Lab 4: HTTP Basic Authentication

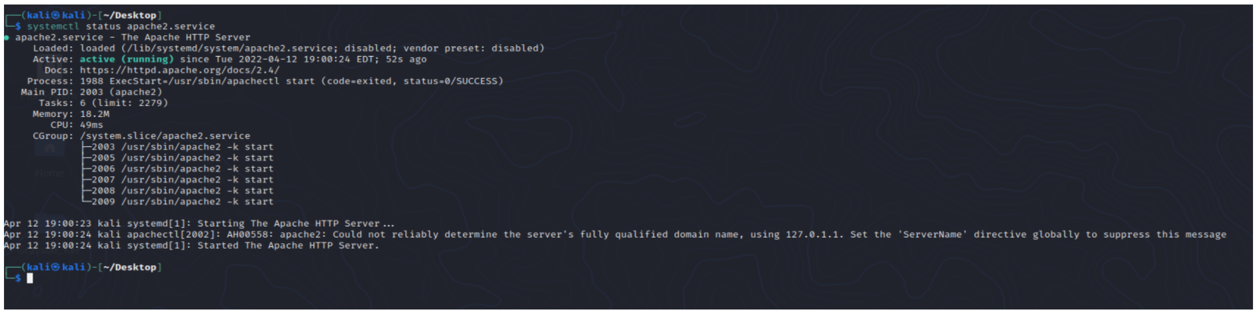
<u>Introduction:</u>

The objective of this lab was to create and configure a personal apache2 HTTP server. Apache is a very common web server software that is widely used in industry. We will see how to configure authentication for these servers, and how such servers are vulnerable to packet sniffers such as wireshark.

<u>Part 1: Starting the Server</u>

The apache2 service comes preinstalled on our kali VM, but can be easily installed from the command line with sudo privileges. We start the server using the `service apache2 start` command.

Figure 1: Running apache2 Server



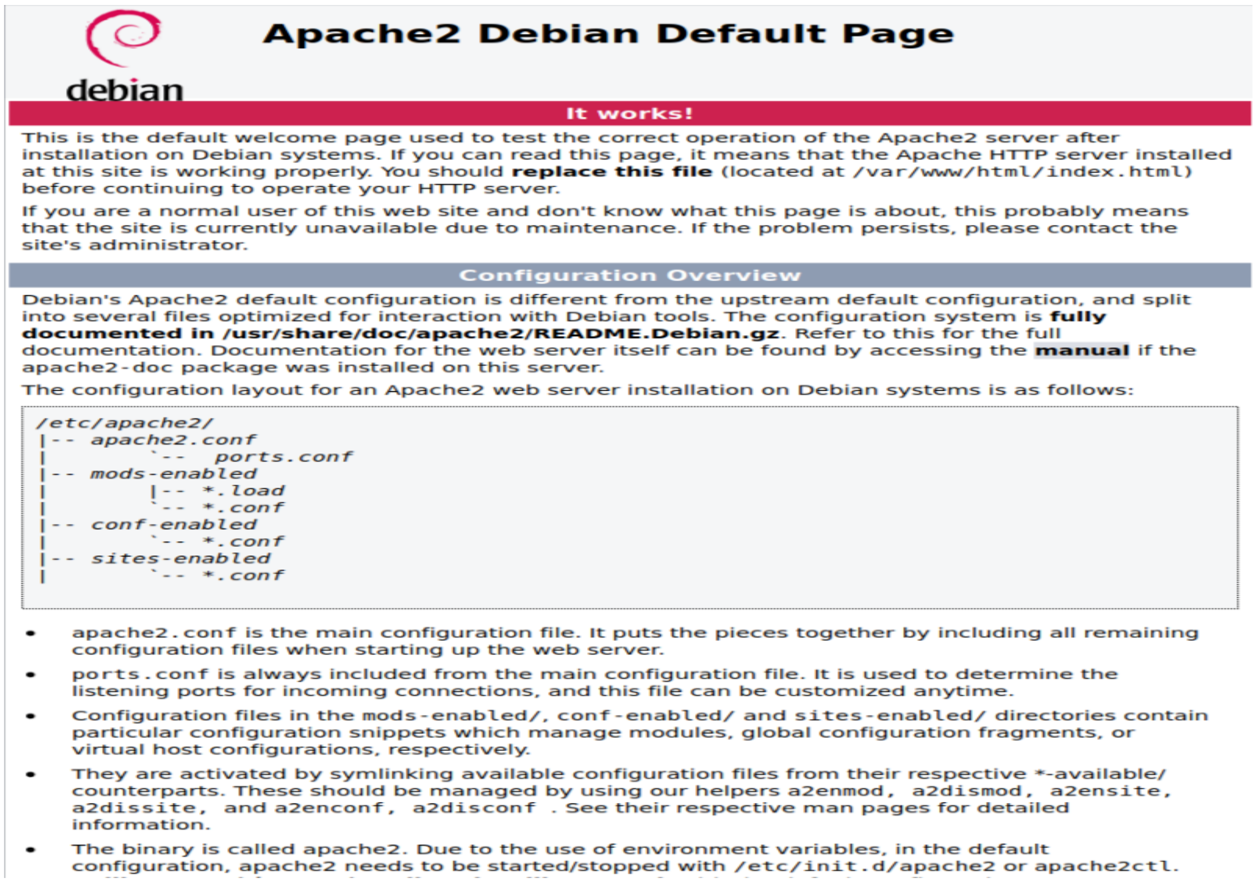
```
┌──(kali㉿kali)-[~/Desktop]
└─$ systemctl status apache2.service
● apache2.service - The Apache HTTP Server
     Loaded: loaded (/lib/systemd/system/apache2.service; disabled; vendor preset: disabled)
     Active: active (running) since Tue 2022-04-12 19:00:24 EDT; 52s ago
       Docs: https://httpd.apache.org/docs/2.4/
    Process: 1988 ExecStart=/usr/sbin/apachectl start (code=exited, status=0/SUCCESS)
   Main PID: 2003 (apache2)
      Tasks: 6 (limit: 2279)
     Memory: 18.2M
        CPU: 49ms
     CGroup: /system.slice/apache2.service
             ├─2003 /usr/sbin/apache2 -k start
             ├─2005 /usr/sbin/apache2 -k start
             ├─2006 /usr/sbin/apache2 -k start
             ├─2007 /usr/sbin/apache2 -k start
             ├─2008 /usr/sbin/apache2 -k start
             └─2009 /usr/sbin/apache2 -k start

Apr 12 19:00:23 kali systemd[1]: Starting The Apache HTTP Server...
Apr 12 19:00:24 kali apachectl[2002]: AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 127.0.1.1. Set the 'ServerName' directive globally to suppress this message
Apr 12 19:00:24 kali systemd[1]: Started The Apache HTTP Server.

┌──(kali㉿kali)-[~/Desktop]
└─$ 
```

Now when we open Firefox with the and search for the ip address `192.168.0.10` of our VM in the URL bar, we get the following web page.

