Lecture Notes

Chapter 7:
Application-Layer and Client/Server Protocols

CYENG 351: Embedded Secure Networking

Instructor: Dr. Shayan (Sean) Taheri
Gannon University (GU)

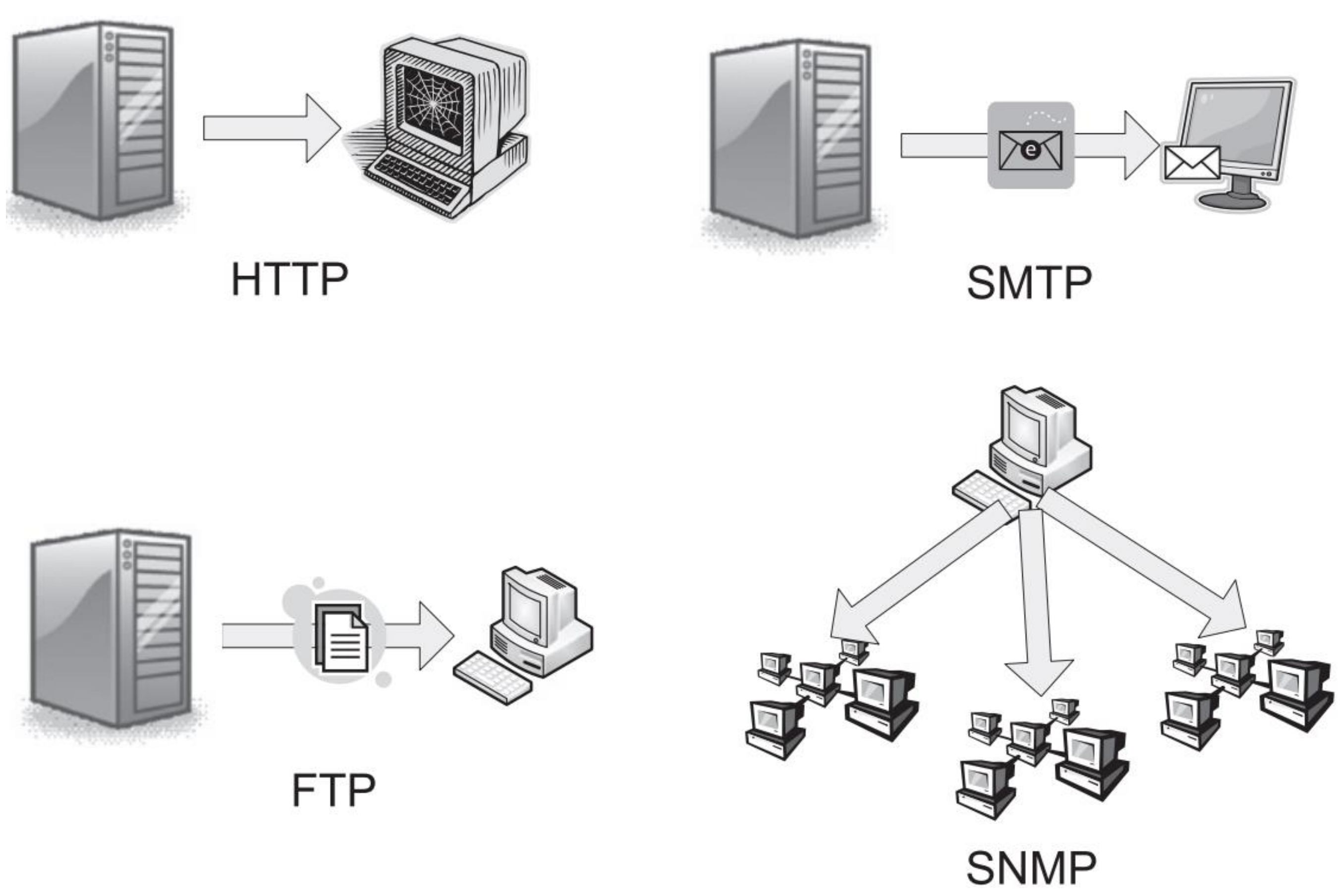
Introduction

- An increasing number of embedded applications use the Internet as a medium for communications, synchronization, or the applications provide services for the Internet and its users.
- The Internet has traditionally been dominated by **the classic client/server model of communications**, and it remains the standard model today, though some inroads are being made by new technologies, notably peer-to-peer networking.
- **Client/server applications** are characterized by a server program that is continuously available and waiting for incoming communications on a network, and a client program that runs on remote hardware (also connected to the network), that can connect to the server at any time to request a service or resource, such as a web page to display to a user or a remote process to handle a problem that is too complex for the client hardware.
- In **the embedded world (i.e., world-wide embedded network)**, applications can be set up so that the embedded hardware can be a server or a client, and can be configured to communicate with other embedded devices, PC's, and anything else that can be connected to the network.
- Constrained-resource systems designers can utilize the flexibility of client/server protocols and mechanisms to implement intuitive web-based interfaces for everyday items, and numerous other possibilities. → Their security should be considered for their widespread usages.

