

サイバーセキュリティ PBL I

高野 祐輝

2019 年 2 月 22 日

1 目的と評価方法

- 最優 ファイアウォール技術とペネトレーションテストを組み合わせ、検疫ネットワークの設計と構築を行うことができる
- 優 ファイアウォール技術で DeMilitarized Zone のあるネットワーク設計と構築を行うことができる
- 良 ファイアウォール技術で適切なネットワークアクセスコントロールができる
- 可 各種サイバー攻撃手法と防御手法について論じることができる

2 セキュリティ哲学

2.1 サイバーセキュリティとは何か

2.2 サイバーキルチェーン

[1]

2.3 セキュリティポリシとユーザビリティ

[2]

3 TCP/IP の基礎

3.1 OSI 参照モデル

インターネットで利用されるプロトコルは、The Internet Engineering Task Force (IETF) という標準化団体により策定され、その標準は Request for Comments (RFC) という名のオープンな仕様として発行されている。例えば、我々が利用しているインターネットプロトコルであるインターネットプロトコルバージョン 4 は、1981 年に 791 番目の RFC として策定された [3]。

IETF 以外の通信に関する標準化団体としては International Telecommunication Union Telecommunication Standardization Sector (ITU-T) や、International Organization for Standardization (ISO) が存在する。実は、1977 年から 1982 年かけて、ITU-T や ISO がコンピュータネットワークの標準通信プロトコルとして、Open Systems Interconnection (OSI) の策定を行っていた。その当時は標準的な通信プロトコルは存

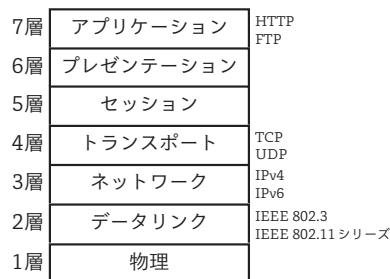


図 1 OSI 参照モデル

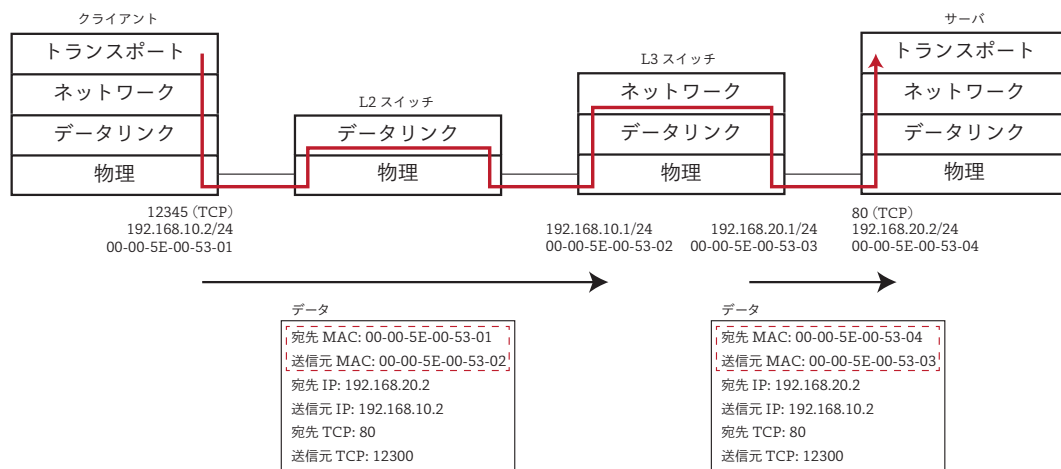


図 2 各層でのデータ転送

在せず、ベンダーごとに様々なプロトコルが利用されていたため、通信プロトコルの統一化が求められていたのである。しかしながら、最終的に OSI は主流とはならず、IETF によって策定されたインターネットプロトコルが広く利用されるようになっていった。

OSI 自体は残らなかったが、OSI 策定の際に考案された OSI 参照モデルと呼ばれるネットワークの抽象化手法は、今日でも広く受け入れられている。図 1 は、OSI 参照モデルによるネットワークの抽象化モデルを表している。OSI 参照モデルでは、ネットワークの機能を階層構造にもとづいて抽象化しており、この抽象化をレイヤリングなどと呼ぶ。OSI 参照モデルでは、下から順に 1 層に物理層、2 層にデータリンク層、3 層にネットワーク層、4 層にトランスポート層、5 層にセッション層、6 層にプレゼンテーション層、7 層にアプリケーション層が位置する。ちなみに、各層のことをレイヤ 1、レイヤ 2 といったり、更に略して L1、L2 などということもある。

