MITOCW | ocw-6.033-lec24

0:00:00 0:00:06 All right. So just a couple of 0:00:08.371 announcements real quick. Can you guys turn this down a 0:00:12.786 little bit? So, for tomorrow, 0:00:15.075 you guys should all be ready to give these short presentations 0:00:20.062 on your design projects in recitation. 0:00:23.088 We posted a one-page guide talking about what it is that 0:00:27.584 you should prepare for next time in class. 0:00:32 These are short presentation. You shouldn't need to put too 0:00:35.723 much time into getting them together, but please look at the 0:00:39.511 Web page if you're not sure of what it is that you're supposed 0:00:43.428 to be doing in class. Today what we're going to do is 0:00:46.766 wrap up our discussion of security. 0:00:48.949 And what I want to do is start off with a fun little diversion 0:00:52.865 which is over the past couple of times that you guys were in 0:00:56.653 class we captured a log of all of the network traffic that was 0:01:00.57 going on while people were in class on their laptops. 0:01:05 And, for the most part, it turned out that you guys 0:01:08.348 were pretty boring. Nobody was actually doing 0:01:11.295 anything exciting, but there were a couple of 0:01:14.242 interesting lessons. In case you're curious as to 0:01:17.457 what we actually did, we ran this TCP dump command. 0:01:20.806 The pseudo command just says run this command as though you 0:01:24.69 are a super user, as though you are SU. 0:01:27.235 And then TCP dump just says dump out all the traffic on the 0:01:31.12 specific network interface. In this case, 0:01:34.762 it is dumping out all the traffic on EN1 which on this 0:01:37.873 computer is the wireless network. 0:01:39.751 So we are just dumping out all of the packets that were sent 0:01:43.214 out over EM1. And what you see here is the 0:01:45.62 yellow part is the sort of header of the packet saying this 0:01:49.024 is an IP packet and specifying who the destination and 0:01:52.135 recipient of this packet are. And then there is some body 0:01:55.422 text, some sort of payload that is associated with this packet. 0:02:00 In this case, this first packet is some 0:02:02.527 binary data. We cannot make any sense of it, 0:02:05.388 but if you look at some of these other packets, 0:02:08.447 for example, this is a packet which is an 0:02:11.108 HTTP request packet. What you see is that there, 0:02:14.235 in fact, is English text. The HTTP protocol uses English 0:02:17.893 text to transmit information, and so this is specifying some 0:02:21.818 information about this is the actual HTTP request going out. 0:02:25.742 In this case, it was going out from this 0:02:28.337 laptop. I just set this up as to 0:02:31.599 illustrate what TCP dump does. We ran a command like this 0:02:35.333 while you guys where in class and captured a trace of what you 0:02:39.4 were doing. And we ran a little Perl script 0:02:42.199 that went through this trace of packets and sucked out all the 0:02:46.266 Web pages that you guys had been looking at. 0:02:49.133 And so I have pictures of some of the Web pages. 0:02:52.266 A lot of it, as I said, is pretty boring. 0:02:54.933 There is somebody who is looking to buy a micro drive so 0:02:58.599 they searched for Micro Center cruiser USB high-speed on 0:03:02.266 Google. We saw this page coming back 0:03:05.627 from Google which was the search request. 0:03:07.953 And, in fact, you sort of poke around some 0:03:10.337 more and see this person actually found the Micro Center 0:03:13.534 Website and went to Micro Center and apparently was trying to 0:03:17.023 purchase this thing. There were also a couple of 0:03:19.755 people looking for movies, people who were looking for a 0:03:22.953 movie to see. In fact, we actually saw this 0:03:25.395 on both days. It wasn't just one day that 0:03:27.72 somebody was looking for movies, but somebody keeps coming to 0:03:31.209 class and looking for movies. The next one, 0:03:35.151 though, is much more interesting. 0:03:37.987 There are a couple of Web pages where somebody is looking at a 0:03:43.392 tool called Gaim-Encryption. Gaim is the open source AOL 0:03:48.265 Instant Messenger client, and Gaim-Encryption is a 0:03:52.607 version of the open source Gaim client that encrypts data that 0:03:58.012 gets transmitted over the network. 0:04:02 And the way it works is when you install this tool it 0:04:04.968 generates you a public/private key pair. 0:04:07.194 And then, when you have a chat session with somebody else who 0:04:10.62 is also using this Gaim-Encryption tool, 0:04:12.846 it sends that person your public key and then it uses 0:04:15.814 public/private key encryption and signing in order to protect 0:04:19.24 information that gets transmitted over the network. 0:04:22.094 So there was somebody who, during class, 0:04:24.32 was looking at this tool. We see they go to the project 0:04:27.403 page, sourceforge, where this tool is hosted. 0:04:31 And they are clicking around looking at what the tool does 0:04:34.071 and reading about it. And then you see that they 0:04:36.604 download this tool. And, in fact, 0:04:38.329 after that, at some point, there is a whole lot of AOL 0:04:41.185 Instant Messenger traffic that is going out on the network. 0:04:44.311 I am just surmising that this traffic actually belongs to this 0:04:47.598 person who downloaded this tool. And you can see that, 0:04:50.455 in fact, the tool appears to be working. 0:04:52.556 So this traffic is not intelligible. 0:04:54.443 And, in other cases, when you see this tool actually 0:04:57.191 goes across the network in plain text, the Gaim normally runs in 0:05:00.586 plain text and you can just see the entire transcript of the 0:05:03.766 chat happening if you run this tool. 0:05:07 There is a very interesting security lesson here which is 0:05:10.538 that sometime later, in this particular trace of a 0:05:13.634 chat, we see this same person. Person one, their name is 0:05:17.109 transmitted in plain text. You see this same person even 0:05:20.585 when using the Gaim tool. You don't see their messages in 0:05:24.123 plain text, but you see this person now actually is talking, 0:05:27.851 having a conversation. And they sort of say oh, 0:05:31.402 I download this Gaim tool, now I can talk encrypted to 0:05:34.501 other people. And yet there is this 0:05:36.488 conversation where it is not encrypted, and this sort of 0:05:39.703 dialogue goes on. They are talking about somebody 0:05:42.509 who is in class and say why doesn't this person go to class 0:05:45.899 anymore? And they are having this little 0:05:48.179 chat session back and forth about why this person isn't in 0:05:51.511 class. And the dialogue goes on 0:05:53.265 talking about their design project and they're worried 0:05:56.363 about it. It is sort of a nice example of 0:05:59.592 a security problem or a potential security problem, 0:06:02.16 which is that this person actually did something that was 0:06:05.037 sort of nice. They came to security class, 0:06:07.143 they are learning about security in class, 0:06:09.25 they go download this tool and think, OK, I've got this tool 0:06:12.28 installed and now my conversations are protected. 0:06:14.746 And then, as soon as they try and talk to somebody who isn't