The World Wide Web

- **The Web** is obviously not the only client/server application that we can look at, there are many others that are used extensively on the Internet and can be of use for many embedded applications.
- Some of the protocols for the **Web include the Hypertext Transfer Protocol** (**HTTP**, the basis for the Web), the **File Transfer Protocol** (**FTP**), the **Simple Mail Transport Protocol** (**SMTP**, or email), and the **Simple Network Management Protocol** (**SNMP**).
- The Web is the most approachable from a security standpoint since most people are at least peripherally aware that some type of security exists when they are making online purchases.

The World Wide Web (Cont.)



Common Application-Layer Protocols

Web-Based Interfaces

- The Web is by far the most used Internet "**Application**" and that makes it the largest target for malicious hackers.
- For this reason, the security on the Web is some of the most mature of any network application.
- Various types of authentication, encryption, and integrity checking are used in combination to make the Web secure.
- Due to the intuitive interfaces that can be provided by Web technologies (HTML, Javascript, etc.), it makes sense to use those technologies with embedded systems, especially for those systems that require interaction with nontechnically-inclined humans.
- Many embedded devices now support web servers and clients, and the improvements in low-cost hardware have allowed some of that technology to filter down into nontraditional applications.
- One of the problems with this, however, is that **security is often sacrificed** to allow for these Internet applications to exist.
- Indeed, some developers believe that security for web interfaces is basically impossible on 8-bit microprocessors.
- There are low-cost components that come with fairly complete security options.

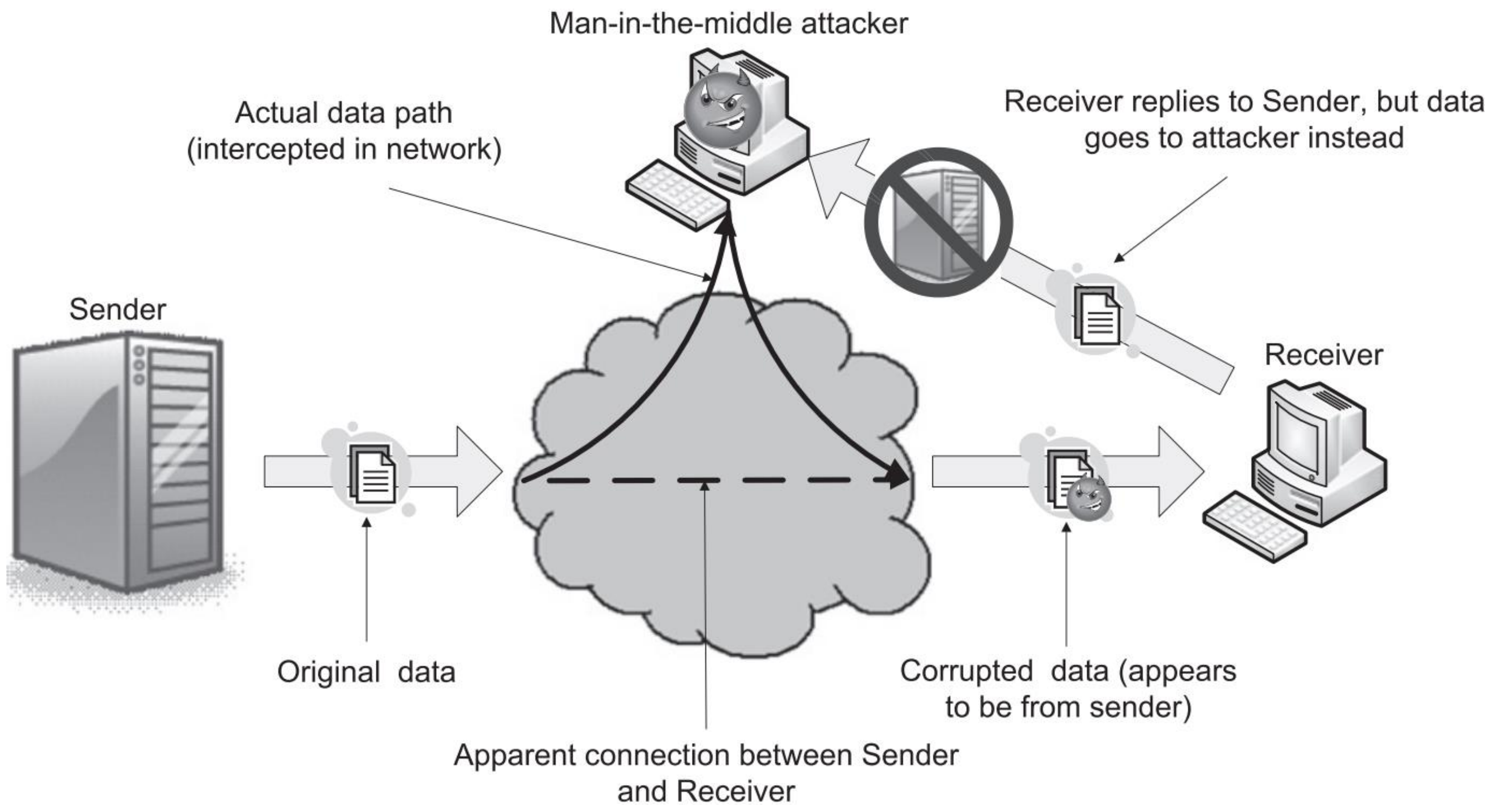
Web-Based Interfaces (Cont.)