Figure 2: Default apache2 web page



debian

Apache2 Debian Default Page

It works!

This is the default welcome page used to test the correct operation of the Apache2 server after installation on Debian systems. If you can read this page, it means that the Apache HTTP server installed at this site is working properly. You should **replace this file** (located at /var/www/html/index.html) before continuing to operate your HTTP server.

If you are a normal user of this web site and don't know what this page is about, this probably means that the site is currently unavailable due to maintenance. If the problem persists, please contact the site's administrator.

Configuration Overview

Debian's Apache2 default configuration is different from the upstream default configuration, and split into several files optimized for interaction with Debian tools. The configuration system is **fully documented in /usr/share/doc/apache2/README.Debian.gz**. Refer to this for the full documentation. Documentation for the web server itself can be found by accessing the **manual** if the apache2-doc package was installed on this server.

The configuration layout for an Apache2 web server installation on Debian systems is as follows:

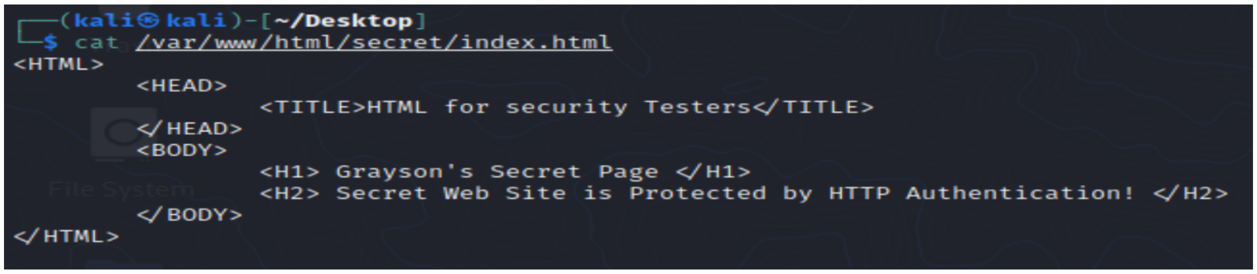
```
/etc/apache2/
|-- apache2.conf
|       `--  ports.conf
|-- mods-enabled
|       |-- *.load
|       `-- *.conf
|-- conf-enabled
|       `-- *.conf
|-- sites-enabled
|       `-- *.conf
```

- apache2.conf is the main configuration file. It puts the pieces together by including all remaining configuration files when starting up the web server.
- ports.conf is always included from the main configuration file. It is used to determine the listening ports for incoming connections, and this file can be customized anytime.
- Configuration files in the mods-enabled/, conf-enabled/ and sites-enabled/ directories contain particular configuration snippets which manage modules, global configuration fragments, or virtual host configurations, respectively.
- They are activated by symlinking available configuration files from their respective *-available/ counterparts. These should be managed by using our helpers a2enmod, a2dismod, a2ensite, a2dissite, and a2enconf, a2disconf . See their respective man pages for detailed information.
- The binary is called apache2. Due to the use of environment variables, in the default configuration, apache2 needs to be started/stopped with /etc/init.d/apache2 or apache2ctl.

This provides us with all the information we need to configure the authentication of the server.

We add a new directory `secret` to the website and provide it with the following HTML code

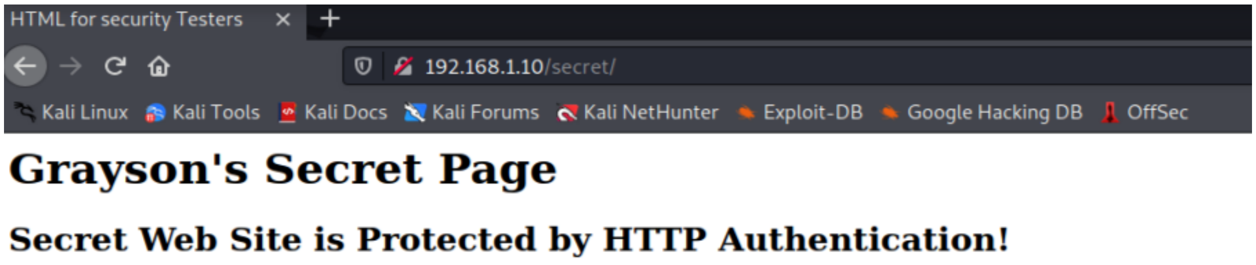
Figure 3: Secret web page HTML code



```
┌──(kali㉿kali)-[~/Desktop]
└─$ cat /var/www/html/secret/index.html
<HTML>
        <HEAD>
                <TITLE>HTML for security Testers</TITLE>
        </HEAD>
        <BODY>
                <H1> Grayson's Secret Page </H1>
                <H2> Secret Web Site is Protected by HTTP Authentication! </H2>
        </BODY>
</HTML>
```

We can navigate to this page in firefox by searching for `192.168.1.10/secret` in the URL search bar. The result is shown below.

Figure 4: Secret web page

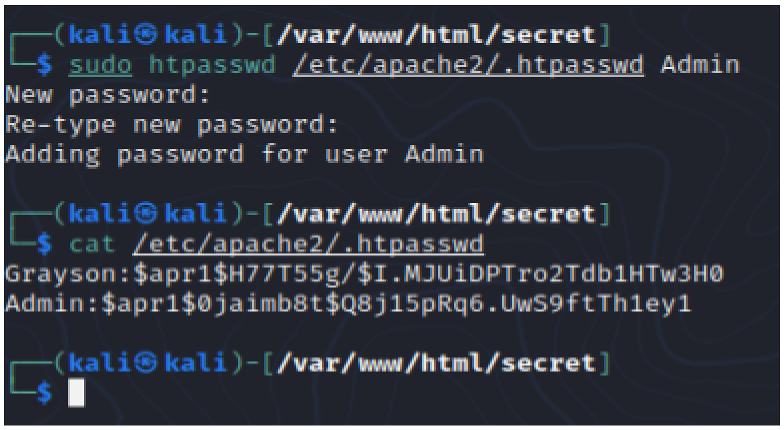


Grayson's Secret Page

Secret Web Site is Protected by HTTP Authentication!