conversations are protected. Now, and then, as soon as they try and talk to somebody who isn't 0:06:17.777 also running this version of the tool, their conversations are no 0:06:21.065 longer protected anymore. This is sort of a nice example 0:06:23.89 of why security is actually a really hard thing to get right. 0:06:28 Because this person may have even believed that this person 0:06:31.459 they were talking to had this tool installed as well, 0:06:34.56 but you sort of don't know that it is not working until somebody 0:06:38.318 points out that it is not working, somebody violates, 0:06:41.419 goes in and is able to access this on unprotected data. 0:06:44.64 This is a nice example of why security is hard and why sort of 0:06:48.278 providing protection is essentially a negative role. 0:06:51.32 You never really know that you are protected until somebody 0:06:54.779 points out that you are not protected. 0:06:58 That was sort of a little lesson about security, 0:07:00.297 and I hope that it sort of pointing out a reason in which 0:07:03.035 why it is that security is something that is both difficult 0:07:05.871 and something that you might want to be conscious of. 0:07:08.413 Often times when you're sitting there using your laptop, 0:07:11.102 you have this perception that is my information I am sending, 0:07:14.035 nobody is going to be able to overhear this information. 0:07:16.724 It is going so fast and being transmitted so instantly that, 0:07:19.608 of course, I am private. You feel very sort of isolated 0:07:22.248 when you're having this communication until somebody 0:07:24.742 points out that it is actually very easy, I mean this is a tool 0:07:27.773 that is standard on all Linux distributions. 0:07:31 And it is one line that we type into our computer to get this 0:07:34.804 information. It is actually very easy to do 0:07:37.468 this, so it is something that when you are transmitting 0:07:40.892 sensitive information out over the network over an email or in 0:07:44.76 an Instant Messenger you might want to actually think about 0:07:48.439 whether or not you want to be sending this information out in 0:07:52.243 the way that you're sending it out. 0:07:54.399 What I want to do now is come back to something we were 0:07:57.824 talking about last time, finish up our discussion a 0:08:00.995 little bit of authorization and then talk about how we can start 0:08:04.99 thinking about designing protocols that actually provide 0:08:08.478 -- How we can reason about these 0:08:12.753 security protocols that we are using. 0:08:16.17 In particular, how we can reason about 0:08:19.683 authentication protocols within networks. 0:08:23.481 If you remember last time, we talked about this idea of 0:08:28.607 building up a secure communication channel and then 0:08:33.354 we talked about doing authorization on top of that 0:08:38.006 channel. Remember, authorization is this 0:08:42.348 process of determining whether a given user should have access to 0:08:47.196 a particular resource that they want access to. 0:08:50.681 We talked about two main techniques for doing this, 0:08:54.469 ACLs and tickets. An ACL is an access control 0:08:57.803 list which is just a list of users who are authorized to 0:09:01.969 access a particular resource. And so the way typically an 0:09:06.451 access control list works is that the system first 0:09:09.086 authenticates the user to determine what their identify is 0:09:12.15 and then it checks to see whether that identity is 0:09:14.784 actually on this access control list and the user is authorized 0:09:18.118 to access this particular resource they want access to. 0:09:21.021 Now, ticket has a slightly different flavor which is 0:09:23.763 typically if a user presents a ticket you do not need to 0:09:26.72 actually check and see what their identity is. 0:09:30 Simply having the ticket is sufficient to allow somebody to 0:09:33.258 have access to a resource. The analogy here is very 0:09:36.067 similar to a physical ticket in the real world. 0:09:38.651 You go to a concert, you hand them a ticket and they 0:09:41.516 let you into the concert without checking your ID. 0:09:44.269 A common example of a ticket in the real world is on the Web you 0:09:47.808 get these cookies which are like tickets. 0:09:50.056 And the reason you use cookies instead of checking access 0:09:53.202 control list every time is that when you have a particular 0:09:56.404 cookie the system doesn't have to re-authenticate you in order 0:09:59.831 to allow you to have access to the system every time you 0:10:02.921 revisit a page. I have a cookie for Amazon dot 0:10:07.918 com on my computer when I am in the process of purchasing some 0:10:13.852 item from Amazon dot com that identified my identity and my 0:10:19.494 shopping cart and I don't have to sort of re-log in every time 0:10:25.428 I request a new secure page from Amazon. 0:10:30 Specifically, what we are talking about is 0:10:34.712 the Web. And we had a browser and a Web 0:10:39.08 server. And the browser and the Web 0:10:42.988 server were talking over one of these secure communication 0:10:49.54 channels. And we said this channel has 0:10:53.793 both been authenticated and is confidential. 0:11:00 0:11:05 Authorization, we said, happens where there is 0:11:08.698 some guard process in charge. This is our guard. 0:11:12.561 And it sits in front of the services on the system. 