- **What base technologies behind web-based interfaces and see what types of security are built-in?**
- **What can be added on, and most importantly?**
- **What can be left out without compromising the security of the application?**
- HTTP is the high-level transport protocol for web pages written in HTML (the Hypertext Mark-up Language).
- HTTP controls the handling of requesting, serving, and processing of HTML files through a client/server model that includes methods for requesting resources, providing resources, and handling error conditions **in a controlled manner**.
- HTTP is **a text-based protocol** that allows for web browsers and servers to send requests for resources and responses to requests over a standard TCP/IP socket connection, or any other network communications protocol that allows for remote network communications using text.
- At its deepest levels, HTTP is essentially a very simple protocol.
- The protocol basically consists of a request sent by the client for a particular resource.
- The server, upon parsing the request, will look for the corresponding resource and provide a response to the client with the resource (typically a text file consisting of HTML-formatted text and images) if that resource exists.
- If the resource does not exist, or some other problem occurred during the transmission or parsing and handling of the request, **HTTP defines a host of error conditions and corresponding codes** to indicate to the client that the resource is unavailable or somehow unreachable.

Web-Based Interfaces (Cont.)

- **HTTP Authentication** comes in two different flavors: **Plain**, which just asks (via a plaintext request to the client) for a username and password, and **Hashed Authentication**, which uses a cryptographic hash algorithm to provide slightly more security than the plain authentication.
- The hash authentication mechanism is of some interest to the embedded systems designer because it represents probably the simplest secure method for authentication (plain authentication is not really secure at all since the password is sent over the network in plaintext - it can be of use with a protocol like SSL, however).
- HTTP authentication as a possible security mechanism for a secure embedded application.

Web-Based Interfaces (Cont.)


Man-in-the-middle attacker
Actual data path (intercepted in network)
Receiver replies to Sender, but data goes to attacker instead
Sender
Receiver
Original data
Corrupted data (appears to be from sender)
Apparent connection between Sender and Receiver

Man-in-the-Middle Attack

Server-Side HTTP Web Interfaces

- For many applications, it will be desirable to implement a web server on the target device, so a user on a PC somewhere on the network can use a natural client application - a simple web browser.
- Almost all web browsers today support various types of security mechanisms, **from the simplest plaintext HTTP authentication to the full secure channels** provided by HTTPS (HTTP secured using SSL).
- For HTTP, **the basic built-in security** is provided by the various forms of HTTP authentication.
- If we are going to implement a web server for our embedded application, then we need to look at how to efficiently generate a challenge message for the connecting clients so that the authentication can be reliable and secure.
- In order to assure that the challenge to a connecting client is secure, we need to have some level of confidence that the challenge itself (a random number) has a very low probability of being predicted by an attacker.
- If the generated challenge is not a cryptographically secure number, an attacker can more easily produce a hash to provide to the server application that will fool the server into accepting an invalid user, or worse, attain the actual hidden password.

Server-Side HTTP Web Interfaces (Cont.)