図 2 は各層でデータ転送が行われている様子を示している。^{*1} データリンク、ネットワーク、トランスポート層のプロトコルにはそれぞれアドレスがあり、各層は、そのアドレスに基づいて転送を行う。データリンク層プロトコルの一つである IEEE 802 では、アドレスは 42 ビットで表され、文字で表現すると 00-00-5E-00-53-02 といった表記になる。図 2 中で宛先 MAC と示される値は、IEEE 802 の宛先 MAC アドレスを示している。なお、MAC は Media Access Control の略である。データリンク層は、ローカルなネッ

^{*1} この図の意味することは現時点では理解できないかもしれないが、この図の意味することを説明するのが本節の目標であるため、現段階で理解できなくても問題ない。

トワークでの通信を行うために用いられる。そのため、MAC アドレスはそのローカルな環境では一意に識別できる必要がある。データリンク層の詳細については 3.3 節で解説する。

ネットワーク層プロトコルの一つである IP のアドレスは、192.168.10.2/24 という 32 ビットの数値で表され、/24 はネットワークのサブネット長を示している。図 2 では、192.168.10.0/24 と 192.168.20.0/24 というサブネットが示されている。IP は、全世界で通信を行うために用いられるプロトコルであり、基本的には IP アドレスは世界で一意に識別できるように割り当てるのが設計理念となっている（現実的にはそうはなっていないが）。なお、前述のアドレスは IPv4 アドレスであるが、IPv6 の場合は 128 ビットのアドレス空間を持つ。ネットワーク層の詳細については 3.4 節で解説する。

トランスポート層プロトコルの TCP と UDP のアドレスは 16 ビットで示され、一般的にポート番号と呼ばれ、TCP や UDP はポート番号をもとにアプリケーションプロセスの識別を行う。よく利用されるポート番号は、インターネット上で利用される識別情報の管理割当を行っている Internet Assigned Number Authority (IANA) が定義しており [4]、一般的にこのようなポート番号を Well Known ポート番号と呼ぶ。例えば、TCP の 80 番ポートは HTTP で利用され、普段我々が Web を閲覧する際は、Web ブラウザが Web サーバの TCP80 番ポートへ接続する。

図 2 では、クライアントからサーバの TCP80 番ポートへむけて通信を行っている様子を示している。一般的に、インターネット上の通信ではデータ中に含まれる各層のアドレスをもとに、L2 または L3 スイッチが転送を行う。L2 スイッチのことをスイッチングハブといたり、L3 スイッチのことをルータということもあるが、本書では L2 スイッチ、L3 スイッチと呼ぶことにする。この図が示すように、L2 スイッチ、L3 スイッチによってデータが転送されても、データ中の IP アドレスとポート番号は変わらないが、MAC アドレスは L3 スイッチでの転送時に更新される。これは、MAC アドレスはローカルなネットワーク内でのみ通用するアドレスであり、L3 スイッチはローカルなネットワーク同士をつなぎ合わせる役割を持っているためである。以降の節では、データリンク、ネットワーク、トランスポートの動きについて詳しく説明する。

重要ポイント

- インターネット関連のプロトコルは、IETF が発行する RFC によって標準化されている
- コンピュータネットワークはレイヤで考えることができる
- Ethernet のアドレスは 48 ビットの MAC アドレス、IPv4 のアドレスは 32 ビットの IPv4 アドレス、IPv6 のアドレスは 128 ビットの IPv6 アドレス、TCP と UDP のアドレスは 16 ビットのポート番号

3.2 おもちゃのネットワークスタック

これより本章では、おもちゃのネットワークスタックを用いて、ネットワークスタックの設計と実装を解説する。おもちゃと言っても、実際に IP ルータや Ethernet ブリッジとして動作するれっきとしたネットワークスタックである。図 3 はおもちゃのネットワークスタックのデータフロー図を示している。この図の下部には、入力と出力用の Ethernet Input/Output Queue というキューがあり、ここで物理的な入出力が行われる。実際に、ネットワークインターフェースカードには入出力用のキューが用意されており、デバイスドライバはこれらキューに対して読み書きすることでデータの送受信を行う。

