PROTOCOL BASICS

Electronic message, point A to point B

SMTP OVERVIEW

- SMTP is the client/server protocol used to ship electronic mail messages, perhaps locally between processes, or more commonly from one server to another, using the network.
- SMTP is an Application Layer protocol, implemented in plain text, having only one Transport Layer requirement: a reliable and ordered data stream. TCP is most commonly used.
- The standard SMTP TCP port number is 25, though there is also a special “submission” port, which is 587.

PRIMARY SMTP RFC'S

- The original specification for SMTP was RFC 821, written in 1982, and pulling material from numerous earlier RFC's governing mail protocols.
- In 2001, a rewrite and update for the SMTP protocol was released as RFC 2821.
- The latest SMTP standard is RFC 5321, written in 2008:
  - `http://tools.ietf.org/html/rfc5321`

BARE BONES SMTP

- At minimum, for a client to send an email using SMTP, four commands must be sent to the server in proper sequence: `EHLO, MAIL, RCPT, DATA`.
- There are, of course, many other commands, most of which will be covered later in the course. For now, the focus is on the basics, which is where most of the heavy lifting occurs with SMTP.

A FEW NOTES

- All lines must end in a carriage return, then a line feed, noted as *<CRLF>*. **All lines in the examples assume this ending, but do not always show it.** Occasionally, *<CRLF>* will be used to indicate the end of a line explicitly, in order to better illustrate a concept. ➥RFC 5321 §2.3.8
- Commands are **case insensitive**.

EHLO

- The `EHLO` command stands for "Extended Hello," and is the way a client identifies itself to a server. Originally, there was just the "Hello" command, `HELO`, but it is now deprecated.

- Extensions to SMTP (discussed later, and known as ESMTP), use `EHLO` to identify what capabilities a mail server supports beyond basic SMTP. The only argument needed is the client's fully qualified hostname. Syntax and Example:

    - `EHLO hostname<`*`CRLF`*`>`

    - `EHLO me.com`

MAIL

- The `MAIL` command is used to specify the return address for the forthcoming email message. Syntax and Example:
  - `MAIL FROM: sender<`*`CRLF`*`>`
  - `MAIL FROM: <bob@me.com>`

RCPT

- The `RCPT` command is used to specify email addresses for the recipient(s) of the forthcoming message. If there is more than one recipient, the command is repeated. Syntax and Sample:
- `RCPT TO: receiver<`*`CRLF`*`>`
- `RCPT TO: <alice@you.com>`

DATA

- The `DATA` command marks the beginning of the message content, and is terminated by a period on a line by itself:
  - `<CRLF>.<CRLF>`
- Example:
  - `DATA`
  - `Content of message....`
  - `.<CRLF>`

QUIT

- The `QUIT` command is the correct method of disconnecting from the mailserver.
- Once issued, the server will return a `221` response and disconnect the network connection.
- Any pending mail transactions will be aborted.

INEVITABLE QUESTION:

- If a period on an empty line terminates the `DATA` command, can you send an email with a period on a line by itself?!
- Yes, of course you can. To address that problem, a technique known as "dot stuffing" is used.
- Dot stuffing calls for the client to insert a period at the start of any message line originally starting with a period. If the server receives a line containing only a period, this is the end of data. Otherwise, the server drops the prepended period and continues as normal. ➥RFC 5321 §4.5.2

SO, ALL TOGETHER NOW

- `EHLO me.com`
- `MAIL FROM: <me@me.com>`
- `RCPT TO: <you@you.com>`
- `DATA`
- `Blah blah blah here is the email blah blah`
- `.`*<CRLF>*
- `QUIT`

THAT’S IT?

- Yup. Well, mostly. As mentioned previously, there are more commands and extensions that will be covered.
- Also, server response codes are extremely important, so let’s begin discussing them now!

SMTP REPLIES

- SMTP replies consist of a three digit number, possibly followed by a brief text message, normally all contained on one line, though multiple lines are possible depending on the server software and the command sent (notably, `EHLO`, `EXPN` and `HELP` generally result in more than one line of response). Examples:
- `220 mail.example.com ESMTP`
- `250 mail.example.com Hello me.com, pleased to meet you`
- `550 5.7.1 Access denied`

SMTP REPLIES

- Each digit of the code has special meaning
- The first digit indicates the overall response: good, bad or incomplete.
- The second digit indicates the category of response, such as syntactical, connection, information, etc.
- The third digit provides additional detail about the category referred to in the second digit.

FIRST DIGIT

- 2__: Positive completion
- 3__: Positive intermediate, need more information
- 4__: Temporary negative completion
- 5__: Permanent negative completion

SECOND DIGIT

- _0_: Syntax errors
- _1_: Informational responses
- _2_: Connection responses
- _3_: Unspecified
- _4_: Unspecified
- _5_: Mail system responses

THIRD DIGIT

- The third digit provides additional meaning to the category given by the second digit. Accepted values given in RFC. Examples:
    - `500` - Syntax error, unrecognized command
    - `501` - Syntax error, invalid arguments
    - `502` - Command not implemented
    - `503` - Bad command sequence
    - `504` - Command parameter not implemented

TEXTUAL RESPONSE

- The textual component of the response is generally meant for human consumption, though there are a handful of responses which should be parsed by the client: 220, 221, 251, 421, and 551.
- For extra credit, note that if the reply text is longer than one line, every response except the last will start with the same code, followed immediately by a hyphen. The last line will not contain the hyphen:

```
250-First line

250-Second line

250 Last line
```

RESPONSES

- The official list of response codes and suggested texts is available in the RFC: ➥RFC 5321 §4.2.3
- Additional response codes are possible for future use, and research into the server software will be necessary to understand their meaning.
- Note that many servers implement an additional code to help indicate to clients what is going on - this is specific to the server software, and not an officially standardized component of SMTP. This value will be sent with the textual component of the response.

```
$ telnet server1.example.com 25
Trying 192.168.1.100...
Connected to 192.168.1.100.
Escape character is '^]'.
220 server1.example.com ESMTP Postfix
ehlo me.com
250-server1.example.com
250-PIPELINING
250-SIZE 10240000
250-VRFY
250-ETRN
250-ENHANCEDSTATUSCODES
250-8BITMIME
250 DSN
mail from: <bob@me.com>
250 2.1.0 Ok
rcpt to: alice@server1.example.com
250 2.1.5 Ok
data
354 End data with <CR><LF>.<CR><LF>
This is a test message!
.
250 2.0.0 Ok: queued as BF4B329EC7
quit
221 2.0.0 Bye
Connection closed by foreign host.
```

LAB 1

1) Spend a few minutes browsing through the RFC for SMTP. Focus on sections 4.1 and 4.2.