- If the attacker can predict challenges that will be sent in the future, it would be possible to do **a dictionary attack on an older message** (with **a known challenge**), derive the password, and then produce a "correct" hash when a predicted challenge is sent.
- For this reason, we need to be sure that the number we generate is as unpredictable as possible.
- We can utilize **a combination of network traffic and hardware properties** such as interrupts firing, serial port traffic, and incoming network packet sizes to feed into a pseudo-random number generator **to provide fairly secure challenges**.
- However, **none of these sources alone is very good to use**, since each source on its own can be manipulated to be more predictable. → If the application does not need the utmost in security (for example, if the application resides on a physically secure and isolated network) then the embedded device should be adequate enough to generate sufficient challenges.
- There are not a great number of things that we can do to make the server implementation of the challenge **less resource-hungry**.
- Unfortunately, there are not a whole lot of features that can be removed from the server side and still have it be compatible with a wide array of HTTP clients.
- Each browser should be **compatible with the HTTP specification**, but the problem is that there are a great number of inconsistencies between the implementations of different browsers, which leads to many possible **incompatibilities**.
- **The best thing** to do is **to support the smallest number of HTTP features** that will work with the broadest number of client browsers.

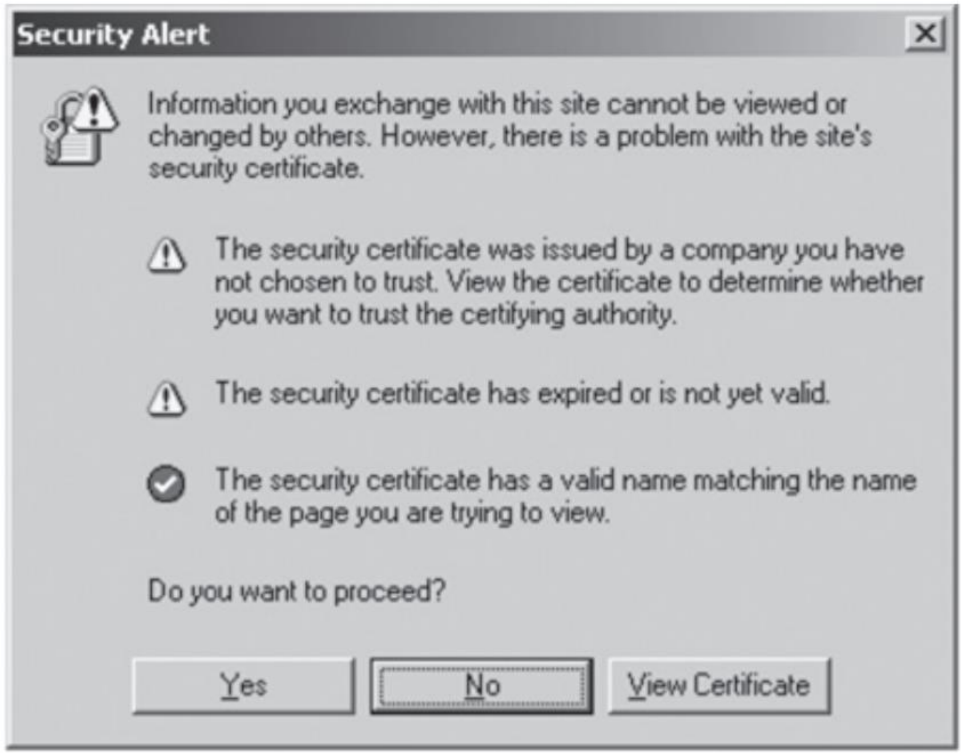
Server-Side HTTP Web Interfaces (Cont.)

- **Security for server-side HTTP** can include a number of **different protocols and options**.
- At the very basic level, we can **implement digest authentication** using a standard hash algorithm such as SHA-1.
- For a little bit better security, we can add in some form of encryption using AES or some other symmetric-key algorithm. → Obviously, this would require shared keys, but depending on the application this may be adequate.
- If we had the advantage of the device being on a VPN or other such hardware-encrypted channel, this would satisfy our security requirements, and a simple password authentication (no digest/hashing required) could suffice for authentication purposes.
- If we had a bare device out on a public network, however, we would need a more robust security setup. → For HTTP, this means HTTPS - HTTP secured using SSL.
- **HTTP** is an inherently stateless protocol, and **SSL** requires the state to check for attacks (especially truncation attacks).
- **HTTP** normally closes a connection by simply closing the TCP/IP socket.

Server-Side HTTP Web Interfaces (Cont.)