To configure the authentication of this page, we need to add a few files. Namely `.htpasswd` to store whitelisted users and their passwords, and another `002-secret.conf` to configure the authentication protocol. The contents of both these files are shown below.

Figure 5: Authorized Users and Passwords

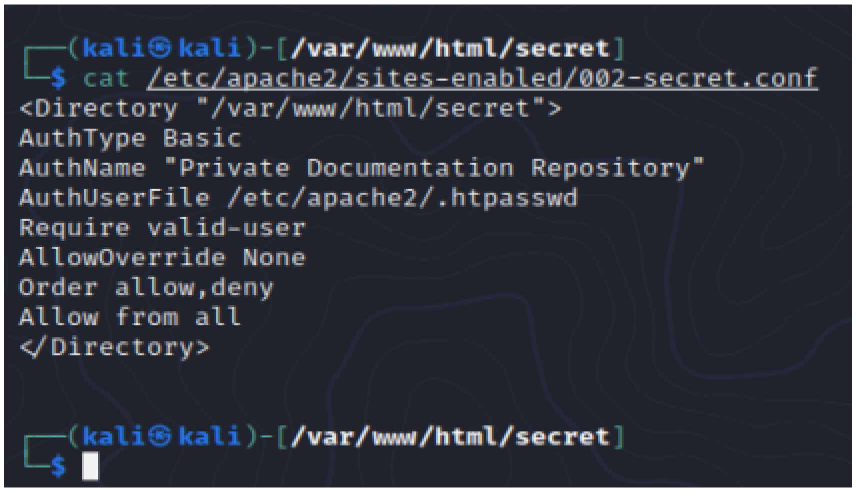


```
┌──(kali㉿kali)-[/var/www/html/secret]
└─$ sudo htpasswd /etc/apache2/.htpasswd Admin
New password:
Re-type new password:
Adding password for user Admin

┌──(kali㉿kali)-[/var/www/html/secret]
└─$ cat /etc/apache2/.htpasswd
Grayson:$apr1$H77T55g/$I.MJUiDPTro2Tdb1HTw3H0
Admin:$apr1$0jaimb8t$Q8j15pRq6.UwS9ftTh1ey1

┌──(kali㉿kali)-[/var/www/html/secret]
└─$ █
```

Figure 6: Authentication Code For Secret Page

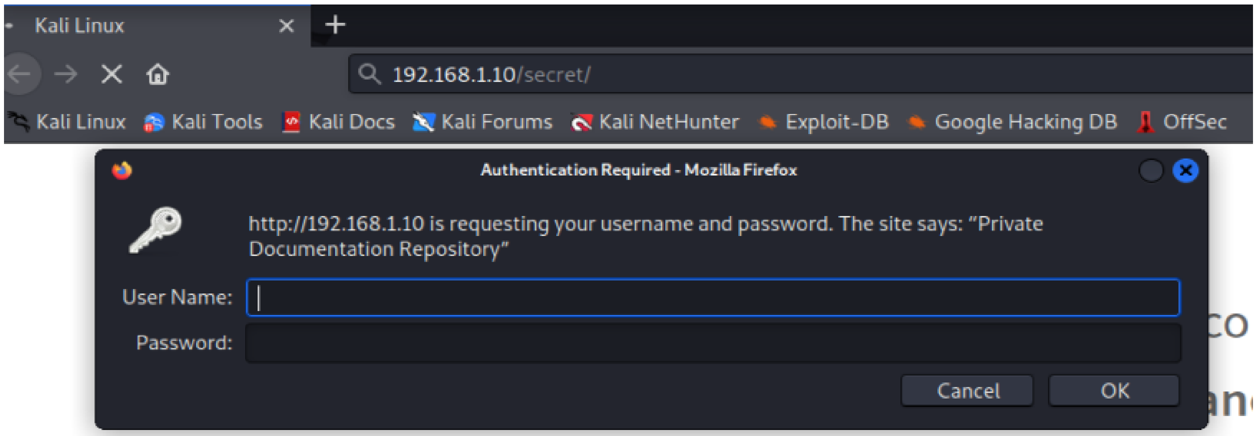


```
┌──(kali㉿kali)-[/var/www/html/secret]
└─$ cat /etc/apache2/sites-enabled/002-secret.conf
<Directory "/var/www/html/secret">
AuthType Basic
AuthName "Private Documentation Repository"
AuthUserFile /etc/apache2/.htpasswd
Require valid-user
AllowOverride None
Order allow,deny
Allow from all
</Directory>

┌──(kali㉿kali)-[/var/www/html/secret]
└─$ 
```

Now when we restart the apache2 server and attempt to navigate to our secret page, Firefox asks for authentication credentials.

Figure 7: Secret Page Authentication Prompt



Authentication Required - Mozilla Firefox

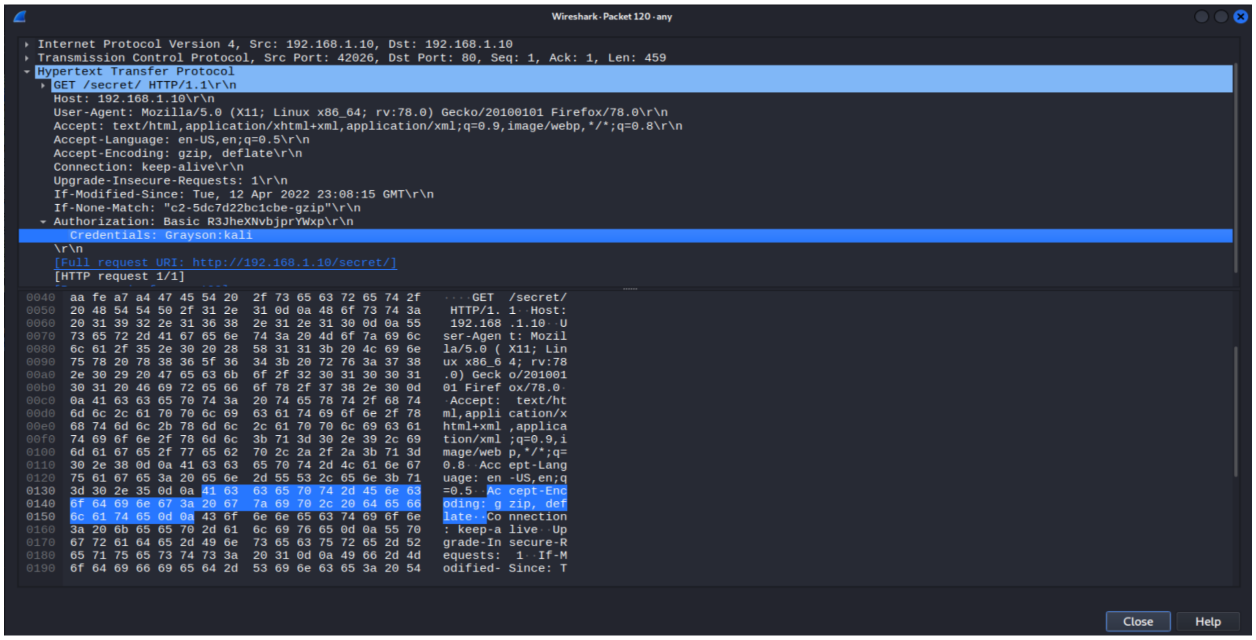
http://192.168.1.10 is requesting your username and password. The site says: "Private Documentation Repository"