2) Using telnet and the four basic commands learned previously, send a simple test email message to stationX@server1.example.com where X represents your station number.

3) Verify receipt of the message by setting up Thunderbird:

   Incoming & Outgoing Mail Server: **server1.example.com**

   Incoming Mail Server Type: **IMAP**

   Username/Password: **stationX**

BUT THAT'S NOT HOW I SEND EMAILS...

- Indeed. We rarely use telnet to send emails. (Except to impress at the parties).
- Normally, a piece of software called a Mail User Agent (MUA) is used to manage all email interfacing - both sending and receiving. Thunderbird is an MUA. So are Outlook and Apple Mail.
- An MUA handles the details of interfacing with the user and with mail servers to provide an intuitive, reliable tool for sending and receiving emails.

LOGGING SMTP CONVERSATIONS

- **Thunderbird** - From command prompt:
  - `export NSPR_LOG_MODULES=SMTP:4`
  - `export NSPR_LOG_FILE=~/Desktop/smtp.log`
  - `/Applications/Thunderbird.app/Contents/MacOS/thunderbird-bin` *(All on one line)*

LOGGING SMTP CONVERSATIONS

- **Outlook** - Overly involved. See:
  - `http://support.microsoft.com/kb/300479`
- Not complex to set up, just quite a few steps to follow, and it’s different for each version of Outlook.
- Also, the log files will be in different locations based on version of Outlook.
- Google will always be your friend for this. :)

LOGGING SMTP CONVERSATIONS

- **Apple Mail** - Also overly involved. See google for tips.
  - Basically, double-click `/Library/Scripts/Mail Scripts/Turn on Logging.scpt`
  - Run the newly opened apple script, and select “Sending”
  - Watch the console log using the Console utility

LAB 2

1) Set up SMTP conversation logging in Thunderbird and send a second test email to stationX@server1.example.com. Verify receipt.

2) Analyze the SMTP conversation and note a few of the differences from the manual steps used in Lab 1. What do you suppose is going on?

3) Dig into the RFC a bit more. Sections 2.1, 3.1, 3.2, 3.3 should be of particular interest.

SERVER SIDE LOGS

- Server side logs are naturally dependent on the software being run for the SMTP server.
- On *nix systems, this is commonly Sendmail, Postfix or Qmail.
- On *doze systems, this is commonly Exchange.
- The location, format and level of information available in the logs will be dependent upon software and configuration, but let’s take a look at the logs on server1.example.com, which is running Postfix.

```
slideshow.end();
```

HEADER BASICS

Subject: Are we having fun yet?

EMAIL CONSTRUCTION

- Up until this point, we have been considering the `DATA` command to encompass the body of the email message.
- This isn't entirely accurate.
- Email messages are actually composed of two parts.
  - The Headers
  - The Body

EMAIL HEADERS

- Email headers are actually a very important aspect of an email message.
- Email headers describe who sent a message, the subject, who the message was destined for, delivery information and more.

MORE RFC'S!

- Enjoyed RFC 5321 enough?
- Well, RFC 5322 dictates the format for email messages
- This is another RFC you need to be familiar with.
  - `http://tools.ietf.org/html/rfc5322`

OVERVIEW

- The main parts of RFC 5322 you need to know are pretty simple:
- Email messages start with the headers, which are lines that start with a header name, followed by a colon, and then the value. One or more headers can be included in a message.
- A blank line terminates the header section and starts the body.
- The body is terminated by `<CRLF>.<CRLF>`

COMMON HEADERS

- Some of the common headers include:
  - Subject, To, From, Date, Reply-To, CC, BCC
- Technically, the only required headers are:
  - Date, From
- ➥RFC 5322 §3.6

```
$ telnet server1.example.com 25
Trying 192.168.1.100...
Connected to 192.168.1.100.
Escape character is '^]'.
220 server1.example.com ESMTP Postfix
helo me.com
250 server1.example.com
mail from: <bob@me.com>
250 2.1.0 Ok
rcpt to: alice@server1.example.com
250 2.1.5 Ok
data
354 End data with <CR><LF>.<CR><LF>
From: <bob@me.com>
Date: Wed, 22 Dec 2010 22:31:21 -0600 (CST)
Subject: Test Message #2
Reply-To: <bob@me.com>

This is another test message!  A properly formatted one!
.
250 2.0.0 Ok: queued as BF4B329EC7
quit
221 2.0.0 Bye
Connection closed by foreign host.
```

LAB 1

1) Spend a few minutes browsing through the RFC for the Internet Message Format. Focus on sections 2.1, 2.2, 2.3, and 3.6.

2) Use telnet to send a simple test email message to stationX@server1.example.com where X represents your station number. Include at least the following headers: Subject, To, From, Date, Reply-To.

3) Verify receipt of the message in Thunderbird. Check that Thunderbird properly understood the message and headers. Does the Subject appear correctly? What about the date?

4) Compare the message you just sent with ones sent in previous labs. Are there headers on the old emails? Where did they come from?