0:11:16.671 When a request comes in to access a service, 0:11:20.205 the Web server passes that request onto the guard module 0:11:24.726 which does the authorization. And then, if the request is 0:11:29.328 authorized, it goes ahead and passes it onto the service. 0:11:35 0:11:40 We talked about this notion of how we establish a secure 0:11:44.399 communication channel. Typically, we do that by 0:11:48.08 exchanging, one way that we talk about doing this is where B, 0:11:52.879 for example, would discover W's public key 0:11:56.159 somehow and then use that public key in order to exchange with W 0:12:01.2 a shared secret key that they could use to encrypt all their 0:12:05.919 communication. I will show that protocol again 0:12:10.461 in a minute, but the question, and the question that we are 0:12:15.067 really going to get at today, we want to focus in more on 0:12:19.514 this question of how it is that B actually discovers W's public 0:12:24.438 key in a public key system and how it is that B can convince 0:12:29.123 itself that, in fact, it is actually talking with 0:12:32.935 this service W. If B wants to have a 0:12:36.152 conversation with Amazon dot com, how does it actually 0:12:39.205 convince itself that this public key that it has, 0:12:41.97 in fact, belongs to Amazon dot com and that it is actually 0:12:45.253 having a conversation with Amazon dot com instead of with 0:12:48.479 somebody else who is pretending to be Amazon dot com or has lied 0:12:52.108 about Amazon dot com's public key so that it can overhear 0:12:55.334 Amazon dot com's traffic. We are going to focus in more 0:12:58.444 on that now. Let's look at this simplified 0:13:02.036 version of an authentication protocol. 0:13:04.634 This is going to be like the Denning-Sacco Protocol that I 0:13:08.636 showed with the broken example or like SSL, for example, 0:13:12.497 on the Internet also sort of works roughly in this way. 0:13:16.289 We've got our browser, we've got our Web server and 0:13:19.799 we've got some certificate authority. 0:13:22.327 This is the kind of picture we drew before. 0:13:25.276 We said the browser sends to the certificate authority a 0:13:29.138 request for, say, the

public key of W. 0:13:33 The certificate authority sends a message back with has the 0:13:36.468 certificate for B and the certificate for W. 0:13:39.039 And these are simply signed things that the certificate 0:13:42.268 authority has signed that include the public key for B and 0:13:45.676 W in them, so that is all certificates are. 0:13:48.187 And then B sends a message to W. 0:13:50.041 And there is some trickiness in getting this message right, 0:13:53.509 but this message basically is a signed and encrypted message 0:13:57.037 that tells W that it would like to initiate a conversation with 0:14:00.744 it. And perhaps it proposes a 0:14:03.502 shared key that it can use in that conversation. 0:14:06.326 It is a little bit tricky to actually figure out and make 0:14:09.69 sure we get the format of that message right, 0:14:12.334 and that was there is bug with the Denning-Sacco Protocol, 0:14:15.759 but you get the idea. So W then sends a message back 0:14:18.824 saying perhaps asking user to go ahead and trying to authorize 0:14:22.489 the user saying, OK, I agree that that is the 0:14:25.133 key, now what is your name and what is your password, 0:14:28.257 for example. And then, once it has exchanged 0:14:32.347 that name and password, B is allowed to log into the 0:14:36.208 system and access whatever these resources B should have access 0:14:40.903 to on the server. The question that I want to 0:14:44.234 focus on is this question about how B actually decides the 0:14:48.55 certificate authority that it should trust. 0:14:51.731 How does B initially establish the sort of authenticity of the 0:14:56.349 certificate authority? When we talked about 0:15:00.093 authentication before, we presented this model that 0:15:02.827 was sort of, well, you are going to decide that 0:15:05.342 you trust somebody by, for example, 0:15:07.201 having a conversation with the. I am going to run into you in 0:15:10.481 the hall and you're going to say my public key is X, 0:15:13.269 and then I am going to be able to have a conversation with you. 0:15:16.659 But, clearly, that is not what happens in the 0:15:19.065 case of the Internet. For example, 0:15:20.869 on the Internet, I do not have to call up Amazon 0:15:23.439 dot com or call up my certificate authority and get 0:15:26.172 the certificate authority's public key from them. 0:15:30 Somehow I have already gotten this thing. 0:15:32.857 And so we sort of want to think about how it is that we can 0:15:37 actually exchange these keys and how it is that these systems can 0:15:41.571 actually decide that they trust each other and that they are 0:15:45.785 willing to authenticate each other. 0:15:48.214 To do that, what we are going to do is look at something 0:15:52.142 called authentication logic. 0:15:55 0:16:05 The specific version of authentication logic we are 0:16:08.485 going to talk about is something called BAN logic. 0:16:11.901 B, A and N just stand for the names of the three people who 0:16:15.944 proposed this thing so it is not a specific acronym you need to 0:16:20.266 know. And so what authentication 0:16:22.427 logic is going to do is allow us to reason about these protocols 0:16:26.819 for deciding whether or not we actually trust something like a 0:16:31.072 given certificate authority. Just to think about this a 0:16:36.703 little bit more, let's look at a specific 0:16:40.864 example. Suppose that you receive a 0:16:44.399 message over the Internet. Suppose this message says it is 0:16:50.327 some message m and the message says sign is signed m with some 0:16:56.672 key k, call it k sub A. And suppose that the contents 0:17:03.27 of message m are m equals give A your credit card number. 