gives us a way to kind of reason about these kinds of deductions about whether or not we actually believe that somebody said something was true or not.

That is fine.

We still, though, at least when we're using public and private key cryptography, haven't exactly answered the question of how we go about establishing this initial trust step.

We said we might establish initial trust by -- At the bottom of this is we have to actually physically overhear somebody say something over a secure communication channel, we think.

We have to have some way to getting this first rule where A says that my, for example, public key is something so we can learn A's public key so that then we can bootstrap communication with A.

We haven't yet answered the question of how we actually exchange these keys except for by this one method that we talked about, this web of trust method.

There is this question about establishing some initial trust.

One way we might do this is using this web of trust approach where I meet a friend, that friend tells me about some people who he or she trusts, and then those friends' friends' in term know a few people and eventually we establish communication with everybody.

There are some computer systems that work this way.

In particular, there is a system called PGP for pretty good privacy which is an email-based privacy encryption system that works this way.

And there are, in fact, websites where sort of the idea is you find somebody else who has used this system, you share and you exchange public keys with them and then, once you have a few public keys, you can sort of build out this web of trust to everybody else by trusting people who your friends trust.

The problem is, one, clearly the Internet doesn't work this way so you don't have this kind of system that you're using on the Internet and, two, these kinds of systems are very difficult to set up and maintain and there ends up being some centralized administration.

You end up with having to have some sort of way, it becomes very difficult to actually discriminate these kinds of public keys in these kinds of systems.

The way this works is there is a centralized website.

There is a website for PGP that is a giant database of keys and you sort of try and discover people's keys by locking this web of trust, but it tends to be difficult to use.

Let's talk a little bit about how this actually works in the context of the Internet.

Suppose there is actually some person, P, who is sitting in front of this browser, and there are some questions that we might want to ask when we're thinking about deciding whether or not we trust this person, P. One question that the Web server might ask is does W know or trust person P, or how does it?

This is a commercial website like Amazon dot com.

Actually, the trust story turns out to not be that complicated.

Amazon dot com doesn't really care that you are who you claim to be, as long as you give them the money that they want.

You can go on and tell them that you are Oscar the Grouch, they don't care, as long as you give them the money that they want for whatever it is that you order.

That is all that matters to them.

So, in fact, what Amazon does is they don't actually know who you are exactly but they might decide that they trust

you because the credit card number is good and is verified by the credit card company.

When they try and charge your credit card they get the money and they are happy.

And there is this other question, though, about how does P trust W?

And the way that this works on the Internet is basically like the way that this authentication protocol that I have shown you here works, this is how SSL works.

And it turns out to be very subtle.

It can be quite hard to actually convince yourself that you should trust a given website if you actually start grilling down on it.

The way that it is going to decide that it trusts W is by checking with some certificate authority.

That is how it works on the Internet.

And there are some issues with it, which is how does the certificate authority authenticate W?

How does the certificate authority decide what W's key was and how do they securely exchange keys?

How did the user get the CA's public key?

And then what if somebody's private keys are stolen?

For example, W or the certificate authority's private keys are stolen.

What I want to do is just sort of show you, at a high level, how this actually works on the Internet.

I thought it might be instructive just to see an example of what is actually going on, on the Internet, and to kind of illustrate how some of these issues are and are not properly addressed by the current sort of SSL-based Internet architecture.

I have a couple of websites open here.

I have an Amazon dot com website and I have, in this case, an Apple developer website, both of which you will notice have HTTPS URLs associated with them.

In this browser, if I click on this little secure icon here, almost any browser would have a similar sort of feature, a little lock somewhere that when you are viewing an HTTPS page.

What I see is something that lists how it was that this site actually decided that this was a secure site that it should communicate with.

When that little lock appears it means that my browser has actually decided that it trusts this site.

We are at the point where we have already said we trust this site.

And the reason that we trust this site, it says, is because this site is WWW dot Amazon dot com and we have this certificate for this site that was issued by RSA Data Security Incorporated.

You might say who is RSA Data Security Incorporated?

I have never heard of these people.

I have never exchanged any information with them.

But they are the certificate authority.

That is the person who signed this thing.

I can look and sort of drill down on this.

And what you will see is some information about Amazon dot com.

This is the actual certificate itself.

This is some information about the certificate authority, including information about what algorithm the certificate authority has used to generate this certificate.

And then now what we have is this public key information, the actual public key that corresponds to Amazon dot com.

The certificate contains Amazon dot com's public key that was encrypted using the RSA algorithm.

And then there also is some information, this signature that goes with this public key.

This is the signature that was signed by this RSA corporation.

If I go to a different site like, for example, this Apple site over here, we see that this thing was actually signed by somebody else.

This certificate is signed by somebody called VeriSign Incorporated.

Again, you have no idea who VeriSign Incorporated is, but you will notice that these are two different certificate authorities.