User Name:

Password:

We then start Wireshark and configure it to listen to our VM's network before supplying our credentials to view the page. When we login, an HTTP GET request is sent to the server, which Wireshark allows us to capture and analyze.

Figure 8: Capturing HTTP Request With Wireshark



```
▸ Internet Protocol Version 4, Src: 192.168.1.10, Dst: 192.168.1.10
▸ Transmission Control Protocol, Src Port: 42026, Dst Port: 80, Seq: 1, Ack: 1, Len: 459
▾ Hypertext Transfer Protocol
  ▸ GET /secret/ HTTP/1.1\r\n
    Host: 192.168.1.10\r\n
    User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101 Firefox/78.0\r\n
    Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8\r\n
    Accept-Language: en-US,en;q=0.5\r\n
    Accept-Encoding: gzip, deflate\r\n
    Connection: keep-alive\r\n
    Upgrade-Insecure-Requests: 1\r\n
    If-Modified-Since: Tue, 12 Apr 2022 23:08:15 GMT\r\n
    If-None-Match: "c2-5dc7d22bc1cbe-gzip"\r\n
  ▾ Authorization: Basic R3JheXNvbjprYWxp\r\n
      Credentials: Grayson:kali
    \r\n
    [Full request URI: http://192.168.1.10/secret/]
    [HTTP request 1/1]
```

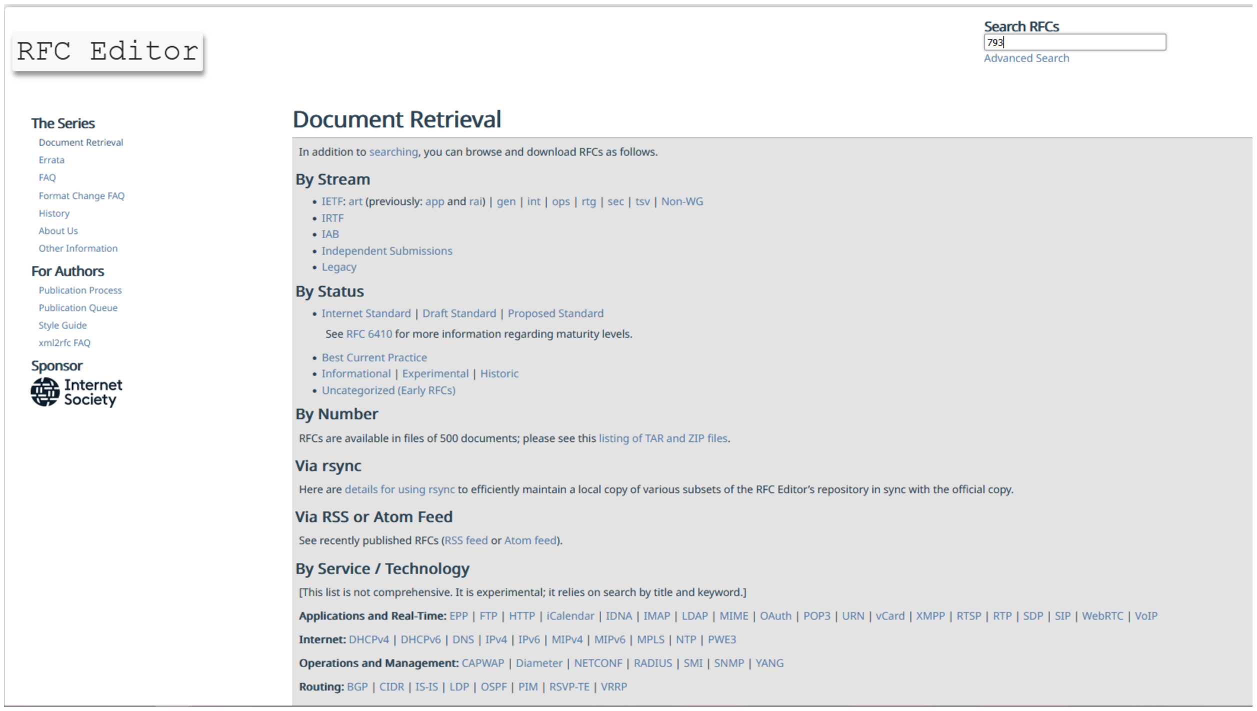
```
0040  aa fe a7 a4 47 45 54 20  2f 73 65 63 72 65 74 2f   ····GET  /secret/
0050  20 48 54 54 50 2f 31 2e  31 0d 0a 48 6f 73 74 3a    HTTP/1. 1··Host:
0060  20 31 39 32 2e 31 36 38  2e 31 2e 31 30 0d 0a 55    192.168 .1.10··U
0070  73 65 72 2d 41 67 65 6e  74 3a 20 4d 6f 7a 69 6c   ser-Agen t: Mozil
0080  6c 61 2f 35 2e 30 20 28  58 31 31 3b 20 4c 69 6e   la/5.0 ( X11; Lin
0090  75 78 20 78 38 36 5f 36  34 3b 20 72 76 3a 37 38   ux x86_6 4; rv:78
00a0  2e 30 29 20 47 65 63 6b  6f 2f 32 30 31 30 30 31   .0) Geck o/201001
00b0  30 31 20 46 69 72 65 66  6f 78 2f 37 38 2e 30 0d   01 Firef ox/78.0·
00c0  0a 41 63 63 65 70 74 3a  20 74 65 78 74 2f 68 74   ·Accept:  text/ht
00d0  6d 6c 2c 61 70 70 6c 69  63 61 74 69 6f 6e 2f 78   ml,appli cation/x
00e0  68 74 6d 6c 2b 78 6d 6c  2c 61 70 70 6c 69 63 61   html+xml ,applica
00f0  74 69 6f 6e 2f 78 6d 6c  3b 71 3d 30 2e 39 2c 69   tion/xml ;q=0.9,i
0100  6d 61 67 65 2f 77 65 62  70 2c 2a 2f 2a 3b 71 3d   mage/web p,*/*;q=
0110  30 2e 38 0d 0a 41 63 63  65 70 74 2d 4c 61 6e 67   0.8··Acc ept-Lang
0120  75 61 67 65 3a 20 65 6e  2d 55 53 2c 65 6e 3b 71   uage: en -US,en;q
0130  3d 30 2e 35 0d 0a 41 63  63 65 70 74 2d 45 6e 63   =0.5··Ac cept-Enc
0140  6f 64 69 6e 67 3a 20 67  7a 69 70 2c 20 64 65 66   oding: g zip, def
0150  6c 61 74 65 0d 0a 43 6f  6e 6e 65 63 74 69 6f 6e   late··Co nnection
0160  3a 20 6b 65 65 70 2d 61  6c 69 76 65 0d 0a 55 70   : keep-a live··Up
0170  67 72 61 64 65 2d 49 6e  73 65 63 75 72 65 2d 52   grade-In secure-R
0180  65 71 75 65 73 74 73 3a  20 31 0d 0a 49 66 2d 4d   equests:  1··If-M
0190  6f 64 69 66 69 65 64 2d  53 69 6e 63 65 3a 20 54   odified- Since: T
```