0:17:09.586 This might be a valid request. We might be purchasing 0:17:15.451 something from Amazon dot com. In that case, 0:17:20.3 you might think it might be OK, we might expect to receive a 0:17:26.954 request like this. So this thing was signed with 0:17:32.076 kA and the message said give A your credit card number, 0:17:35.941 but how do we actually know, you know, maybe A, 0:17:39.234 in this case, is give to Amazon dot com and 0:17:42.241 we know that we are talking to Amazon dot com. 0:17:45.463 And so if this, in fact, were an authentic 0:17:48.398 message, we might be actually willing to give Amazon our 0:17:52.335 credit card number. How do we actually know, 0:17:55.413 though, that this message, that kA is the key that belongs 0:17:59.494 to Amazon dot com? We want to answer the question 0:18:05.103 how do we know that or trust that kA, we are going to say, 0:18:11 speaks for A? This notation speaks for just 0:18:15.344 means that if we see something that, for example, 0:18:20.31 is signed with kA that claims to be from A, 0:18:24.655 we might actually trust, in fact, that this thing came 0:18:30.137 from A. And we want to sort of 0:18:34.304 determine, we are going to try and answer the question of how 0:18:40.59 do we trust this? One way that we might trust 0:18:45.2 this is true is if we had heard, for example, 0:18:49.809 some message, let's call it m2 -- 0:18:54 0:19:04 -- from somebody B that had said that -- 0:19:08 0:19:18 Or this message said kA is A's key. 0:19:20.391 One way that we might start thinking about, 0:19:23.346 one way that we might trust A is if we had heard from somebody 0:19:27.638 else or heard from some other place that kA was A's key. 0:19:32 But now this is kind of a slippery slope because now you 0:19:35.007 say the question, well, how did you determine 0:19:37.414 that you trust B's key? There is sort of this infinite 0:19:40.312 process that you could end up going through. 0:19:42.664 And that doesn't sound like a very good idea. 0:19:45.07 One way in which sort of we might bottom out this recursion, 0:19:48.296 we might end this sort of infinite process of collecting 0:19:51.304 keys is if we actually sort of, in the example I gave you, 0:19:54.421 heard from one person at some point suppose there is a B and a 0:19:57.757 C and D and E and an F. And then eventually we get to 0:20:02.076 F. F is our friend who we had 0:20:04.081 talked to, and F had said my public key is this and here is 0:20:08.233 the public keys for a few people that I know. 0:20:11.384 So you might trust F and you might believe that F knows a few 0:20:15.68 other people. And then those people who F 0:20:18.544 knows might in turn know a few other people. 0:20:21.622 And eventually we might be able to sort of have this web of 0:20:25.775 inner connected people all of whom we trust. 0:20:30 So this would be sort of a "web of trust". 0:20:32.748 We might be able to build up a web of trust like this, 0:20:36.301 but you could imagine that these relationships are pretty 0:20:40.055 complicated. For example, 0:20:41.664 B might trust a few people and C might trust a few people and D 0:20:45.821 might trust a few people. And, every time we're presented 0:20:49.575 with some message, we need to have some sort of 0:20:52.659 principled way of thinking about whether we have a set of these 0:20:56.815 keys, for example, that actually allow us to 0:20:59.698 decide whether we should accept this message or believe this 0:21:03.653 message or not. If we had a web of trust like 0:21:07.674 this, we would need some way of going about validating or 0:21:10.907 verifying that, in fact, these messages were 0:21:13.389 messages we should trust. That is what authentication 0:21:16.391 logic is really all about. This is the simplified band 0:21:19.451 logic that is presented in the text. 0:21:21.472 It is on page 11-85. If

about. This is the simplified BAN logic that is presented in the text. It's on page 77 so if you want to copy it down, 0:21:24.243 I will try and leave time for you, but in case you want to 0:21:27.534 just copy it out of the text you might not need to worry about 0:21:31.055 copying down the exact logic here. 0:21:34 There are three rules in this simplified authentication logic 0:21:39.016 and they use this notation, this speaks for notation, 0:21:43.364 and they are also going to use some notation that says 0:21:47.795 notation. What this rule says is if A 0:21:50.806 says B speaks for A then B speaks for A. 0:21:54.066 So this sounds like sort of an obvious statement. 0:21:59 Saying that A says something means that if we hear A say 0:22:03.298 something that means that A told us and we believed that this, 0:22:08.066 in fact, was A who told us this thing. 0:22:10.958 So, in order for A to say something, we have to actually 0:22:15.257 believe this cannot be a message, for example, 0:22:18.774 that we received over an untrusted network. 0:22:22.057 This has to be I sat next to A in the hall and A actually said 0:22:26.825 B speaks for me. In this case, 0:22:29.091 B might be, for example, some key. 0:22:33 So A might say my public key is kA. 0:22:35.746 If that is true and, in fact, we believe that A is 0:22:39.705 the person who said this to us then we might be able to infer 0:22:44.552 from that that this key kA actually speaks for A. 0:22:48.43 This is what the first rule is and this is the base case of you 0:22:53.438 have a trusted communication channel with communicating with 0:22:58.205 A and you can use that trusted communication channel to 0:23:02.567 actually build up some belief about, for example, 0:23:06.445 a key representing somebody. Now, the next rule is one that 0:23:12.462 sort of says given that we have some belief about one of these 0:23:17.47 things like, for example, if we know that B speaks for A, 0:23:22.067 we know what A's public key is and we overhear B saying that A 0:23:27.074 says X then we might believe, in fact, that A says X. 0:23:32 A simple example of this is suppose we get a message that 0:23:36.997 has been signed by this key kA. If we see that then we might, 0:23:42.351 in fact, believe that kA actually says that A says m. 0:23:46.992 And we might believe that kA says this because we believe 0:23:51.989 that this sort of sign primitive actually means that if we see 0:23:57.433 something that was signed with kA's key that, 0:24:01.359 in fact, that thing was generated by kA. 0:24:06 Nobody else could have generated a signed message that 0:24:10.729 worked with kA. We can actually now receive a 0:24:14.655 message from A that has been signed using kA and we can, 0:24:19.563 in fact, believe that A said something. 