TRACE FIELDS

- The trace fields are used just for that - tracing message paths. **This is extremely useful debugging information!**
- The most important trace field is the Received header.
- Messages should normally contain one or more Received header lines, tracing a path from the sender to the receiver.
- ➥RFC 5321 §4.4

RECEIVED LINES

- The RFC states that each time an SMTP server accepts a message, either for relaying (discussed later) or local delivery, it **MUST** prepend a Received line to the existing headers.
- This allows for a very convenient method of tracing an email through various servers on it’s way to delivery!
- Note: the format of the text in the received line is only loosely guided by the RFC - in the wild you will see many variations, and occasionally find Received lines being used internally by mail providers for private debugging (gmail, yahoo, etc)

```
Return-Path: <deborah.carter@rackspace.com>
X-Original-To: nathan@edgecloud.com
Delivered-To: xxxxxxxxx@homiemail-mx9.g.dreamhost.com
Received: from mx1.dfw1.rackspace.com (mx1.dfw1.rackspace.com [72.3.128.180])
        (using TLSv1 with cipher DHE-RSA-AES256-SHA (256/256 bits))
        (No client certificate requested)
        by homiemail-mx9.g.dreamhost.com (Postfix) with ESMTPS id 8F782285AD5
        for <nathan@edgecloud.com>; Tue, 21 Dec 2010 07:59:14 -0800 (PST)
Received: from SAT2EXH02.RACKSPACE.CORP ([10.3.22.154])
        by mx1.dfw1.rackspace.com (8.14.2/8.14.2) with ESMTP id oBLFx6LE014292;
        Tue, 21 Dec 2010 09:59:11 -0600
        (envelope-from deborah.carter@rackspace.com)
Received: from SAT2EXD04.RACKSPACE.CORP ([169.254.4.106]) by
 SAT2EXH02.RACKSPACE.CORP ([10.3.22.154]) with mapi id 14.01.0255.000; Tue, 21
 Dec 2010 09:59:09 -0600
From: Deborah Carter <deborah.carter@rackspace.com>
To: "'Nathan Isburgh'" <nathan@edgecloud.com>
CC: Duane LaBom <duane.labom@rackspace.com>
Subject: Happy Holidays!
Thread-Topic: Happy Holidays!
Thread-Index: AcuhJ/9WF23yaKPhT5+ghDadIZnyTw==
Date: Tue, 21 Dec 2010 15:59:09 +0000
Message-ID: <0BA978672ACDB141B758F523E421AFE40B669C@SAT2EXD04.RACKSPACE.CORP>
Accept-Language: en-US
Content-Language: en-US
X-MS-Has-Attach: yes
X-MS-TNEF-Correlator:
x-originating-ip: [10.3.22.134]
Content-Type: multipart/related;
        boundary="_004_0BA978672ACDB141B758F523E421AFE40B669CSAT2EXD04RACKSPAC_";
        type="multipart/alternative"
MIME-Version: 1.0
```

LAB 2

1) Return to RFC 5321 and spend some time on section 4.4.

2) Reexamine the headers on all of the test emails you have sent. Can you understand the Received lines?

3) Look through some real email messages - work or personal. Check out the headers and see if you can follow the Received lines from originating server to final destination. Note how different mail systems use the Received lines.

4) Consider: How might you use this information to find spammer IP addresses?

```
slideshow.end();
```

SMTP, DNS AND SPAM

Flippity Flop… It’s done! SPAM-burger on a bun!

“REAL” EMAILS

- So how does a “real” email get from a sending user’s MUA to the destination mail server (which eventually will get to the receiver’s MUA)?
- For example, when someone emails instructor@edgecloud.com, what happens?

STEP ONE

- First off, the user composes their email, adding a subject, recipient, body, etc.
- When the user presses Send, the MUA takes all of the information presented and wraps it up into a mail object, exactly like we discovered earlier. To and From headers, the Subject header and more, are all pieced together with the body of the email message.
- Now the MUA needs to send the mail object out...

STEP TWO

- Remember setting up the “Outgoing mailserver” in Thunderbird? This is known as your relay mailserver. Relaying will be discussed in more detail later.

- For right now, consider the relay mailserver as your email shipping center. Your MUA needs this relay mailserver to know where to send outgoing email messages.

- Note: With Outlook and Exchange (*shudder*), the process is slightly different, because Microsoft loves to reinvent the wheel constantly, and they use a proprietary exchange format between clients and the Exchange server. Yay.

STEP TWO

- The MUA connects to the relay server and uses SMTP to transfer the mail message.
- The relay server is now responsible for getting the email delivered to the correct server.
- There are several reasons why this is done:
  - Reliability - temporary failures on receiving server
  - Limited Spam mitigation
  - Performance - parallel deliveries

STEP THREE

- Now the relay server has this email to deliver, and has to figure out where to send it..
- DNS to the rescue!
- There are records in DNS known as MX records. These records detail what servers are responsible for email for a given domain.

MX RECORDS

- Mail Exchanger: Defines hosts responsible for incoming email for the named zones.

  zone* IN MX *preference mailhandler

- Example:

```
edgecloud.com. IN MX 10 mail1.edgecloud.com.
```

LOOKING UP MX RECORDS

- `dig domain mx`
- `nslookup`
  - `set query=mx`
  - `domain`

STEP THREE

- Using the MX records, the relay server now knows what machine to connect and send SMTP commands to.
- But what’s this about a preference?

MX RECORDS

- MX records allow for enhanced mail routing functionality.
- When an email is shipped out, the server first canonicalizes the delivery address. So, for example, if test.edgecloud.com is an alias (CNAME) for edgecloud.com, then an email to instructor@test.edgecloud.com is rewritten to instructor@edgecloud.com.
- Then the server looks up the MX records for edgecloud.com, choosing the record with the **lowest** preference and attempting delivery there. If the server can not be contact, the next lowest preference server is attempted.
- This allows for backup email servers!

BACKUP EMAIL SERVERS

- But, how does the backup server handle forwarding on the message to the primary?
- Preference values to the rescue again!
- The backup server will compare it's own MX preference value with the list, and discard all records at or below it's own level, thereby eliminating the chance for mail delivery loops (assuming everything is set up correctly)
- Quite elegant, no?

BACKUP EMAIL SERVERS

- Side note: MX records should always point to canonical hostnames, in other words, names that have an A record associated with them.

- Pointing an MX to a CNAME is *possible*, but heavily frowned upon, and it will probably lead to some servers not talking to that mail server. Definitely something to double check when troubleshooting email systems.

ADDITIONAL DNS

- Remember PTR records? Used for mapping IP address domains back to their canonical hostname?
- The PTR and A records for all mail servers should match. Meaning, if mail1.edgecloud.com is an MX entry:
  - mail1.edgecloud.com -> 10.20.30.40
  - 40.30.20.10.in-addr.arpa -> mail1.edgecloud.com
- This is a simple fact that many mailservers automatically check to try and reduce spam and spoofed emails.