The credentials used to access the page are visible in plaintext, which is obviously not ideal. This is the primary reason for which HTTPS is chosen over HTTP for most applications. If the server were configured to run over HTTPS, then this sensitive data would appear in its SSL encrypted form.

Part 2: Viewing RFC Documents

RFC or Request For Comments document many of the common protocols and standards designed and set by the IETF. (Internet Engineering Task Force) You can search for them here: https://www.rfc-editor.org/retrieve/

Figure 9: RFC Search Page



RFC Editor

Search RFCs

793

Advanced Search

The Series

- Document Retrieval
- Errata
- FAQ
- Format Change FAQ
- History
- About Us
- Other Information

For Authors

- Publication Process
- Publication Queue
- Style Guide
- xml2rfc FAQ

Sponsor

Internet Society

Document Retrieval

In addition to searching, you can browse and download RFCs as follows.

By Stream

- IETF: art (previously: app and rai) | gen | int | ops | rtg | sec | tsv | Non-WG
- IRTF
- IAB
- Independent Submissions
- Legacy

By Status

- Internet Standard | Draft Standard | Proposed Standard

  See RFC 6410 for more information regarding maturity levels.
- Best Current Practice
- Informational | Experimental | Historic
- Uncategorized (Early RFCs)

By Number

RFCs are available in files of 500 documents; please see this listing of TAR and ZIP files.

Via rsync

Here are details for using rsync to efficiently maintain a local copy of various subsets of the RFC Editor's repository in sync with the official copy.

Via RSS or Atom Feed

See recently published RFCs (RSS feed or Atom feed).

By Service / Technology

[This list is not comprehensive. It is experimental; it relies on search by title and keyword.]

Applications and Real-Time: EPP | FTP | HTTP | iCalendar | IDNA | IMAP | LDAP | MIME | OAuth | POP3 | URN | vCard | XMPP | RTSP | RTP | SDP | SIP | WebRTC | VoIP

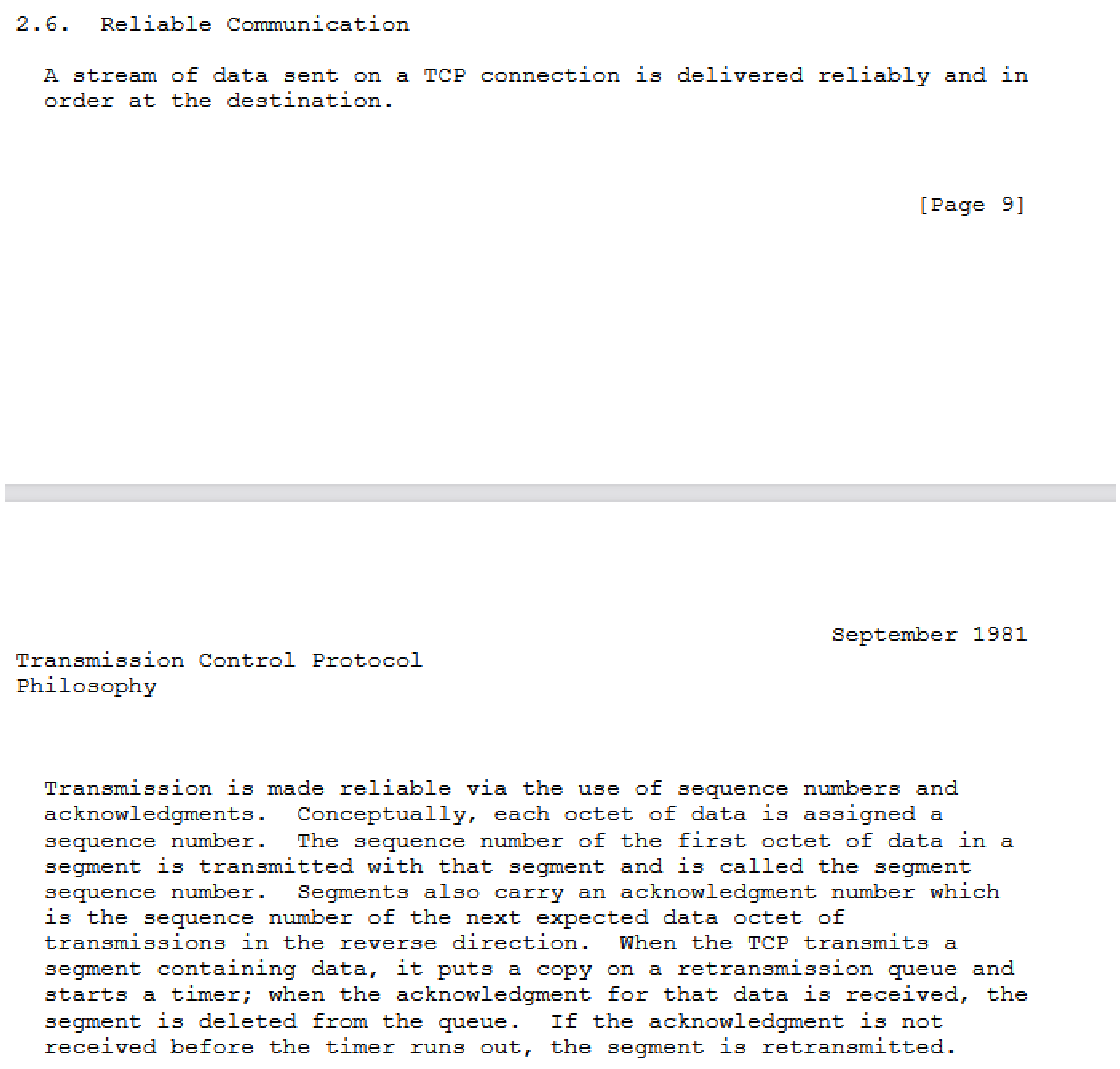
Internet: DHCPv4 | DHCPv6 | DNS | IPv4 | IPv6 | MIPv4 | MIPv6 | MPLS | NTP | PWE3