0:24:22.953 Now, the final rule that we have is sort of a transitivity 0:24:28.039 rule which is a way of expanding this web of who is related to 0:24:33.483 whom. This says if B speaks for A and 0:24:37.363 A speaks for C then what we're going to believe is that B 0:24:41.633 speaks for C. This is just a transitive 0:24:44.53 relationship. For example, 0:24:46.436 if we know that kA speaks for A and we definitely overhear B 0:24:50.935 saying A speaks for B, if B delegates and says that A 0:24:54.9 can speak for me then we might believe that key kA also speaks 0:24:59.551 for B. This is a simple transitivity 0:25:05.349 relationship. The idea with these rules is 0:25:11.5 that we are going to be able to apply these rules any time we 0:25:20.5 have a conversation to decide, again, whether we actually 0:25:28.9 trust the people who we are communicating with. 0:25:37 And you may have noticed, when I was talking about this, 0:25:43.004 that there were a lot of statements like nobody can 0:25:48.462 actually fake this protocol sign X. 0:25:52.174 We believe that nobody could forge this message that was 0:25:58.179 signed with kA for example. If we believe that that is true 0:26:03.848 then we can say kA says this thing. 0:26:05.876 This authentication logic is full of a lot of these sort of 0:26:09.334 like if we believe this thing is true kinds of assumptions that 0:26:13.032 we are going to be making. 0:26:15 0:26:25 My example was kind of confusing. 0:26:29.258 In this example, B is actually C. 0:26:33.517 kA is B. It says if we know that A 0:26:37.908 speaks for B then we might be willing to believe that kA 0:26:45.228 actually speaks for B, given that kA speaks for A. 0:26:51.749 So we have this transitivity kind of a relationship. 0:27:00 Sorry that the notation was a little bit confusing. 0:27:03.347 I agree. The challenge here is that we 0:27:05.824 want to try and make, any time we're using this 0:27:08.903 authentication logic, in order to actually determine 0:27:12.317 that somebody definitely said something we are going to have 0:27:16.267 to make some set of assumptions about what we trust and what we 0:27:20.418 believe. And one of the things that is 0:27:22.895 important, any time you are using this authentication logic, 0:27:26.845 is to actually make these assumptions as explicit as you 0:27:30.527 can. 0:27:32 0:27:42 Let's look at a set of assumptions that we are making 0:27:49.548 here. I said something like when I 0:27:54.338 see a message that says sign m, kA. 0:28:00 And then I infer from that that A says m, assuming that I 0:28:05.09 already know that kA speaks for A, if I have something like this 0:28:10.818 and I make this inference that A says m given that kA speaks for 0:28:16.545 A then that's all a sign thing. I am making a set of 0:28:21.181 assumptions when I do this. I am making a set of 0:28:25.454 assumption, in particular, about what this sign protocol 0:28:30.454 does. I am assuming, 0:28:33.293 for example, that sign is actually secure in 0:28:37.267 some way. I am assuming that this 0:28:40.224 signature is not forgeable, that somebody else could not 0:28:45.306 have generated a signature unless they actually had access, 0:28:50.666 for example, to the private key or to the 0:28:54.363 private key of A. This signature is not 0:28:57.874 forgeable. I am also assuming, 0:29:02.984 for example, that any private keys are 0:29:08.795 actually private. I am assuming that, 0:29:14.45 for example, A hasn't gone out and 0:29:19.633 broadcasted the private key to everybody else in the world. 0:29:30 Because if A had done that then I wouldn't actually know that A 0:29:33.419 actually said m. It could have been anybody who 0:29:35.955 said m. I would be in trouble if that 0:29:37.941 were the case. Any time that I sort of take 0:29:40.257 this and make this inference from seeing a sign like this 0:29:43.345 then that suggests that I am sort of making these two 0:29:46.213 assumptions. And it is not that these 0:29:48.198 assumptions are wrong or bad, it is just that it is good to 0:29:51.397 think about exclusively what they are. 0:29:53.437 Assuming that the signature is forgeable well, 0:29:55.919 we all know that cryptography can sometimes be broken. 0:30:00 It is not that cryptography is totally foolproof, 0:30:02.851 but we may have a relatively high confidence that this 0:30:06 cryptography is going to be difficult for somebody to break. 0:30:09.504 And, while assuming that private keys are actually 0:30:12.415 private, we might sort of know A personally and believe that A is 0:30:16.217 outright upstanding individual and we are going to trust that 0:30:19.782 she hasn't gone and disseminated her key everywhere around the 0:30:23.405 world because that wouldn't really be in her interest and we 0:30:26.91 take her to be a trustworthy person. 0:30:30 This is a bit of a leap of faith that we are

Internet and we shouldn't take her to be a trustworthy person. Unless. This is a bit of a leap of faith that we are making about 0:30:34.855 this. But if we believe those two 0:30:37.63 things then we can, in fact, infer that A says m 0:30:41.705 when we see a message that is signed with kA. 0:30:45.52 Let's look at another example that works with this sort of 0:30:50.462 public key cryptography that we talked about in this example 0:30:55.578 with this authentication protocol here. 0:31:00 Suppose that A tells me my public key is kA pub and this is 0:31:05.337 done over a secure communication channel like in-person 0:31:10.306 communication, I might then infer that kA pub 0:31:14.355 actually speaks for A. Now, suppose I see a message 0:31:18.957 that has been signed with kA private, I might say, 0:31:23.466 and this is an inference, that kA private says that A 0:31:28.251 says m. The question we want to ask now 0:31:32.153 is should we actually believe that A said m? 0:31:35.141 I saw a message, I know what A's public key is 0:31:38.267 and I trust A's public key maybe and I saw this sign thing. 0:31:42.297 Now I need to think about do I believe now that A actually said 0:31:46.605 m given that I saw a message that was signed with kA private? 0:31:50.774 There is something obviously that we need to do which is that 0:31:54.942 we need to verify this message. Suppose we go and we run this 0:32:00.098 verify step on this message and this verify comes out to be 0:32:04.296 true. I verify using kA pub that, 0:32:06.611 in fact, this message was appropriately signed. 0:32:09.94 Verify says yes, the signature on this message 0:32:13.197 checks. Now should I believe that that 0:32:15.875 A actually said m? That depends. 0:32:18.118 There are a couple of conditions, again, 0:32:20.94 sort of set of assumptions that we are making. 