- All modern web browsers support some form of HTTPS, and the port redirect (which is triggered by prefixing the URL with "https" in place of "http") usually happens without any user interaction.
- If the certificate is known by the browser, or has been authenticated by one of the browser's affiliated **Certificate Authorities**, then the browser simply loads the secure page.
- If the certificate is unknown, malformed, or is not independently authenticated, then the browser will display warnings prompting the user to decide whether or not to proceed with loading the page.
- Every browser has a slightly different way of notifying the user about errors (which, if you remember, are possibly attacks), but those behaviors are somewhat dictated by SSL and are generally similar.
- In any case, an error in the browser can be disquieting to a user, so it is usually best to handle these cases either through documentation or appropriate use of SSL certificates.

Server-Side HTTP Web Interfaces (Cont.)


Security Alert

Information you exchange with this site cannot be viewed or changed by others. However, there is a problem with the site's security certificate.

⚠ The security certificate was issued by a company you have not chosen to trust. View the certificate to determine whether you want to trust the certifying authority.

⚠ The security certificate has expired or is not yet valid.

✓ The security certificate has a valid name matching the name of the page you are trying to view.

Do you want to proceed?

Screenshot of Internet Explorer Receiving a Possibly Invalid Certificate

HTTP Client Web Interfaces

- **<u>Implementing client-side HTTP</u>** has some distinct advantages over server-side, especially when it comes to <u>utilizing the built-in security mechanisms of the HTTP protocol</u>.
- With the embedded device <u>utilizing a client-side implementation</u>, then the cryptographic random numbers used for <u>digest authentication (the challenge numbers)</u> are likely generated using more powerful hardware (on the server end), with much <u>more capability of generating appropriately secure numbers</u>.
- Besides this useful property, the **client-side implementation** can be much simpler than the server.
- It is much more likely that you will only need <u>to support a single server</u> with your client implementation than being able <u>to support only a single client implementation</u> with an embedded HTTP server.
- **<u>The security of an HTTP connection</u>** is typically dictated by the server, so the security needs of an <u>embedded HTTP client</u> are dependent upon <u>the specific application</u>.
- If the client will only connect with a single server, then <u>the security can be tailored</u> to match that server's needs, <u>allowing for a reduction in application code size</u>, since the security options need not be comprehensive.
- If the client will be contacting multiple, possibly unknown servers, then the scope of security mechanisms supported should be broad and flexible.

Combination Client/Server HTTP Applications

- **A combination approach** may be desirable if the embedded devices in the application need to communicate with one another, as well as with larger PC or server machines.
  - The disadvantage here is that we need to support a broader range of technologies in order to support both client and server HTTP, which will require additional space and memory.
- **HTTP provides a slick**, usually highly graphical interface for many applications.
  - Sometimes the application does not need (or simply cannot support) a full web server.
  - A console interface may be more appropriate.

Console Applications

- In many cases, the embedded device on the other end of your network may not be sophisticated enough to bother trying to implement a full HTTP stack or go all out with full-blown SSL and HTTPS.
- Of the alternatives to HTTP, one of the most common methods for communicating that is still in widespread use today is **the venerable Telnet**.
- **The Telnet protocol** is exceedingly simplistic, providing basic text communication capability that is designed explicitly for implementing remote console interfaces.
- Telnet is a natural technology for embedded devices since it has very little overhead and can be implemented very simply.
- **Telnet**, with its low overhead and simple implementation, is great for embedded applications but suffers from one major drawback: it has no security options.
- Like HTTP, Telnet is designed to send information in the clear.
- There are several alternatives to **plain Telnet** that provide complete all-around security.
- The most recognizable of these alternatives is the **Secure Shell protocol**, or **SSH**.
- In reality **SSH is a complete replacement for Telnet** and has basically nothing to do with SSL. → It also provides a number of additional features that make it more of **a protocol suite** than a single protocol.