LAB 1

1) Using dig and nslookup, determine the mail servers for emails addressed to users at rackspace.com, edgecloud.com and gmail.com.

2) Try sending a manual email to one or more of the mail servers discovered in question 1. Does it work?

3) Verify the forward and reverse DNS lookups for the mail servers found in question 1.

SPAM

- Spam: the scourge of email
- In the old days, spam wasn't considered or even imagined. The old, solid protocol, SMTP, wasn't designed with heavy security or authentication in mind, but simplicity and reliability.
- Naturally, it didn't take long for the abuse to start, and the internet community has been fighting it ever since.
- What follows is a discussion of some of the more popular and effective spam prevention techniques in use today.

SENDER POLICY FRAMEWORK

- Otherwise known as SPF.
- SPF is actually a very simple idea, and quite effective *for the servers that implement it*.
- Basically, SPF utilizes a DNS TXT record to store a list of the servers which should be the only ones sending messages from a domain.
- Side note: Sender Id is *similar*, but not the same. SPF verifies the envelope sender (MAIL FROM:), whereas Sender ID verifies the one or more senders listed in the headers. Sender ID is often confused with SPF because it's record syntax was derived from SPF, and identifies itself with spf2.0 TXT records. Bad Microsoft. Bad!

SPF

- When a mailserver is receiving a mail message, it simply looks up the SPF information for the incoming email domain, and if the connected server isn’t listed, the email is not accepted.
- There are, of course, a few details about SPF worth mentioning..

SPFV1

- RFC 4408 is the official specification for SPFv1.
  - `http://tools.ietf.org/html/rfc4408`
- Heaps of useful information here.

SPFV1 DNS RECORDS

- The DNS records contain the configuration guiding SPF. An example will serve best:
  - `edgecloud.com. IN TXT`

    `“v=spf1 +a:mail1.edgecloud.com -all”`
- This entry says that emails coming from edgecloud.com must originate from one of the IP addresses associated with the A records for mail1.edgecloud.com. All other originators are impostors.

SPFV1 DNS RECORDS

- You will notice the use of TXT records. This is because SPF is still an *experimental* protocol. If it gets adopted, it will receive it's own DNS record type, SPF. For now, since TXT records are being used, the data has to start with "v=spf1" so mailservers can distinguish SPF records from other TXT records.
- The rest of the record is a list of allowed and denied servers. Some common tags you will see in an SPF record:

SPFV1 TAGS

- The following is a list of qualifiers for servers. Prepended to the front of a server identity, these tags control SPF behavior:

+	Pass (default)
–	Fail
~	Soft fail
?	Neutral

SPFV1 SERVER IDENTITIES

- Identifying servers with SPF configuration is quite flexible. Some common mechanisms and examples:

`a:domain_name`	`a:mail1.edgecloud.com`
`mx:domain_name`	`mx:edgecloud.com`
`ip4:ipv4_address`	`ip4:10.20.30.0/24`

SPFV1 EXTRA

- SPF even supports the ability to include spf configs from another domain, or even redirect all spf configuration details to another domain:
  - `include:domain_name   include:rackspace.com`
  - `redirect:domain_name  redirect:yahoo.com`

SPFV1 LINKS

- SPF is extremely flexible and there are some gotchas to setting up SPF. Here are a few handy links to get things rolling:
  - `http://www.openspf.org/Introduction`
  - `http://www.openspf.org/SPF_Record_Syntax`
  - `http://tools.ietf.org/html/rfc4408`

LAB 2

1) Browse the links provided earlier, covering more details of SPF. Can you see where SPF could be really effective? What is the number one drawback to SPF?

2) Look up the SPF configuration for state.tx.us. Explain the meaning of the rules in your own words. Refer to documentation if necessary.

3) Look through some personal or work emails. Examine the headers. Do you see Received-SPF? What do you think it means? Find one with a designated match and verify against DNS.

DOMAIN KEYS

- Domain Keys is a deprecated protocol, meant to reliably authenticate an email's identity using its domain and information in DNS. Domain Keys was merged with Identified Mail to create DKIM, which is the standards replacement for Domain Keys.
- There is a Historic category RFC describing the Domain Keys protocol.
  - `http://tools.ietf.org/html/rfc4870`

DOMAIN KEYS

- Some sites still run DK (Yahoo most notably, seeing as how they invented it), but most have converted, or are converting over to DKIM.
- DKIM is fully backwards compatible with DK, including the DNS configuration records (covered in next section)
- See the following link for details comparing DK and DKIM:
- `http://www.dkim.org/info/dkim-faq.html#related`

DK/DKIM HIGH LEVEL

- These two protocols transparently sign an email message as it is leaving a server. If Rackspace chose to implement DKIM, the RS Mailservers would sign and prepend the DKIM-Signature header to all outgoing emails. *This doesn't help anything by itself!*
- To complete the process, **receiving** mailservers have to see the signature header, look up the public keys in DNS, verify the signatures, and let the receiving user know that all was well (or not!) with headers. Which the user should look for, btw! *Without this part, DKIM [and DK] is impotent!*

DK HEADERS

- The DK header is “DomainKey-Signature.” Example:

```
DomainKey-Signature: a=rsa-sha1; c=nofws;
        d=gmail.com; s=gamma;
        h=mime-version:date:message-id:subject:from:to:content-type;
        b=CeZhVULWEQryIb+3k2wU5MNTuo62ooPfVTyqZXeYTlOxt65Xs2hsnl8/4HHyWb61b7
         pEm3RcWT+czwDsAU8ijH7j0rXnJrswFMkSeIs0E857UuQXVi1fCz7IToeQaQi4WhNmB4
         3ra/2hh5VpdD7j7DnDCmZ9kSv/+wBIBkD3RJQ=
```