0:32:24.197 One thing we are clearly doing is assuming that this crypto 0:32:28.394 system is secure. We are also believing that this 0:32:34.006 verification step actually gives us, that by taking something 0:32:39.826 that has been signed with a private key and then verifying 0:32:45.354 it with the public key that that is as good as A actually saying 0:32:51.464 something. One thing clearly that depends 0:32:55.344 on is that the crypto system is secure. 0:33:00 0:33:10 There are a couple of other assumptions that we are making. 0:33:14.442 One thing is that it definitely, again, 0:33:17.353 requires that kA private has actually been kept secret. 0:33:21.489 And, furthermore, it requires that A didn't lie 0:33:25.012 to us about what kA public was. Because if A had given us a 0:33:29.818 wrong kA public then somebody else could have signed this 0:33:33.591 message and then we might try and verify the message using the 0:33:37.701 lied about kA public and then we might be in trouble. 0:33:41.204 Again, this is similar to this example with just kA but just 0:33:45.178 showing how it works with public and private key encryption. 0:33:49.153 It is this case that, again, we sort of need to 0:33:52.252 always think about making our assumptions as explicit as 0:33:55.957 possible. And that is really what it's 0:33:58.45 about. Deciding that we trust, 0:34:01.319 for example, the signature is true is 0:34:03.385 analogous to deciding that we have to sort of trust that we 0:34:06.713 trust the cryptography. It is possible to apply this 0:34:09.639 authentication logic in other environments as well that aren't 0:34:13.139 necessarily based on cryptography. 0:34:15.032 For example, instead of trusting that the 0:34:17.327 cryptographic system works, I might trust that a particular 0:34:20.655 communication channel is secure. For example, 0:34:23.18 if I pick up the telephone and call you and give you some 0:34:26.393 message, you might trust that it's me because, 0:34:28.975 for example, you believe that it wouldn't be 0:34:31.442 possible for somebody to fake my voice. 0:34:35 And you recognize my voice because you've come to lecture 0:34:39.117 so many times. That would be an example of a 0:34:42.279 way in which you might decide that you trust that something is 0:34:46.764 actually true, and that would be an assumption 0:34:50.073 that you're making about somebody not being able to fake 0:34:54.117 my voice. The point of authentication 0:34:56.764 logic is that it gives us a way to kind of reason about these 0:35:01.176 kinds of deductions about whether or not we actually 0:35:04.926 believe that somebody said something was true or not. 0:35:10 That is fine. We still, though, 0:35:12.542 at least when we're using public and private key 0:35:16.525 cryptography, haven't exactly answered the 0:35:20 question of how we go about establishing this initial trust 0:35:24.915 step. We said we might establish 0:35:27.542 initial trust by -- At the bottom of this is we 0:35:31.848 have to actually physically overhear somebody say something 0:35:35.819 over a secure communication channel, we think. 0:35:38.9 We have to have some way to getting this first rule where A 0:35:42.871 says that my, for example, 0:35:44.582 public key is something so we can learn A's public key so that 0:35:48.759 then we can bootstrap communication with A. 0:35:51.634 We haven't yet answered the question of how we actually 0:35:55.331 exchange these keys except for by this one method that we 0:35:59.165 talked about, this web of trust method. 0:36:03 There is this question about establishing some initial trust. 0:36:08.944 One way we might do this is using this web of trust approach 0:36:14.789 where I meet a friend, that friend tells me about some 0:36:20.04 people who he or she trusts, and then those friends' 0:36:25.092 friends' in term know a few people and eventually we 0:36:30.145 establish communication with everybody. 0:36:35 There are some computer systems that work this way. 0:36:37.828 In particular, there is a system called PGP 0:36:40.204 for pretty good privacy which is an email-based privacy 0:36:43.258 encryption system that works this way. 0:36:45.351 And there are, in fact, websites where sort of 0:36:47.896 the idea is you find somebody else who has used this system, 0:36:51.234 you share and you exchange public keys with them and then, 0:36:54.458 once you have a few public keys, you can sort of build out 0:36:57.682 this web of trust to everybody else by trusting people who your 0:37:01.189 friends trust. The problem is, 0:37:03.852 one, clearly the Internet doesn't work this way so you 0:37:06.865 don't have this kind of system that you're using on the 0:37:09.934 Internet and, two, these kinds of systems are 0:37:12.435 very difficult to set up and maintain and there ends up being 0:37:15.846 some centralized administration. You end up with having to have 0:37:19.37 some sort of way, it becomes very difficult to 0:37:21.928 actually discriminate these kinds of public keys in these 0:37:25.111 kinds of systems. The way this works is there is 0:37:27.783 a centralized website. There is a website for PGP that 0:37:31.97 is a giant database of keys and you sort of try and discover 0:37:35.72 people's keys by locking this web of trust, 0:37:38.389 but it tends to be difficult to use. 0:37:40.614 Let's talk a little bit about how this actually works in the 0:37:44.364 context of the Internet. Suppose there is actually some 0:37:47.796 person, P, who is sitting in front of this browser, 0:37:50.974 and there are some questions that we might want to ask when 0:37:54.661 we're thinking about deciding whether or not we trust this 0:37:58.283 person, P. One question that the Web 0:38:02.607 server