Operations and Management: CAPWAP | Diameter | NETCONF | RADIUS | SMI | SNMP | YANG

Routing: BGP | CIDR | IS-IS | LDP | OSPF | PIM | RSVP-TE | VRRP

We search for RFC 793, which documents the design philosophy of the widely used TCP protocol. Section 2 of this document delves into detail as to the reliability of TCP, and how it works. Some excerpts are shown below.

Figure 10: Section 2.6 - TCP Reliability

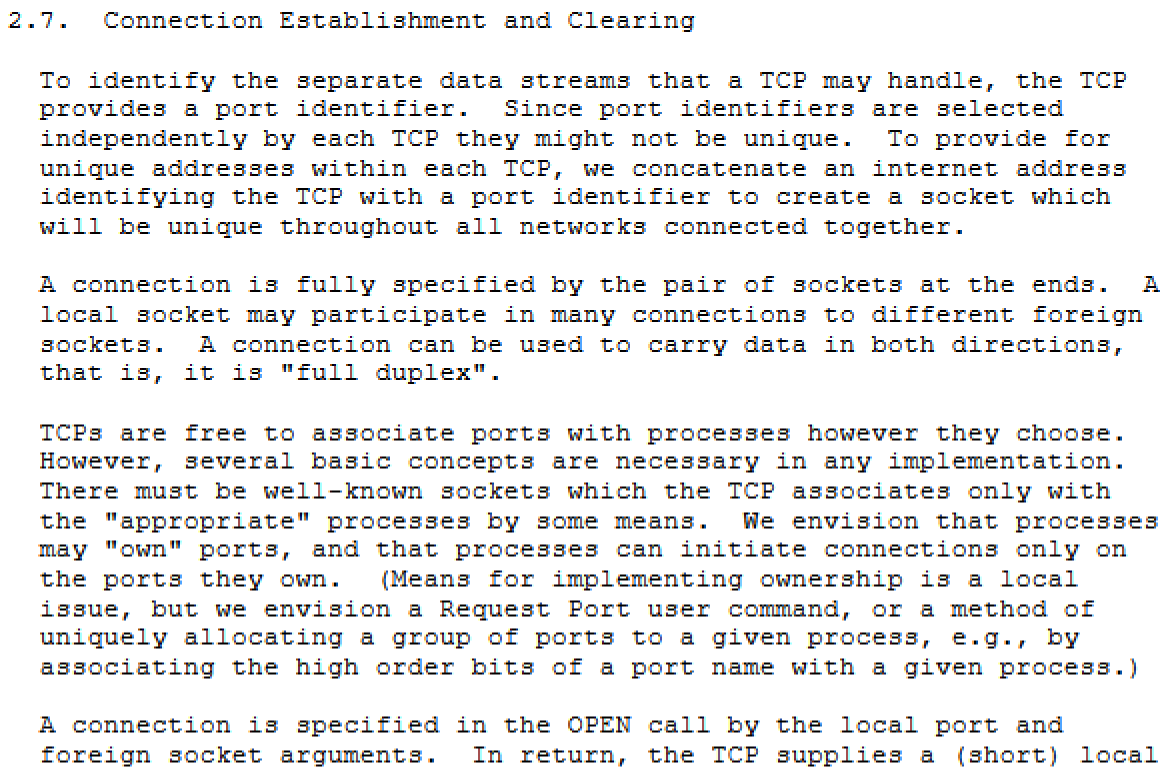


2.6. Reliable Communication

A stream of data sent on a TCP connection is delivered reliably and in order at the destination.

Transmission is made reliable via the use of sequence numbers and acknowledgments. Conceptually, each octet of data is assigned a sequence number. The sequence number of the first octet of data in a segment is transmitted with that segment and is called the segment sequence number. Segments also carry an acknowledgment number which is the sequence number of the next expected data octet of transmissions in the reverse direction. When the TCP transmits a segment containing data, it puts a copy on a retransmission queue and starts a timer; when the acknowledgment for that data is received, the segment is deleted from the queue. If the acknowledgment is not received before the timer runs out, the segment is retransmitted.

Figure 11: Section 2.7 - TCP Connection Establishment



2.7. Connection Establishment and Clearing

To identify the separate data streams that a TCP may handle, the TCP provides a port identifier. Since port identifiers are selected independently by each TCP they might not be unique. To provide for unique addresses within each TCP, we concatenate an internet address identifying the TCP with a port identifier to create a socket which will be unique throughout all networks connected together.

A connection is fully specified by the pair of sockets at the ends. A local socket may participate in many connections to different foreign sockets. A connection can be used to carry data in both directions, that is, it is "full duplex".

TCPs are free to associate ports with processes however they choose. However, several basic concepts are necessary in any implementation. There must be well-known sockets which the TCP associates only with the "appropriate" processes by some means. We envision that processes may "own" ports, and that processes can initiate connections only on the ports they own. (Means for implementing ownership is a local issue, but we envision a Request Port user command, or a method of uniquely allocating a group of ports to a given process, e.g., by associating the high order bits of a port name with a given process.)

A connection is specified in the OPEN call by the local port and foreign socket arguments. In return, the TCP supplies a (short) local connection name by which the user refers to the connection in subsequent calls. There are several things that must be remembered about a connection. To store this information we imagine that there is a data structure called a Transmission Control Block (TCB). One implementation strategy would have the local connection name be a pointer to the TCB for this connection. The OPEN call also specifies whether the connection establishment is to be actively pursued, or to be passively waited for.