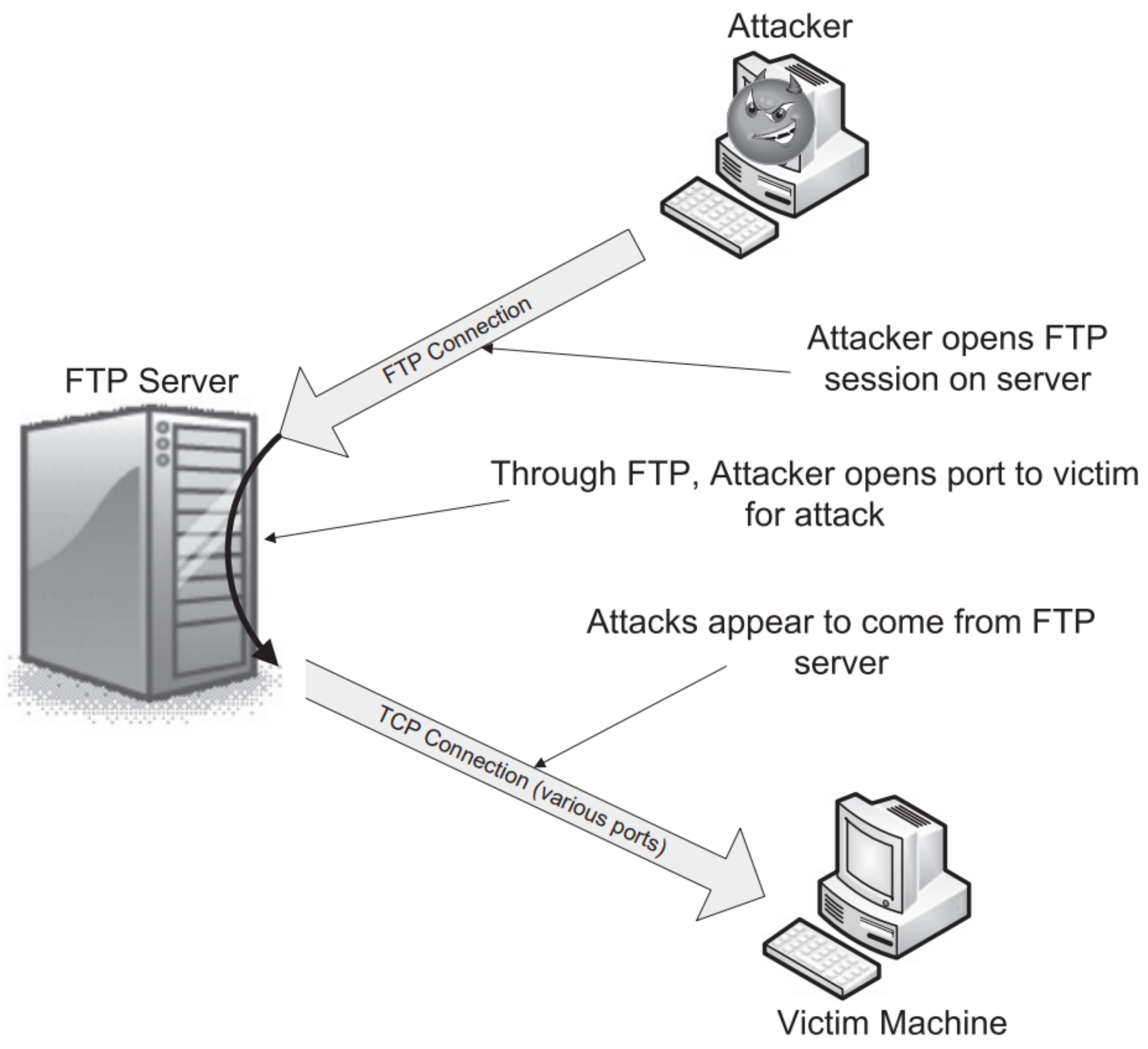
Console Applications (Cont.)

- **SSH and SSL share a lot of properties**:
  - They both utilize public-key cryptography to exchange symmetric encryption keys.
  - They both utilize a similar handshake method to establish a connection.
  - They both have similar integrity checking mechanisms using various hashing schemes.
- SSH is fairly popular, being that it is **a simple, efficient, secure implementation** that can be used in place of more general solutions like SSL.
- In reality, both SSH and SSL-based secure Telnet are nearly equivalent from the user's standpoint.
- The protocols are designed to be mostly invisible (SSL more so than SSH), so you can build your console to be compatible with either.
- **Telnet uses plaintext only**, no fancy graphics or fonts, so it is fast and has the added benefit of not requiring serious translation for the device to understand.
- **Implementing Telnet over SSL** is as simple as replacing any TCP/IP socket calls with the appropriate SSL API calls in the Telnet source code.
- The **File Transfer Protocol (FTP)** and its variants can provide useful functionality, but like anything else that uses a network, can benefit from security.

File Transfer Protocol

- **FTP** provides a network interface for file transfer operations, so that files may be shared between physically remote systems.
- The typical interface to FTP is a console, similar to Telnet or other remote console protocols, but the interface is just a wrapper for the actual protocol.
- FTP suffers from a complete lack of real security.
- An FTP client may prompt the user for a username and password, but the protocol itself simply transmits everything in plaintext over the network.
- FTP can be used for a wide range of functions in an embedded system, from data gathering to uploading firmware updates.
- The FTP protocol will happily forward a file to any port the user chooses, and if that file contains relevant commands for a particular service in ordinary text, the file will be treated as incoming commands.
- If the FTP server and the victim machine are behind **a firewall**, it should be easy to see that an attacker could access the victim machine even if it was not directly accessible from outside the firewall.
- A couple of ways to secure against this vulnerability are to restrict the ports the FTP server can use to unused TCP ports, or to simply disallow the target port to be redirected.