about dealing whether or not he trust this other person, P. One question that the Web client/server server might ask is does W know or trust person P, 0:38:07.717 or how does it? This is a commercial website 0:38:12.202 like Amazon dot com. Actually, the trust story turns 0:38:17.521 out to not be that complicated. Amazon dot com doesn't really 0:38:23.779 care that you are who you claim to be, as long as you give them 0:38:30.245 the money that they want. You can go on and tell them 0:38:35.836 that you are Oscar the Grouch, they don't care, 0:38:38.964 as long as you give them the money that they want for 0:38:42.501 whatever it is that you order. That is all that matters to 0:38:46.377 them. So, in fact, 0:38:47.534 what Amazon does is they don't actually know who you are 0:38:51.274 exactly but they might decide that they trust you because the 0:38:55.355 credit card number is good and is verified by the credit card 0:38:59.435 company. When they try and charge your 0:39:03.721 credit card they get the money and they are happy. 0:39:08.32 And there is this other question, though, 0:39:12.074 about how does P trust W? And the way that this works on 0:39:17.236 the Internet is basically like the way that this authentication 0:39:23.055 protocol that I have shown you here works, this is how SSL 0:39:28.404 works. And it turns out to be very 0:39:32.227 subtle. It can be quite hard to 0:39:34.702 actually convince yourself that you should trust a given website 0:39:39.9 if you actually start grilling down on it. 0:39:43.282 The way that it is going to decide that it trusts W is by 0:39:47.902 checking with some certificate authority. 0:39:51.202 That is how it works on the Internet. 0:39:54.172 And there are some issues with it, which is how does the 0:39:58.71 certificate authority authenticate W? 0:40:03 How does the certificate authority decide what W's key 0:40:09.022 was and how do they securely exchange keys? 0:40:13.795 How did the user get the CA's public key? 0:40:18.34 And then what if somebody's private keys are stolen? 0:40:24.136 For example, W or the certificate 0:40:27.772 authority's private keys are stolen. 0:40:33 What I want to do is just sort of show you, at a high level, 0:40:36.162 how this actually works on the Internet. 0:40:38.253 I thought it might be instructive just to see an 0:40:40.772 example of what is actually going on, on the Internet, 0:40:43.613 and to kind of illustrate how some of these issues are and are 0:40:46.883 not properly addressed by the current sort of SSL-based 0:40:49.778 Internet architecture. I have a couple of websites 0:40:52.404 open here. I have an Amazon dot com 0:40:54.227 website and I have, in this case, 0:40:55.942 an Apple developer website, both of which you will notice 0:40:58.944 have HTTPS URLs associated with them. 0:41:02 In this browser, if I click on this little 0:41:04.81 secure icon here, almost any browser would have a 0:41:08.101 similar sort of feature, a little lock somewhere that 0:41:11.666 when you are viewing an HTTPS page. 0:41:13.997 What I see is something that lists how it was that this site 0:41:18.042 actually decided that this was a secure site that it should 0:41:22.018 communicate with. When that little lock appears 0:41:25.172 it means that my browser has actually decided that it trusts 0:41:29.217 this site. We are at the point where we 0:41:33.055 have already said we trust this site. 0:41:35.698 And the reason that we trust this site, it says, 0:41:39.149 is because this site is WWW dot Amazon dot com and we have this 0:41:43.701 certificate for this site that was issued by RSA Data Security 0:41:48.179 Incorporated. You might say who is RSA Data 0:41:51.263 Security Incorporated? I have never heard of these 0:41:54.86 people. I have never exchanged any 0:41:57.283 information with them. But they are the certificate 0:42:01.76 authority. That is the person who signed 0:42:04.212 this thing. I can look and sort of drill 0:42:06.664 down on this. And what you will see is some 0:42:09.304 information about Amazon dot com. 0:42:11.316 This is the actual certificate itself. 0:42:13.642 This is some information about the certificate authority, 0:42:17.163 including information about what algorithm the certificate 0:42:20.746 authority has used to generate this certificate. 0:42:23.701 And then now what we have is this public key information, 0:42:27.222 the actual public key that corresponds to Amazon dot com. 0:42:32 The certificate contains Amazon dot com's public key that was 0:42:36.124 encrypted using the RSA algorithm. 0:42:38.392 And then there also is some information, this signature that 0:42:42.447 goes with this public key. This is the signature that was 0:42:46.297 signed by this RSA corporation. If I go to a different site 0:42:50.283 like, for example, this Apple site over here, 0:42:53.308 we see that this thing was actually signed by somebody 0:42:56.951 else. This certificate is signed by 0:42:59.288 somebody called VeriSign Incorporated. 0:43:03 Again, you have no idea who VeriSign Incorporated is, 0:43:06.072 but you will notice that these are two different certificate 0:43:09.557 authorities. It is not the case that there 0:43:11.98 is one master certificate authority out there on the 0:43:14.993 Internet that is in charge of everything. 0:43:17.356 On a Mac anyway, the same is true of most other 0:43:20.074 operating systems, there is a way that we can go 0:43:22.851 and look at the actual list of all the people who we trust are. 0:43:26.514 This is sort of an instructive thing to do. 0:43:30 To go and see who it is that you actually trust that is on 0:43:34.5 this computer. I can click on this certificate 0:43:38.052 over here. This is a list of all the 0:43:40.815 certificates on this machine that I have and where I would 0:43:45.315 accept things from. You can see there are a very 0:43:49.026 large number of certificates that this machine has installed. 0:43:53.763 These are not just certificates but actual certificate 0:43:57.947 authorities. People who I would actually 0:44:02.519 trust to sign a communication with a given site to provide me 0:44:08.119 with a certificate for, say, some Web server that I 0:44:12.786 want to communicate with. Now the question is so where 0:44:17.733 did these things come from? They came from one of two 0:44:22.586 places. Either I added them to the 0:44:25.666 system myself. In this case, 0:44:28.87 I have, for example, an MIT certification authority. 0:44:32.284 If any of you have ever used MIT certificates like when 0:44:35.899 you're logging onto Websys, in order to be able to access 0:44:39.648 that system you had to add support for a given certificate 0:44:43.464 to your browser. This MIT certification 0:44:46.008 authority is a certificate that came from MIT and this keychain 0:44:50.158 thing which shows me on my certificates has this funny sort 0:44:54.041 of message that says this certificate is not in the 0:44:57.389 trusted root database. The reason it says that is, 0:45:01.525 this is a certificate for the MIT certification authority, 0:45:04.745 I just got this certificate for the certification authority off 0:45:08.248 the Web somewhere. And I didn't do something that 0:45:10.96 it wanted me to do to actually verify that this, 0:45:13.615 in fact, was a legitimate certificate from the certificate 0:45:16.836 authority. Somebody could have put this 0:45:18.983 thing on the Web and said this is a certificate for the MIT 0:45:22.259 certificate authority and you should use it to discover 0:45:25.31 certificates for anybody who claims to be coming from MIT dot 0:45:28.7 edu. They could have lied

