

拡張可能な パケット解析ライブラリ開発

サイボウズ・ラボニュース第5期 成果報告会

@slankdev

自己紹介

- Twitter: @slankdev
- GitHub: slankdev
- サイボウズ・ラボユース第5期生
(第6期もお世話になります)
- セキュリティ・キャンプ 2015 修了生
- パケットとかネットワークとか好き

サイボウズ・ラボユース 第5期

- 内容

- 開発テーマ: C/C++によるソフトウェア開発
- メンター: 光成 滋生さん

- LibPGENを開発

- 読み方: りぶぴーじえん
- パケット解析のライブラリ
- <https://github.com/slankdev/libpgen>

注意

- ここでの”解析”は少し広義な意味として使います。

Agenda

- LibPGEN
 - LibPGENとは、その設計は
 - 簡単に使う
 - 拡張性
- 少しがっつり使う
 - 新たなプロトコルに関するパケット解析
- 今後の展開と総まとめ

Agenda

- LibPGEN
 - LibPGENとは、その設計は
 - 簡単に使う
 - 拡張性
- 少しがっつり使う
 - 新たなプロトコルに関するパケット解析
- 今後の展開と総まとめ

LibPGENとは

- C++11で開発しているパケット解析のライブラリ
- Linux, BSDで動作
- パケット解析についてのあらゆる作業をサポート (後述)
- 拡張がしやすい設計
 - 新たなプロトコルに対しての拡張が容易 (後述)

LibPGENの設計

- 大きく分けて三つのコンポーネントに分かれています

- Module
 - Core, IOを使って書かれたモジュール群
 - めんどくさい作業をまとめてやってくれるクラスや関数群
 - 説明は省略

- Core
 - パケットやアドレスのバイナリを解析したり、変更したりする部分
 - pgen::ethernet, pgen::ip, ...etc

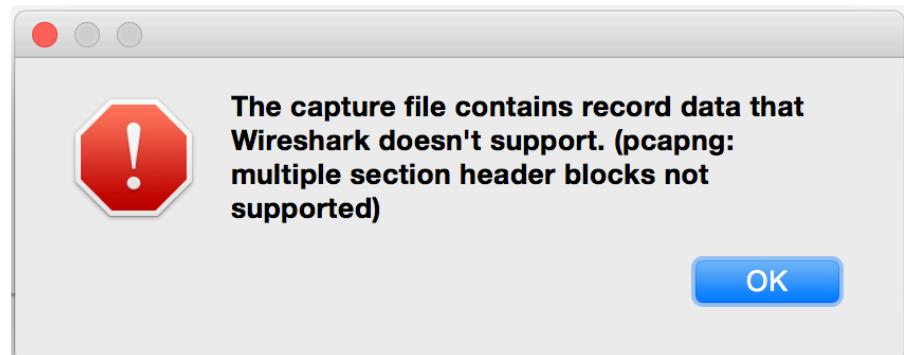
- IO
 - データをネットワークインターフェースやpcap, pcapngファイルに入出力するクラス群
 - pgen::net_stream, pgen::pcap_stream ...etc

LibPGEN: IO

- 幾つかのインターフェースを触れる
 - ネットワークインターフェース
 - pcapファイル
 - pcapngファイル



も完全対応ではない (最新版は知りません)



- ストリームクラスとして実装
- OSごとの実装の違いを吸収

OSごとの実装の違いを吸収

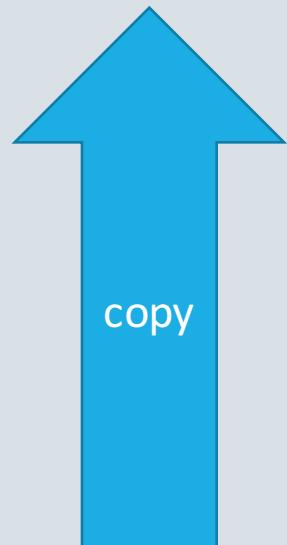
- read(2)の挙動がBSDとLinuxで違う
 - Linux: 1回のreadでパケット1つ読む
 - BSD : 1回のreadでバッファに溜まっている全てのパケットを読む
- この説明は割愛しますが、
面白いので興味があれば懇親会で
- そもそもファイルディスクリプタの開き方が全然ちがう
 - Linux: PF_PACKETアドレスファミリでsocket(2)
 - BSD : BPF(/dev/bpfN)を直接open(2)
- まだ理解仕切れていない。。。