A passive OPEN request means that the process wants to accept incoming connection requests rather than attempting to initiate a connection. Often the process requesting a passive OPEN will accept a connection request from any caller. In this case a foreign socket of all zeros is used to denote an unspecified socket. Unspecified foreign sockets are allowed only on passive OPENs.

A service process that wished to provide services for unknown other processes would issue a passive OPEN request with an unspecified foreign socket. Then a connection could be made with any process that requested a connection to this local socket. It would help if this local socket were known to be associated with this service.

Well-known sockets are a convenient mechanism for a priori associating a socket address with a standard service. For instance, the "Telnet-Server" process is permanently assigned to a particular socket, and other sockets are reserved for File Transfer, Remote Job Entry, Text Generator, Echoer, and Sink processes (the last three being for test purposes). A socket address might be reserved for access to a "Look-Up" service which would return the specific socket at which a newly created service would be provided. The concept of a well-known socket is part of the TCP specification, but the assignment of sockets to services is outside this specification. (See [4].)

Processes can issue passive OPENs and wait for matching active OPENs from other processes and be informed by the TCP when connections have been established. Two processes which issue active OPENs to each other at the same time will be correctly connected. This flexibility is critical for the support of distributed computing in which components act asynchronously with respect to each other.

There are two principal cases for matching the sockets in the local passive OPENs and an foreign active OPENs. In the first case, the local passive OPENs has fully specified the foreign socket. In this case, the match must be exact. In the second case, the local passive OPENs has left the foreign socket unspecified. In this case, any foreign socket is acceptable as long as the local sockets match. Other possibilities include partially restricted matches.

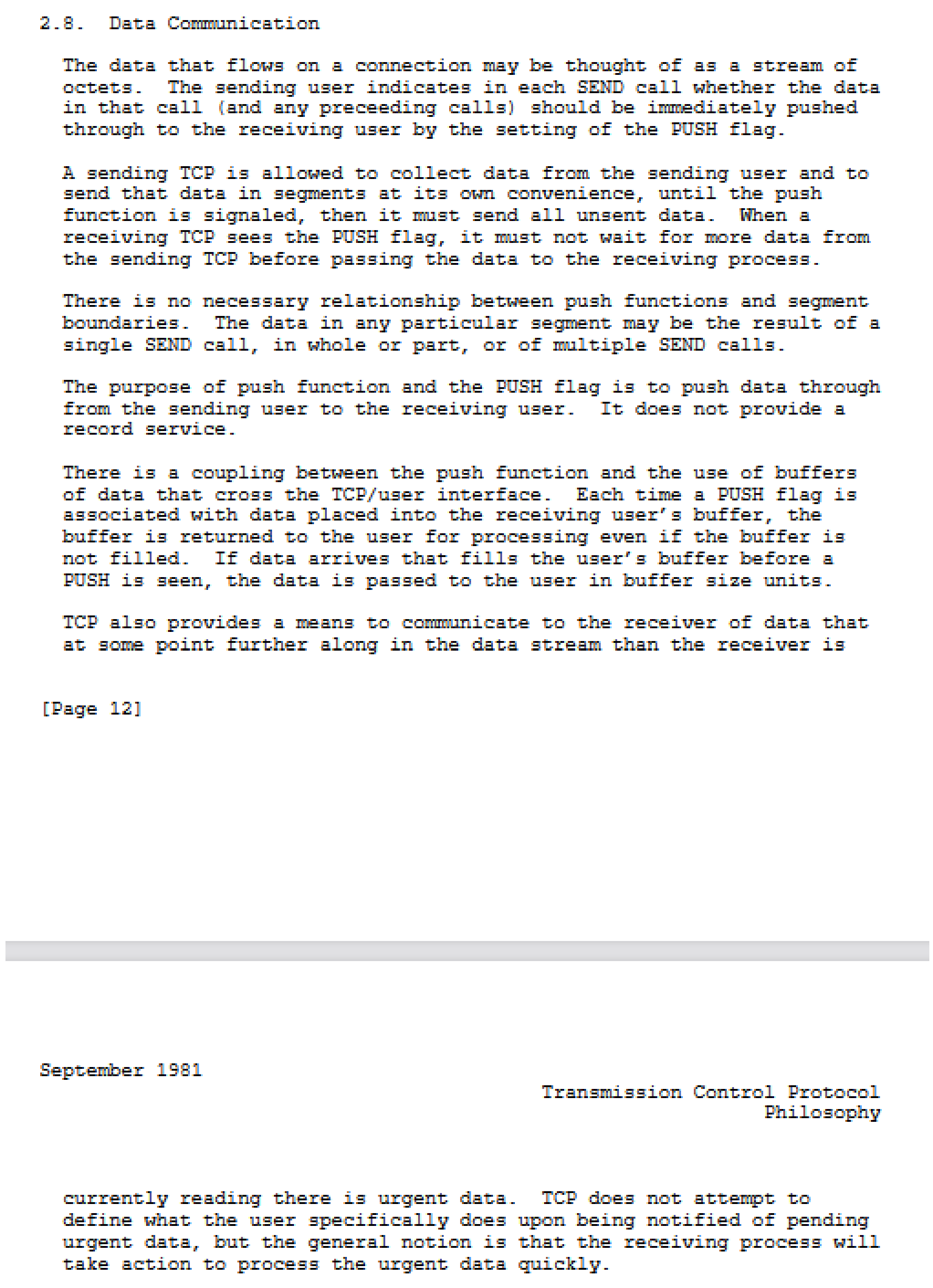
If there are several pending passive OPENs (recorded in TCBs) with the same local socket, an foreign active OPEN will be matched to a TCB with the specific foreign socket in the foreign active OPEN, if such a TCB exists, before selecting a TCB with an unspecified foreign socket.

The procedures to establish connections utilize the synchronize (SYN) control flag and involves an exchange of three messages. This exchange has been termed a three-way hand shake [3].

A connection is initiated by the rendezvous of an arriving segment containing a SYN and a waiting TCB entry each created by a user OPEN command. The matching of local and foreign sockets determines when a connection has been initiated. The connection becomes "established" when sequence numbers have been synchronized in both directions.

The clearing of a connection also involves the exchange of segments, in this case carrying the FIN control flag.