- ➥RFC 4870 §3.3
- To look up the public key, perform a TXT query on:
  - `<`*`s`*`>._domainkey.<`*`d`*`>`
  - `gamma._domainkey.gmail.com`

```
dig gamma._domainkey.gmail.com txt

; <<>> DiG 9.6.0-APPLE-P2 <<>> gamma._domainkey.gmail.com txt
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 57517
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0,
ADDITIONAL: 0

;; QUESTION SECTION:
;gamma._domainkey.gmail.com. IN TXT

;; ANSWER SECTION:
gamma._domainkey.gmail.com. 300   IN TXT "k=rsa\; t=y\;
p=MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDIhyR3oItOy22ZOaBrIVe9m/
iME3RqOJeasANSpg2YTHTYV
+Xtp4xwf5gTjCmHQEMOs0qYu0FYiNQPQogJ2t0Mfx9zNu06rfRBDjiIU9tpx2T
+NGlWZ8qhbiLo5By8apJavLyqTLavyPSrvsx0B3YzC63T4Age2CDqZYA
+OwSMWQIDAQAB"

;; Query time: 65 msec
;; SERVER: 192.168.1.1#53(192.168.1.1)
;; WHEN: Fri Dec 24 14:19:38 2010
;; MSG SIZE  rcvd: 287
```

DK VERIFICATION

- Verifying DK signatures manually is an exceptionally involved process. Fortunately, there's a better option:
- Look for the "Authentication-Results" header. ➥RFC 5451
- Most mail systems use this header to list results for SPF, DK and DKIM authentication tests. Example:
- `Authentication-Results: mr.google.com;` **`spf=pass`** `(google.com: domain of xxxxxxxxx@gmail.com designates 10.216.13.194 as permitted sender) smtp.mail=xxxxxxxxx@gmail.com;` **`dkim=pass`** `header.i=xxxxxxxxx@gmail.com`

DKIM

- Domain Keys Identified Mail. The standards successor to DK, and actually a much improved protocol.
  - `http://tools.ietf.org/html/rfc4871`
- DKIM closely resembles DK in it's operation and configuration - there are minor differences which serve to improve the effectiveness of the protocol. A couple of important changes:
  - The DKIM signature header is included in the signing process to prevent tampering
  - DKIM can sign the body of the message

DKIM HEADERS

- The DKIM header is “DKIM-Signature.” Example:

```
DKIM-Signature: v=1; a=rsa-sha256; c=relaxed/relaxed; d=yahoo.com; s=s1024; t=1293157230;
bh=iKTTWPnEnpqSGt2IXyu5juE5NZRvwAFda4KBcNIY+ks=; h=Message-ID:X-YMail-OSG:Received:X-
Mailer:Date:From:Subject:To:MIME-Version:Content-Type; b=jRUlKpqN2IQ6SvQucDK2guUcFbY3dM/
ufvz/mZJIKT4TjJW8F+mHaA
+51HVfwDNfSUbRHGcKYVhdaqnkhNH1tJpazMXA45OWKX4iGV4ylc1uJ8nzYRanDzWNk2+TSR1ZEER1qoqNtzhHz9fSC
sWO+sFPcSCU8l+KWEhmBYaN91A=
```

- ➥RFC 4871 §3.5
- To look up the public key, perform a TXT query on:
  - `<`*s*`>._domainkey.<`*d*`>`
  - `s1024._domainkey.yahoo.com`

```
dig s1024._domainkey.yahoo.com txt

; <<>> DiG 9.6.0-APPLE-P2 <<>> s1024._domainkey.yahoo.com txt
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 38271
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0,
ADDITIONAL: 0

;; QUESTION SECTION:
;s1024._domainkey.yahoo.com. IN TXT

;; ANSWER SECTION:
s1024._domainkey.yahoo.com. 86400 IN TXT "k=rsa\; t=y\;
p=MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDrEee0Ri4Juz+QfiWYui/
E9UGSXau/2P8LjnTD8V4Unn+2FAZVGE3kL23bzeoULYv4PeleB3gfm"
"JiDJOKU3Ns5L4KJAUUHjFwDebt0NP+sBK0VKeTATL2Yr/S3bT/xhy
+1xtj4RkdV7fVxTn56Lb4udUnwuxK4V5b5PdOKj/+XcwIDAQAB\; n=A 1024
bit key\;"

;; Query time: 33 msec
;; SERVER: 192.168.1.1#53(192.168.1.1)
;; WHEN: Fri Dec 24 14:42:01 2010
;; MSG SIZE  rcvd: 307
```

DKIM VERIFICATION

- Verifying DKIM signatures manually is just as much a PITA as DK.
- Again, just look for the “Authentication-Results” header.
  - ➥RFC 5451
- `Authentication-Results: mr.google.com; `**`spf=pass`**` (google.com: domain of xxxxxxxxx@gmail.com designates 10.216.13.194 as permitted sender) smtp.mail=xxxxxxxxx@gmail.com; `**`dkim=pass`**` header.i=xxxxxxxxx@gmail.com`

LAB 3

1) Browse through the various RFC's and websites documenting DK and DKIM.

2) Look through some personal or work emails. Examine the headers. Do you see any DK or DKIM signature headers? How about Authentication-Results headers? Is there always going to be an Auth Results header? Why or Why not?

3) Manually look up the public keys associated with 3 of the signed emails in your inbox.

EMAIL BLACKLISTS

- An email blacklist is a collection of IP addresses for servers known to send spam email.
- Blacklists are used by some mailservers to try and combat spam, by either rejecting entirely any message a blacklisted server attempts to send, or by bumping up a “spam score” on the messages received from the blacklisted server.
- Most blacklists are stored in the DNS database.
- `http://en.wikipedia.org/wiki/Comparison_of_DNS_blacklists`

POPULAR BLACKLISTS

- **SORBS** - Spam and Open Relay Blocking System
  - `http://www.sorbs.net/`
- **Spamhaus**
  - `http://www.spamhaus.org/`
- **SpamCop**
  - `http://spamcop.net/`
- **RFC-Ignorant**
  - `http://rfc-ignorant.org/`

CHECKING BLACKLISTS

- Checking for listing in a black list can be done in one of two ways:
  - Web based lookup tool - either on the specific blacklist website, or using a multi-lookup tool like:
    - `http://multirbl.valli.org/`
  - DNS

CHECKING BLACKLISTS

- Checking a website is simple. But checking DNS is slightly more involved. Following is a description for most blacklists:
  - First, invert the octets from the IP address:
    - `172.172.92.137  ->  137.92.172.173`
  - Then, append the domain name for the spam list:
    - `137.92.172.173.zen.spamhaus.org`
  - Then, do an A record lookup. If it exists, the IP is listed.