File Transfer Protocol (Cont.)


Attacker
FTP Connection
Attacker opens FTP session on server
FTP Server
Through FTP, Attacker opens port to victim for attack
Attacks appear to come from FTP server
TCP Connection (various ports)
Victim Machine

FTP Bounce Attack

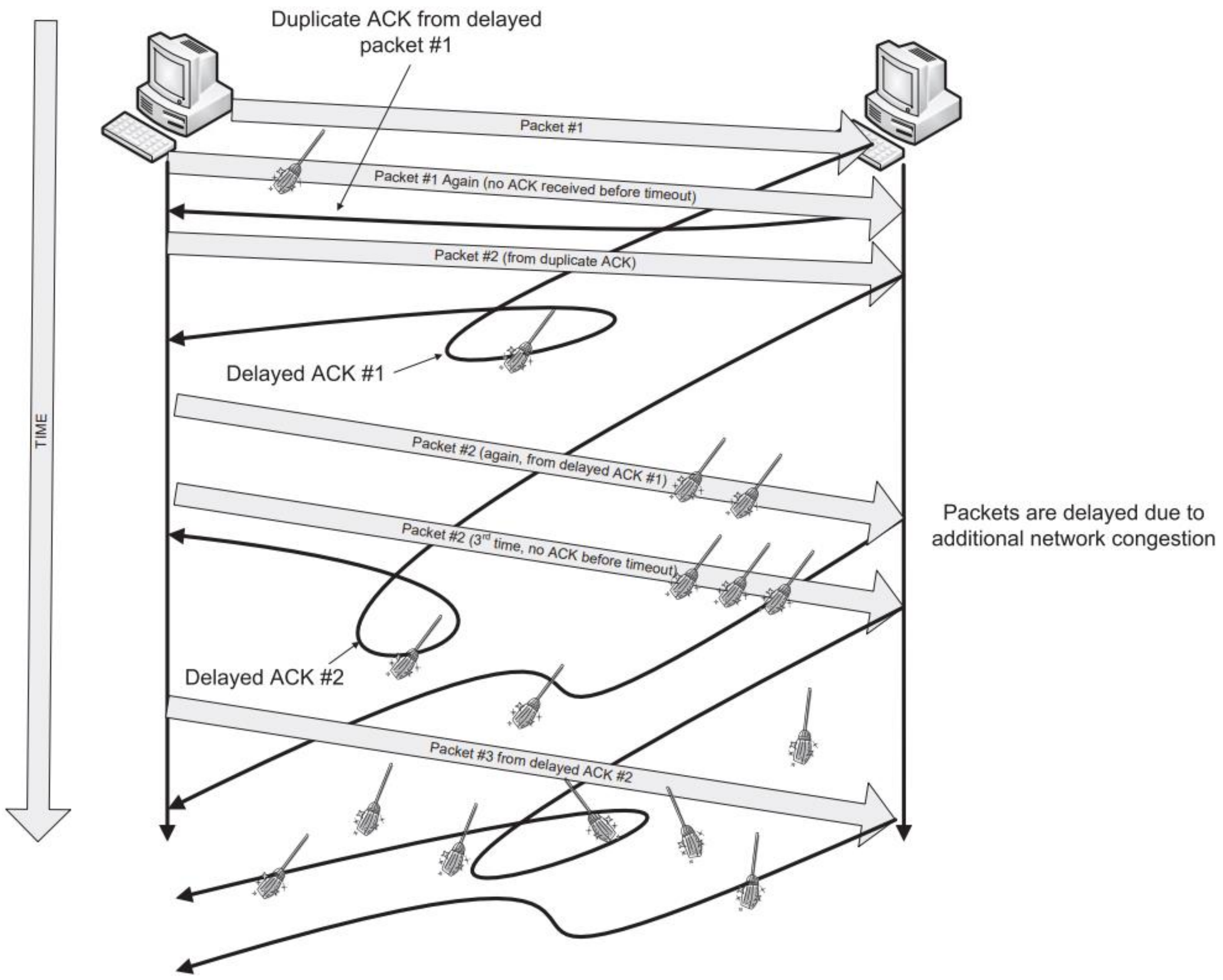
File Transfer Protocol (Cont.)