about it 0:45:31.839 and then I would be in trouble. It is possible that this is a 0:45:35.644 fake certificate for the MIT certificate authority, 0:45:38.816 but I sort of have made this trust, this assumption that it 0:45:42.494 is not because I downloaded it from a website. 0:45:45.348 And I believe that, in fact, this is a valid 0:45:48.076 certificate. The other things you see here 0:45:50.676 are this large number of certificates. 0:45:53.023 For example, one of the ones that we see, 0:45:55.56 let's see if we can find RSA here, this RSA security. 0:46:00 Here I see RSA security. And this thing actually says 0:46:02.872 this certificate is valid. Now there is this question 0:46:05.745 about where did RSA security come from? 0:46:07.845 And the way that this works is that this browser and, 0:46:10.718 in this case, the operating system itself has 0:46:13.149 actually been shipped with a bunch of certificates from a 0:46:16.243 bunch of different certificate authorities. 0:46:18.563 And so these have been installed by the operating 0:46:21.215 system manufacturer. And I am saying I trust Apple 0:46:23.922 to have gotten the appropriate certificates and loaded them 0:46:27.127 appropriately into the computer. And any time you download a 0:46:31.394 browser basically that browser already has a set of 0:46:33.977 certificates for a set of certificate authorities already 0:46:36.87 installed in it. And that is where all of these 0:46:39.247 base certificate authorities come from. 0:46:41.21 You can see that when you are using one of these things you 0:46:44.206 have already placed quite a bit of trust and made a number of 0:46:47.306 assumptions about who you trust and what you should trust just 0:46:50.457 by using this system. We have answered this question 0:46:53.092 now about where did the user get the public key for the 0:46:55.881 certificate authority? Either they explicitly added it 0:46:59.76 by downloading it from a website, for example, 0:47:02.315 or it came with the browser. What about this question how 0:47:05.495 does the certificate authority authenticate an actual website? 0:47:08.958 How does Amazon dot com get a certificate from VeriSign? 0:47:12.082 The answer to this turns out to be that Amazon dot com probably 0:47:15.602 paid VeriSign some money. And it is not even clear you 0:47:18.611 can say that, but in the case of VeriSign you 0:47:21.11 are probably pretty sure that Amazon dot com paid them some 0:47:24.403 money. And you can probably go to the 0:47:26.447 VeriSign website and see a list of things that they claimed to 0:47:29.911 have done in sort of authenticating Amazon dot com. 0:47:34 They may have forced Amazon dot com to present them with some 0:47:37.582 information that actually suggests that they are the 0:47:40.626 company Amazon dot com or they own some right to the name 0:47:43.97 Amazon dot com or that they are incorporated under the name 0:47:47.432 Amazon dot com. Basically, the certificate 0:47:49.88 authority has some protocol that it uses to authenticate W. 0:47:54 And we don't know what that protocol is, but you're sort of 0:47:57.001 implicitly trusting that all these certificate authorities 0:47:59.951 that Apple has already approved for you, in fact, 0:48:02.435 use some process that we will be reasonably assured of 0:48:05.178 guarantying your privacy or guarantying that the websites 0:48:08.076 that you are connecting to are, in fact, valid websites that 0:48:11.13 you should feel comfortable giving your credit card number 0:48:14.08 to. You can see that this is a 0:48:15.58 little bit dicey. If feels a little bit like it 0:48:17.961 is not clear that this is incredibly secure. 0:48:20.186 And then the last question is well, what if your private keys 0:48:23.292 are stolen? If your private keys are stolen 0:48:26.431 in this particular example, the certificates that you have 0:48:29.063 from Amazon dot com, if Amazon dot com's private 0:48:31.233 keys are stolen and one of the certificate authority's private 0:48:34.05 keys are stolen then you are in trouble. 0:48:35.85 These certificates typically have expiration dates associated 0:48:38.621 with them, but these expiration dates are often like a year or 0:48:41.437 more into the future. And so you are going to 0:48:43.469 continue to trust that certificate until this 0:48:45.501 certificate expires in the future when you will go try and 0:48:48.133 get a new one. For that whole time that you 0:48:50.072 trust that certificate somebody who had access to the private 0:48:52.843 keys from that certificate authority or from that website 0:48:55.428 would be able to pretend to be that certificate authority or 0:48:58.153 that website. So that might be problematic. 0:49:01.912 That is sort of the end of this little discussion about how 0:49:05.736 certificate authorities work. I hope you can see that this 0:49:09.494 authentication logic is sort of a way that we can start to get 0:49:13.516 at thinking about what the trust assumptions that we are making 0:49:17.604 are when we are using these systems. 0:49:19.912 This wraps up our discussion of security. 0:49:22.549 We have one class session last. And the last class session is 0:49:26.505 actually going to be a guest lecturer. 0:49:30 It is going to be somebody talking about how law and 0:49:33.341 computers and technology interact with each other. 0:49:36.551 We are kind of step back and talk much more about high-level 0:49:40.417 pictures like policy and law next time, and hopefully it will 0:49:44.348 give you a good wrap-up for the whole class of 6.033. 0:49:47.755 We will see you next time. Please come to class because 0:49:51.293 there will be questions on the guest lecturer on the exam, 0:49:55.027 so you want to make sure you don't miss it.