GREYLISTS

- Greylists are used to fight spam by temporarily deferring the first email from an unknown sender.
- The idea is that most spam engines will not bother retrying a temporary deferral - they will just move on to the next victim.
- Yahoo is one of the most notable employers of a slight variation on greylisting. Yahoo, being such an enormous email provider, maintains internal blacklists, whitelists and more. When a particular sender starts looking suspicious, Yahoo will start temporarily deferring their emails.

TAR PITS

- **Tarpitting** - A clever idea used to slow down spammer sending processes. There are many implementations, but the basic idea is to greatly slow down the SMTP sequence, particularly if a host is known to be a spammer. Other common implementations include a mechanism that tracks, per sender, the number of emails sent per unit time. As the rate increases, the SMTP server slows down.

OTHER SPAM ISSUES

- **Botnets** - A Botnet is a collection of compromised servers being controlled by a single entity (or person), generally to make money, do harm, etc. A spamming botnet is an obvious way to make money and get around many of the spam prevention techniques.

- **Open Relay** - An open relay is a mailserver that accepts email from anywhere, destined to anywhere. This is not how a mailserver should work - either it accepts email for anywhere, from only certain hosts (ISP mailserver for customers), or it accepts email from anywhere, destined to particular mailboxes (MX server). Open relays allow spammers to funnel messages and bypass many of the spam prevention techniques, and therefore open relays often get blacklisted.

OTHER SPAM ISSUES

- **Smart Host** - A smart host is a relay mailserver which is configured to accept mail messages and forward them on to another relay mailserver for additional processing and delivery. Smart hosts do not perform MX lookups or any other delivery processing, they merely pass the message along to another mail server for handling. Since smart hosts are in the same thread as open relays, they often have message acceptance requirements to limit abuse (host based access controls, authentication, etc).

EMAIL PROVIDERS AND SPAM

- Most of the major email providers (yahoo, gmail, hotmail, etc) employ a variety of spam prevention techniques.
  - Blacklists, greylists, whitelists - both internal and external
  - Tarpitting
  - Content analysis using keyword and bayesian filters
  - Advanced traffic analysis
  - Dedicated spam prevention departments

LAB 4

1) Look up `173.172.92.137` in at least 3 different blacklists. Perform lookups via web tools and DNS directly.

2) Browse the Wikipedia article on Anti-spam techniques:

   `http://en.wikipedia.org/wiki/Anti-spam_techniques`

```
slideshow.end();
```

ADVANCED PROTOCOL

SMTP AUTH, encryption and other assorted goodies

ADDITIONAL SMTP COMMANDS

- The rest of the SMTP commands:
  - `RSET` - Reset current mail transaction - discard all data
  - `VRFY` - Verify the given user or mailbox is valid*
  - `EXPN` - Expand the given user or mailbox mailing list into the list of recipients*
  - `HELP` - Request usage help from the server*
  - `NOOP` - No operation - basically a keep-alive

* Often disabled or not implemented

"REMOTE" SMTP SERVICE

- Recall from the previous lecture the discussion on open relays.
- An open relay is definitely a Bad Thing, but sometimes email service needs to be provided to clients that can't be identified solely by inbound IP address. Consider traveling employees, hosting customers, etc. How can email service be offered without creating an open relay?
- SMTP AUTH

SMTP AUTH

- SMTP Authentication is an SMTP protocol extension designed to allow for authenticated SMTP sessions. Using an authentication process allows a mailserver to identify a client and grant full email relaying privilege to that session. More RFC's!
- RFC 4954 describes SMTP AUTH:
  - `http://tools.ietf.org/html/rfc4954`
- SASL - Simple Authentication and Security Layer
  - `http://tools.ietf.org/html/rfc4422`

SMTP AUTH

- The SMTP Auth RFC describes the generic process by which clients can authenticate themselves to the server, namely by using SASL mechanisms. It does not describe the details of the authentication process itself. This RFC calls for the `AUTH` keyword in the `EHLO` response.

- The SASL RFC describes a very generic authentication procedure which can be used by many different network services to handle authentication. This RFC also does not describe the details of specific authentication processes itself. =)

SMTP AUTH

- Between these two RFC's, mailservers can now advertise their authentication support via the EHLO extension list response:
  - `250-AUTH PLAIN LOGIN`
- As a side note, you will often see an "AUTH" line as well as an "AUTH=" line in the EHLO response. This is for backwards compatibility with some very old mail clients which did not correctly implement the RFC for SMTP AUTH. See:
- `http://www.postfix.org/postconf.5.html#broken_sasl_auth_clients`

AUTH PLAIN

- RFC 4616 is the first SASL authentication process we will discuss:
    - `http://tools.ietf.org/html/rfc4616`
- RFC 4616 describes the PLAIN SASL authentication mechanism, more commonly known as "AUTH PLAIN".
- If the `EHLO` response line for "AUTH" includes the word "PLAIN", then the AUTH PLAIN mechanism is supported by that server.