Figure 12: Section 2.8 - TCP Communication



2.8. Data Communication

The data that flows on a connection may be thought of as a stream of octets. The sending user indicates in each SEND call whether the data in that call (and any preceeding calls) should be immediately pushed through to the receiving user by the setting of the PUSH flag.

A sending TCP is allowed to collect data from the sending user and to send that data in segments at its own convenience, until the push function is signaled, then it must send all unsent data. When a receiving TCP sees the PUSH flag, it must not wait for more data from the sending TCP before passing the data to the receiving process.

There is no necessary relationship between push functions and segment boundaries. The data in any particular segment may be the result of a single SEND call, in whole or part, or of multiple SEND calls.

The purpose of push function and the PUSH flag is to push data through from the sending user to the receiving user. It does not provide a record service.

There is a coupling between the push function and the use of buffers of data that cross the TCP/user interface. Each time a PUSH flag is associated with data placed into the receiving user's buffer, the buffer is returned to the user for processing even if the buffer is not filled. If data arrives that fills the user's buffer before a PUSH is seen, the data is passed to the user in buffer size units.

TCP also provides a means to communicate to the receiver of data that at some point further along in the data stream than the receiver is

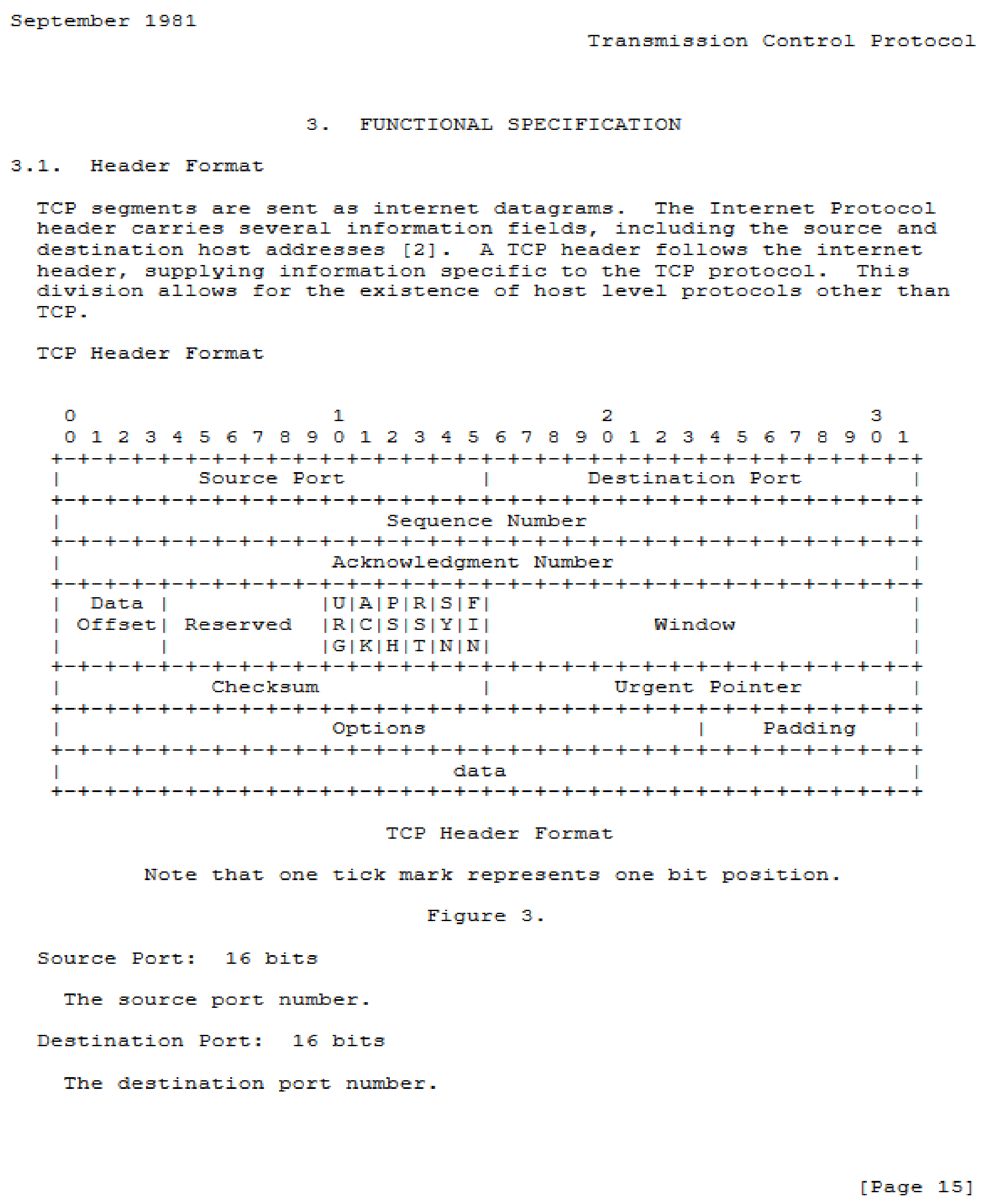
[Page 12]

September 1981
Transmission Control Protocol
Philosophy

currently reading there is urgent data. TCP does not attempt to define what the user specifically does upon being notified of pending urgent data, but the general notion is that the receiving process will take action to process the urgent data quickly.

Section 3 further elaborates on the functional aspects of TCP's design. The first page of this section provides a valuable overview of the layout of a TCP header, which is shown below.

Figure 13: TCP Header Format



```
September 1981
                                           Transmission Control Protocol



                     3.  FUNCTIONAL SPECIFICATION

3.1.  Header Format

  TCP segments are sent as internet datagrams.  The Internet Protocol
  header carries several information fields, including the source and
  destination host addresses [2].  A TCP header follows the internet
  header, supplying information specific to the TCP protocol.  This
  division allows for the existence of host level protocols other than
  TCP.

  TCP Header Format


    0                   1                   2                   3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |          Source Port          |       Destination Port        |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                        Sequence Number                        |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                    Acknowledgment Number                      |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |  Data |           |U|A|P|R|S|F|                               |
   | Offset| Reserved  |R|C|S|S|Y|I|            Window             |
   |       |           |G|K|H|T|N|N|                               |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |           Checksum            |         Urgent Pointer        |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                    Options                    |    Padding    |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                             data                              |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

                            TCP Header Format

          Note that one tick mark represents one bit position.

                               Figure 3.

  Source Port:  16 bits

    The source port number.

  Destination Port:  16 bits

    The destination port number.



                                                               [Page 15]
```