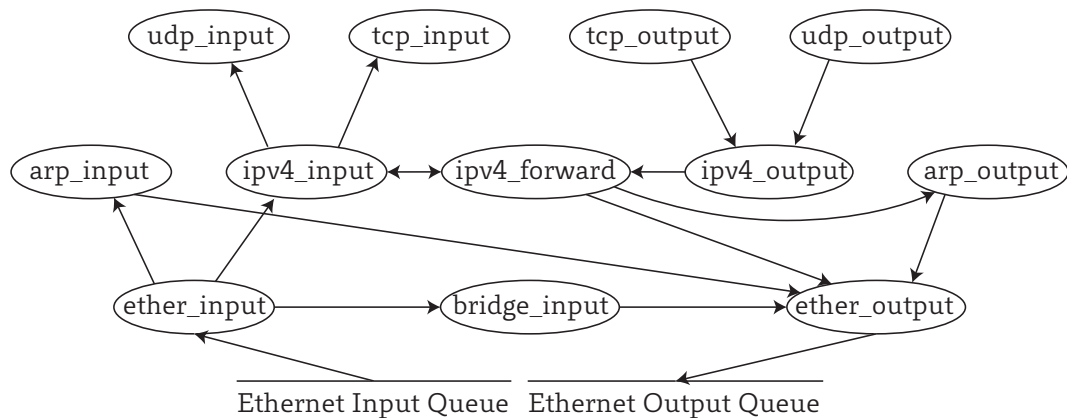


図3 おもちゃのネットワークスタックのデータフロー図

3.3 データリンク層

ソースコード 1 ether_input 関数 (ether.c)

```

1  /*
2  * Ethernet フレーム入力関数
3  * 引数:
4  *   ifp: 入力インターフェース
5  *   eh: 入力フレーム
6  *   len: 入力フレーム長
7  */
8  void ether_input(struct my_ifnet *ifp, struct ether_header *eh, int len) {
9      printf("ether_input:\n");
10     printf("UUUUIF#: %d\n", ifp->idx);
11     printf("UUUUsrcMAC: %02X-%02X-%02X-%02X-%02X-%02X\n", eh->ether_shost[0],
12            eh->ether_shost[1], eh->ether_shost[2], eh->ether_shost[3],
13            eh->ether_shost[4], eh->ether_shost[5]);
14     printf("UUUUdstMAC: %02X-%02X-%02X-%02X-%02X-%02X\n", eh->ether_dhost[0],
15            eh->ether_dhost[1], eh->ether_dhost[2], eh->ether_dhost[3],
16            eh->ether_dhost[4], eh->ether_dhost[5]);
17     printf("\n");
18
19     if (IS_BROADCAST(eh->ether_dhost)) {
20         // ブロードキャストアドレスの場合、ブリッジ処理へ
21         if (IS_L2BRIDGE)
22             bridge_input(ifp, eh, len);
23     } else if (memcmp(ifp->ifaddr, eh->ether_dhost, ETHER_ADDR_LEN) != 0) {
24         // 宛先MACアドレスが自インターフェース宛でないならブリッジ処理を行い終了
25         if (IS_L2BRIDGE)
26             bridge_input(ifp, eh, len);
27         return;
28     }
29
30     switch (ntohs(eh->ether_type)) {
31     case ETHERTYPE_IP: // IPv4 入力
32         ipv4_input((struct ip *)((uint8_t *)eh + ETHER_HDR_LEN));
33         break;
34     case ETHERTYPE_IPV6: // IPv6 入力

```

```

35     ipv6_input((struct ip6_hdr *)((uint8_t *)eh + ETHER_HDR_LEN));
36     break;
37 case ETHERTYPE_ARP: // ARP 入力
38     arp_input(ifp, (struct arphdr *)((uint8_t *)eh + ETHER_HDR_LEN));
39     break;
40 default:
41     printf("eh->ether_type is neither IPv6 nor IPv6\n");
42     return;
43 }
44
45 return;
46 }

```

ソースコード 2 Ethernet プロトコルヘッダ定義 (/usr/include/netinet/if_ether.h)

```

1  #define ETHER_ADDR_LEN 6      /* Ethernet address length */
2
3  /*
4   * Ethernet address - 6 octets
5   */
6  struct ether_addr {
7      u_int8_t ether_addr_octet[ETHER_ADDR_LEN];
8  };
9
10 /*
11  * The length of the combined header.
12  */
13 struct ether_header {
14     u_int8_t ether_dhost[ETHER_ADDR_LEN];
15     u_int8_t ether_shost[ETHER_ADDR_LEN];
16     u_int16_t ether_type;
17 };
18

```

3.4 ネットワーク層

ソースコード 3 IPv4 ヘッダ定義 (/usr/include/netinet/ip.h)

```

1  /*
2   * Structure of an internet header, naked of options.
3   */
4  struct ip {
5      #if _BYTE_ORDER == _LITTLE_ENDIAN
6          u_int      ip_hl:4,          /* header length */
7                      ip_v:4;          /* version */
8      #endif
9      #if _BYTE_ORDER == _BIG_ENDIAN
10         u_int      ip_v:4,          /* version */
11                     ip_hl:4;          /* header length */
12      #endif
13         u_int8_t ip_tos;              /* type of service */
14         u_int16_t ip_len;              /* total length */
15         u_int16_t ip_id;               /* identification */
16         u_int16_t ip_off;              /* fragment offset field */
17         #define IP_RF 0x8000          /* reserved fragment flag */