- Somewhat more problematic for embedded applications are the issues with passwords in FTP.
- The FTP spec allows for unlimited password attempts after a failure.
- This **"feature"** enables brute-force password guessing through automated means - in an embedded application with limited resources, the bigger threat would really be **a denial-of-service condition** while the attacker continually tries new passwords without letting other client connections through.
- The limited resources would likely not respond quickly enough for the brute-force attack to be as effective as it would be if the server was a more powerful machine.
- Finally, the plaintext transmission of password information is exactly the same issue as the one for HTTP authentication and the Telnet protocol.
- The only real solution here is to use FTP over some form of security protocol such as SSL or IPSEC.
- Using FTP over SSL is very similar to **the use of HTTP over SSL or TLS**.
- **The full FTP protocol** offers a large number of options for implementation, many of which are not strictly necessary for some applications.

File Transfer Protocol (Cont.)

- **Trivial File Transfer Protocol (TFTP)** was only intended for the reading and writing of files, without any of the other features (such as listing directory contents) of FTP.
- A flaw in the design led to an error condition known as **"Sorcerer's Apprentice Syndrome"** (or SAS, named after the animation sequence from Disney's classic movie Fantasia).
- TFTP was implemented over UDP, so the timeout was necessary to assure that the entire file was transmitted, but it hid an interesting problem → If certain packets were not actually lost but simply delayed enough to violate the timeout (usually due to network traffic congestion), the packets would be considered lost.
- In **TFTP**, the receiving implementation sends a message (ACK) back to the sender after each packet is received.
- **TFTP** is of particular interest to embedded systems designers, since it can be used as a method for updating firmware.

File Transfer Protocol


Duplicate ACK from delayed packet #1
TIME
Packet #1
Packet #1 Again (no ACK received before timeout)
Packet #2 (from duplicate ACK)
Delayed ACK #1
Packet #2 (again, from delayed ACK #1)
Packet #2 (3rd time, no ACK before timeout)
Packets are delayed due to additional network congestion
Delayed ACK #2
Packet #3 from delayed ACK #2

Sorcerer’s Apprentice Syndrome and TFTP

File Transfer Protocol (Cont.)

- To <u>secure a firmware upload</u> using something like RC4, the device would have to be initialized with some key (and an initialization vector since RC4 is a stream cipher).
  - **<u>Rivest Cipher 4 (RC4) is a form of stream cipher</u>**. → It encrypts messages one byte at a time via an algorithm.
- It is recommended that <u>a protocol be implemented that allows this key to be changed upon deployment</u>, so the compromise of a single device would not lead to a possible compromise of all other identical devices in the field.
- Even better would be <u>to initialize every device with a unique key</u> (the logistics of this may be prohibitive, however).
- Having the key, <u>any file to be uploaded to the device would first be encrypted</u> and then TFTP would be used to send the encrypted image.
- To further <u>add to the security of the image</u>, **a checksum should be calculated on the original image** and included in the encrypted payload.

Email, DNS, DHCP, and SNMP

- **Certain protocols are distributed throughout a network**, rather than being simply available between two devices. → For Email Communication → Simple Mail Transfer Protocol (SMTP), Domain Name Service (DNS), Dynamic Host Configuration Protocol (DHCP), and Simple Network Management Protocol (SNMP).
- Probably the most recognizable and common form of communication on the Internet, **email** suffers from a complete lack of security in its default state.
- In the embedded world, **email** provides a simple and effective form of notification when an embedded device requires attention.
- **SMTP** can be easily implemented, and email addresses are easily set up.
- **DNS**, relies on a distributed network of servers to provide a lookup service that maps IP addresses to human-readable names, called domain names or **Uniform Resource Locators (URLs)**.
- **DNS** is a complex combination of protocols and functionality that provide a dynamic mapping of network addresses, allowing for multiple machines to be represented by single domain name.
- One protocol that is commonly used to provide IP addresses in a dynamic fashion is the **Dynamic Host Configuration Protocol, or DHCP**.
- For embedded devices, **DHCP** is only important in that it is commonly used to assign addresses to devices to avoid collisions between static IP addresses.
- Originally designed to provide monitoring and configuration services for remote servers, **SNMP** has gone through three incarnations, referred to as SNMPv1, SNMPv2, and SNMPv3.
- In general, **SNMP** is anything but simple, so securing it is also a complex task.

Assignment

- **Reading Assignment:**
  - Stapko, T., 2011. **Practical embedded security: building secure resource-constrained systems**. Elsevier.
    - ✓ "Chapter 7: Application-Layer and Client/Server Protocols", Pages 129-148.

Questions?