BASE 64 ENCODING

- Before we can discuss AUTH PLAIN, we need to introduce Base64 encoding
- Base64 encoding is a method of translating arbitrary data into printable characters, usually due to legacy constraints in a system requiring US ASCII characters only.
- The Base64 RFC:
  - `http://tools.ietf.org/html/rfc4648`

BASE 64 ENCODING

- Base64 encoding can be done in several ways. Perl:
  - `perl -MMIME::Base64 -e 'print encode_base64("Encode me")'`
  - Make sure to escape out @ and $ symbols in string. See Example.
- Websites: google for base64 encoder or similar
- By hand...
- Tools: consult a google

AUTH PLAIN

- Now that we can encode values with Base64, we can finish our discussion of AUTH PLAIN.
- To authenticate using AUTH PLAIN:
  - `AUTH PLAIN <`*`credentials`*`>`
- Where <*`credentials`*> is a Base64 encoded string of the form:
  - `<NULL>username<NULL>password`

```
$ perl -MMIME::Base64 -e 'print encode_base64("\000user
\000password")'
AHVzZXIAcGFzc3dvcmQ=
$ telnet mail.example.com 25
Trying 10.20.30.40...
Connected to mail.example.com.
Escape character is '^]'.
220 mail.example.com ESMTP
ehlo test
250-mail.example.com
250-PIPELINING
250-SIZE 40960000
250-ETRN
250-STARTTLS
250-AUTH PLAIN LOGIN
250-AUTH=PLAIN LOGIN
250-ENHANCEDSTATUSCODES
250 8BITMIME
auth plain AHVzZXIAcGFzc3dvcmQ=
235 2.7.0 Authentication successful
quit
221 2.0.0 Bye
Connection closed by foreign host.
```

AUTH LOGIN

- Now that we have covered one of the authentication mechanisms, and learned about Base64, the other authentication mechanisms will follow easily.
- AUTH LOGIN is similar to AUTH PLAIN, except that the username and password are supplied (Base64 encoded) in individual commands.
- An example will best demonstrate...

```
$ perl -MMIME::Base64 -e 'print encode_base64("user")'
dXNlcg==
$ perl -MMIME::Base64 -e 'print encode_base64("password")'
cGFzc3dvcmQ=
$ telnet mail.example.com 25
Trying 10.20.30.40...
Connected to mail.example.com.
Escape character is '^]'.
220 mail.example.com ESMTP
ehlo test
250-mail.example.com
250-PIPELINING
250-SIZE 40960000
250-ETRN
250-STARTTLS
250-AUTH PLAIN LOGIN
250-AUTH=PLAIN LOGIN
250-ENHANCEDSTATUSCODES
250 8BITMIME
auth login
334 VXNlcm5hbWU6
dXNlcg==
334 UGFzc3dvcmQ6
cGFzc3dvcmQ=
235 2.7.0 Authentication successful
quit
221 2.0.0 Bye
Connection closed by foreign host.
```

LAB 1

1) Perform an AUTH PLAIN login to server1.example.com with your stationX credentials.

2) Use AUTH LOGIN to login to server1.example.com with the same credentials.

3) Decode the Base64 responses the server uses on AUTH LOGIN.

AUTH CRAM-MD5

- AUTH CRAM-MD5 is yet another authentication mechanism. It's quite a bit more complex, and generally only manually tested using a tool to generate the credentials.
- CRAM - Challenge/Response Authentication Mechanism.
- MD5 - Message Digest algorithm 5
- Thanks to the use of the MD5 algorithm, CRAM-MD5 is a much more secure form of authentication, because the password is hashed before being sent on the wire. CRAM-MD5 RFC:
  - `http://tools.ietf.org/html/rfc2195`

AUTH CRAM-MD5

- After the client sends the “AUTH CRAM-MD5” command, the server responds with a Base64 encoded challenge.
- The client decodes the challenge, and incorporates it into the response. This particular method is known as “Keyed hashing” and is fully described in RFC 2104:
  - `http://tools.ietf.org/html/rfc2104`

AUTH CRAM-MD5

- The response is the Base64 encoding of:
  - `<user><space><md5-response>`
- Where `<user>` is the username, and `<md5-response>` is:
- `MD5(('password' XOR opad), MD5(('password' XOR ipad), challenge))`
- Definitions for ipad and opad: ➥RFC 2104 §2

AUTH CRAM-MD5

- As you can see, CRAM-MD5 is a bit more tedious if you were to calculate everything by hand. If CRAM-MD5 needs to be tested/verified, it is best to use a tool such as `gen-auth`:
  - `http://www.jetmore.org/john/code/gen-auth`

ENCRYPTION

- Up until this point, all traffic exchanged between clients and servers have been in clear text, or unencrypted form.
- This is fine for many situations, but sometimes additional security is necessary, and for that, there is an SMTP extension for tunneling the SMTP session inside an encrypted TLS tunnel.
    - `http://tools.ietf.org/html/rfc3207`

TLS

- TLS stands for Transport Layer Security, and is a protocol used to encrypt all traffic above the Transport layer.
- This makes TLS perfect as an add-on for SMTP, because once the TLS session is negotiated, all communication continues as defined in the RFC.
- If the word “SMARTTLS” is included in the EHLO responses, then the server supports encrypted SMTP via TLS.

TESTING TLS

- The only way to manually test a TLS enabled mailserver is to use a tool that will manage the TLS layer behind the scenes, passing the decrypted information back to the screen:

-
  ```
  openssl s_client -starttls smtp -crlf -connect 10.20.30.40:25
  ```

- This tells openssl to start a secure client (`s_client`) session, allowing for a roughly telnet-like interface to the user, but encrypting the communication using TLS.

- `-starttls` tells openssl to connect in plain text initially, then switch to TLS using the appropriate commands for the given protocol (SMTP).

- `-crlf` tells openssl to convert bare line feeds into carriage return line feeds.

LAB 2

1) Find a mail server that supports TLS and initiate a secure connection to that server. Verify connectivity by issuing a few SMTP commands.

FIXUP

- The Fixup feature found on many Cisco devices can be a really helpful tool, because it can dynamically rewrite some application layer messages to have the correct network information in them (think NAT-ing) so the protocol will function through firewalls and translating routers.
- The problem is, the Fixup implementation for SMTP only works for the original, basic SMTP protocol. Not ESMTP, which is where we find all of our nifty features like TLS and AUTH.

FIXUP

- If you are testing a mail server and see things like
  - 220 *******************
- Then you have a mail server behind a Cisco device that's running the Fixup feature, and it will hose most SMTP traffic.
- A quick google search will turn up copious documentation about this issue.
- **Bottom line: Do not use the Fixup feature with SMTP.**

```
slideshow.end();
```

ADVANCED HEADERS

MIME! X-Headers!

MIME

- Email messages and SMTP were designed to move *textual* information around the network. And at first, this was all that was necessary.
- Over time, though, transmitting different forms of data became important. Being able to email files, pictures or anything else proved to be an important function.
- But how to transmit raw binary data through a simple text-based protocol?
- MIME! Multipurpose Internet Mail Extensions!