```

```

18 #define IP_DF 0x4000                /* dont fragment flag */
19 #define IP_MF 0x2000                /* more fragments flag */
20 #define IP_OFFMASK 0x1fff           /* mask for fragmenting bits */
21     u_int8_t  ip_ttl;                /* time to live */
22     u_int8_t  ip_p;                  /* protocol */
23     u_int16_t ip_sum;                 /* checksum */
24     struct    in_addr ip_src, ip_dst; /* source and dest address */
25 };

```

ソースコード 4 IPv6 アドレス構造体 (/usr/include/netinet6/in6.h)

```

1 /*
2  * IPv6 address
3  */
4 struct in6_addr {
5     union {
6         u_int8_t  __u6_addr8[16];
7         u_int16_t  __u6_addr16[8];
8         u_int32_t  __u6_addr32[4];
9     } __u6_addr; /* 128-bit IP6 address */
10 };

```

ソースコード 5 IPv6 ヘッダ定義 (/usr/include/netinet/ip6.h)

```

1 /*
2  * Definition for internet protocol version 6.
3  * RFC 2460
4  */
5
6 struct ip6_hdr {
7     union {
8         struct ip6_hdrctl {
9             u_int32_t ip6_un1_flow; /* 20 bits of flow-ID */
10            u_int16_t ip6_un1_plen; /* payload length */
11            u_int8_t  ip6_un1_nxt; /* next header */
12            u_int8_t  ip6_un1_hlim; /* hop limit */
13        } ip6_un1;
14        u_int8_t ip6_un2_vfc; /* 4 bits version, top 4 bits class */
15    } ip6_ctlun;
16    struct in6_addr ip6_src; /* source address */
17    struct in6_addr ip6_dst; /* destination address */
18 } __packed;

```

3.5 トランスポート層

ソースコード 6 TCP ヘッダ定義 (/usr/include/netinet/tcp.h)

```

1 typedef u_int32_t tcp_seq;
2
3 /*
4  * TCP header.
5  * Per RFC 793, September, 1981.
6  */
7 struct tcphdr {
8     u_int16_t th_sport; /* source port */

```

```

9         u_int16_t th_dport;           /* destination port */
10        tcp_seq th_seq;                /* sequence number */
11        tcp_seq th_ack;                /* acknowledgement number */
12 #if _BYTE_ORDER == _LITTLE_ENDIAN
13        u_int32_t th_x2:4,             /* (unused) */
14                th_off:4;              /* data offset */
15 #endif
16 #if _BYTE_ORDER == _BIG_ENDIAN
17        u_int32_t th_off:4,             /* data offset */
18                th_x2:4;               /* (unused) */
19 #endif
20        u_int8_t th_flags;
21 #define TH_FIN    0x01
22 #define TH_SYN    0x02
23 #define TH_RST    0x04
24 #define TH_PUSH   0x08
25 #define TH_ACK    0x10
26 #define TH_URG    0x20
27 #define TH_ECE    0x40
28 #define TH_CWR    0x80
29        u_int16_t th_win;               /* window */
30        u_int16_t th_sum;               /* checksum */
31        u_int16_t th_urp;               /* urgent pointer */
32 };

```

ソースコード 7 UDP ヘッダ定義 (/usr/include/netinet/udp.h)

```

1  /*
2   * Udp protocol header.
3   * Per RFC 768, September, 1981.
4   */
5  struct udphdr {
6      u_int16_t uh_sport;                /* source port */
7      u_int16_t uh_dport;                /* destination port */
8      u_int16_t uh_ulen;                 /* udp length */
9      u_int16_t uh_sum;                  /* udp checksum */
10 };

```

- 3.6 トランスポートより上の層
- 4 PF (Packet Filter) の基礎
- 5 パケットフィルタリング
- 6 攻撃手法と対策
- 7 DMZ (DeMilitarized Zone) の構築
- 8 検疫ネットワークの構築
- 9 ロードバランス
- 10 ロギング

付録 A Vagrant による実験環境の構築

付録 B PF の構文

参考文献

- [1] Eric M Hutchins, Michael J Cloppert, and Rohan M Amin. Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains. *Leading Issues in Information Warfare & Security Research*, Vol. 1, p. 80, 2011.
- [2] B. Fraser. Site security handbook, September 1997. RFC2196.
- [3] J. Postel. Internet protocol, September 1981. RFC0791.
- [4] Internet Assigned Number Authority (IANA). Service Name and Transport Protocol Port Number Registry. <https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml>.