It is not the case that there is one master certificate authority out there on the Internet that is in charge of everything.

On a Mac anyway, the same is true of most other operating systems, there is a way that we can go and look at the actual list of all the people who we trust are.

This is sort of an instructive thing to do.

To go and see who it is that you actually trust that is on this computer.

I can click on this certificate over here.

This is a list of all the certificates on this machine that I have and where I would accept things from.

You can see there are a very large number of certificates that this machine has installed.

These are not just certificates but actual certificate authorities.

People who I would actually trust to sign a communication with a given site to provide me with a certificate for, say, some Web server that I want to communicate with.

Now the question is so where did these things come from?

They came from one of two places.

Either I added them to the system myself.

In this case, I have, for example, an MIT certification authority.

If any of you have ever used MIT certificates like when you're logging onto Websys, in order to be able to access that system you had to add support for a given certificate to your browser.

This MIT certification authority is a certificate that came from MIT and this keychain thing which shows me on my certificates has this funny sort of message that says this certificate is not in the trusted root database.

The reason it says that is, this is a certificate for the MIT certification authority, I just got this certificate for the certification authority off the Web somewhere.

And I didn't do something that it wanted me to do to actually verify that this, in fact, was a legitimate certificate from the certificate authority.

Somebody could have put this thing on the Web and said this is a certificate for the MIT certificate authority and you should use it to discover certificates for anybody who claims to be coming from MIT dot edu.

They could have lied about it and then I would be in trouble.

It is possible that this is a fake certificate for the MIT certificate authority, but I sort of have made this trust, this assumption that it is not because I downloaded it from a website.

And I believe that, in fact, this is a valid certificate.

The other things you see here are this large number of certificates.

For example, one of the ones that we see, let's see if we can find RSA here, this RSA security.

Here I see RSA security.

And this thing actually says this certificate is valid.

Now there is this question about where did RSA security come from?

And the way that this works is that this browser and, in this case, the operating system itself has actually been shipped with a bunch of certificates from a bunch of different certificate authorities.

And so these have been installed by the operating system manufacturer.

And I am saying I trust Apple to have gotten the appropriate certificates and loaded them appropriately into the computer.

And any time you download a browser basically that browser already has a set of certificates for a set of certificate authorities already installed in it.

And that is where all of these base certificate authorities come from.

You can see that when you are using one of these things you have already placed quite a bit of trust and made a number of assumptions about who you trust and what you should trust just by using this system.

We have answered this question now about where did the user get the public key for the certificate authority?

Either they explicitly added it by downloading it from a website, for example, or it came with the browser.

What about this question how does the certificate authority authenticate an actual website?

How does Amazon dot com get a certificate from VeriSign?

The answer to this turns out to be that Amazon dot com probably paid VeriSign some money.

And it is not even clear you can say that, but in the case of VeriSign you are probably pretty sure that Amazon dot com paid them some money.

And you can probably go to the VeriSign website and see a list of things that they claimed to have done in sort of authenticating Amazon dot com.

They may have forced Amazon dot com to present them with some information that actually suggests that they are the company Amazon dot com or they own some right to the name Amazon dot com or that they are incorporated under the name Amazon dot com.

Basically, the certificate authority has some protocol that it uses to authenticate W.

And we don't know what that protocol is, but you're sort of implicitly trusting that all these certificate authorities that Apple has already approved for you, in fact, use some process that we will be reasonably assured of guarantying your privacy or guarantying that the websites that you are connecting to are, in fact, valid websites that you should feel comfortable giving your credit card number to.

You can see that this is a little bit dicey.

If feels a little bit like it is not clear that this is incredibly secure.

And then the last question is well, what if your private keys are stolen?

If your private keys are stolen in this particular example, the certificates that you have from Amazon dot com, if Amazon dot com's private keys are stolen and one of the certificate authority's private keys are stolen then you are in trouble.

These certificates typically have expiration dates associated with them, but these expiration dates are often like a year or more into the future.

And so you are going to continue to trust that certificate until this certificate expires in the future when you will go try and get a new one.

For that whole time that you trust that certificate somebody who had access to the private keys from that certificate authority or from that website would be able to pretend to be that certificate authority or that website.

So that might be problematic.

That is sort of the end of this little discussion about how certificate authorities work.

I hope you can see that this authentication logic is sort of a way that we can start to get at thinking about what the trust assumptions that we are making are when we are using these systems.

This wraps up our discussion of security.

We have one class session last.

And the last class session is actually going to be a guest lecturer.

It is going to be somebody talking about how law and computers and technology interact with each other.

We are kind of step back and talk much more about high-level pictures like policy and law next time, and hopefully it will give you a good wrap-up for the whole class of 6.033. We will see you next time.

Please come to class because there will be questions on the guest lecturer on the exam, so you want to make sure you don't miss it.