RFC MADNESS!

- MIME has *several* RFC's:
  - `http://tools.ietf.org/html/rfc2045`
  - `http://tools.ietf.org/html/rfc2046`
  - `http://tools.ietf.org/html/rfc2047`
  - `http://tools.ietf.org/html/rfc4288`
  - `http://tools.ietf.org/html/rfc4289`
  - `http://tools.ietf.org/html/rfc2049`

MIME

- Now, if you were going to implement a MIME library to create and parse MIME content, then you would need to dig through every single one of those RFC's with a fine-toothed comb.
- Fortunately, in this class, we are only seeking a basic understanding of MIME. :)

MIME HEADERS

- The first header to look for is the MIME-Version header. The presence of this header indicates that the message is formatted to the MIME specification.
  - `MIME-Version: 1.0`
- Note that the version number will always be 1.0 due to some issues that were encountered when an attempted change was made. There have, of course, been many updates and revisions to the MIME protocol.

MIME HEADERS

- The Content-Type header is a very important one, used to describe the contents of the message. The most basic is “text/plain” as:
  - `Content-Type: text/plain`
- This is interpreted to mean that the contents of the rest of the message are in a plain text format.
- This header allows for the most important aspect of MIME messages: the Multipart message!

MULTIPART MIME

- `Content-Type: multipart/mixed`
- The multipart/mixed content type declares the message to contain various, mixed parts. Usually a text/plain part, with one or more other parts.
- `multipart/alternative` is another multipart content type. The alternative indicates that there are alternative forms of the message which can be displayed, based on user preference. Usually a text/plain and a text/html.

MULTIPART MIME

- When a multipart message is created, an additional field is added to the content-type header: the boundary.
- `Content-Type: multipart/mixed; boundary=”here”`
- The boundary is used to segment each of the parts of the multipart message. The last or ending boundary will have “--” appended to indicate it is the final boundary marker.
- At each boundary, additional Content-* headers can be added.
- An example illustrates these concepts...

```
To: <test@example.com>
...More Regular Headers...
MIME-Version: 1.0
Content-Type: multipart/alternative;
    boundary="----=_Part_8571692_700621548.1293443597341"

------=_Part_8571692_700621548.1293443597341
Content-Type: text/plain; charset=UTF-8
Content-Transfer-Encoding: 7bit

Text/plain message here!

------=_Part_8571692_700621548.1293443597341
Content-Type: text/html; charset=UTF-8
Content-Transfer-Encoding: 7bit

<html>
  <body>
Text/html message here!
  </body>
</html>
------=_Part_8571692_700621548.1293443597341--
```

CONTENT TRANSFER ENCODING

- The Content-Transfer-Encoding header is used to identify the binary to ascii encoding scheme used for the data. Most common:
  - `7bit` - Default for plain text - human readable
  - `quoted-printable` - Similar to 7bit, but supports a few more characters - mostly human readable
  - `base64` - Encodes arbitrary data - not human readable.

```
To: <test@example.com>
...More Regular Headers...
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary="----=_Part_4_3415473.1293093601660"

------=_Part_4_3415473.1293093601660
Content-Type: multipart/alternative;
    boundary="----=_Part_3_32680187.1293093601660"

------=_Part_3_32680187.1293093601660
Content-Type: text/plain; charset=us-ascii
Content-Transfer-Encoding: 7bit

The quick brown fox!

------=_Part_3_32680187.1293093601660
Content-Type: text/html; charset=us-ascii
Content-Transfer-Encoding: 7bit

<html>
<body>
More quick brown foxen!
</body>
</html>

------=_Part_3_32680187.1293093601660--

------=_Part_4_3415473.1293093601660
Content-Type: application/pdf; name=attachment-file.pdf
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename=attachment-file.pdf

JVBERi0xLjAKMSAwIG9iaiA8PCAvVHlwZSAvQ2F0YWxvZyAvUGFnZXMgMiAwIFIgPj4gZW5kb2Jq
CjIgMCBvYmogPDwgL1R5cGUgL1BhZ2VzIC9Db3VudCAxIC9NZWRpYUJveCBbIDAgMCA2MTIgNzky
.....  SNIP  ......
MjcgMDAwMDAgbiAKMDAwMDAwNTY3NSAwMDAwMCBuIAowMDAwMDA1ODQ1IDAwMDAwIG4gCjAwMDAw
MDU5NjQgMDAwMDAgbiAKdHJhaWxlcgogPDwgL1NpemUgMTQgL1Jvb3QgMSAwIFIgPj4Kc3RhcnR4
cmVmCjM1OTc3CiUlRU9GCg==
------=_Part_4_3415473.1293093601660--
```

Embedded multipart/alternative part. So flexible! Notice new boundary?

CONTENT DISPOSITION

- The Content-Disposition header is used to mark parts as attachments (not displayed directly in the message) or inline (displayed inline with the message body)
- Note that many MUA's will not, by default, display inline content for security reasons.
- For attachment disposed content, a filename is often provided in the header for a suggested name if the attachment is to be saved locally.

ADDITIONAL CONTENT TYPES

- Some of the other common content types include:
  - `image/jpeg`
  - `application/octet-stream`
  - `application/ms-word`
  - `audio/mp3`
- Lists and definitions can be found in the RFC’s, or google

X-HEADERS

- Finally, the last header that merits some discussion is the X-header.
- Actually, there are lots X Headers. Because X Headers were created to allow for arbitrary headers.
- Any header starting with "X-" is an X Header, and is not officially defined by an RFC or any other standard, though googling will produce lists of common X Headers.
- Trivia: The original RFC for messages (822) declared the prefix "X-" to be off limits for defined names, and therefore usable privately. Updates to that RFC do not include the declaration, but the "X-" names stuck and are commonly used everywhere

LAB 1

1) Open a text editor and manually piece together a MIME message containing a text part and an html part. Include an X header that you make up. Manually send it to a personal email address you have access to from the classroom. Try directly sending to the MX host specified for your domain.

2) Check your personal email and verify the MIME parts. Also, check for your X header.

```
slideshow.end();
```