BSDとLinuxの違いの
説明はないので雰囲気で
感じ取ってください

PF_PACKETでのreadの挙動

User Land

Packet



Kernel Land

Packet

Packet

Packet

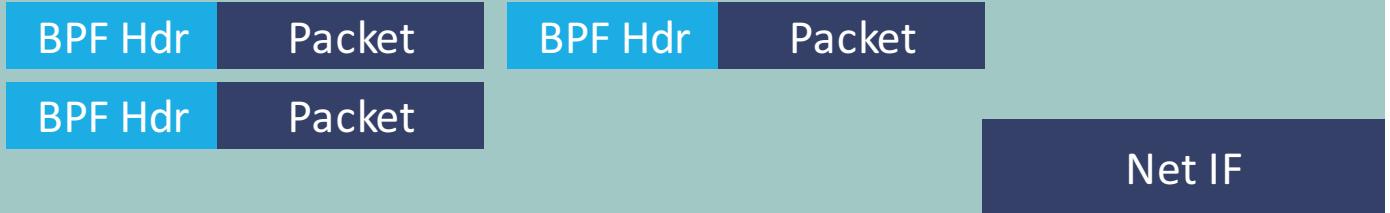
Net IF

BPFでのreadの挙動

User Land



Kernel Land



LibPGENでの実装 (BSD用)

Special Thanks, KOZOS Sakai

User Land

Packet

copy

Sub User Land

BPF Hdr Packet

BPF Hdr Packet

BPF Hdr Packet

BPF Hdr Packet

Kernel Land

BPF Hdr Packet

BPF Hdr Packet

BPF Hdr Packet

BPF Hdr Packet

Net IF

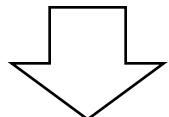
感じ取る時間終わりです

LibPGEN: Core

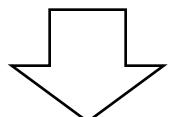
- 現在このライブラリで一番大事なところ
- パケット解析などをサポートするクラス群
- パケットのバイナリ生成、既存のバイナリを解析、既存のバイナリを編集などができる

LibPGEN: Core

```
ffff ffff ffff 703e aceb 27a5 0806 0001  
0800 0604 0001 703e aceb 27a5 0000 0000  
0000 0000 0000 0ad2 7c7e 5b2a 0100 2d1a  
ac19 1bff ffff 0000 0000 0000
```



Packet Class



```
aaaa aaaa aaaa 703e aceb 27a5 0806 0001  
0800 0604 0002 703e aceb 27a5 0000 0000  
0000 0000 0000 0ad2 7c7e 5b2a 0100 2d1a  
ac19 1bff ffff 0000 0000 0000 4920 6c6f  
7665 2070 6163 6b65 742e
```

○ 使用用途

- ゼロからパケットのバイナリを組み立て
- 既存のバイナリを変更
- 長さを変えたり途中に追加など柔軟に

○ パケットクラスとして実装

Agenda

- LibPGEN

- LibPGENとは、その設計は
- 簡単に使う
- 拡張性

- 少しがっつり使う

- 新たなプロトコルに関するパケット解析

- 今後の展開と総まとめ

簡単に使う

- ARPing の実装

- ARPリクエストでpingの動作を実装
- 使うクラス、関数など
 - pgen::arpクラス, ARPパケットを実装したクラス
 - pgen::module::detect関数, パケットの種類をフィルタリング
 - pgen::net_streamクラス, NETIFへの送受信
- Usage: ./arping interface targetIP
- LibPGENを使うことが目的

ARPing プログラム

```
#include <pgen2.h>
using pgen::arp;
using pgen::net_stream;
using pgen::module::detect;
using pgen::open_mode;
using pgen::packet_type;

int main(int argc, char** argv) {
    if (argc < 3) {
        fprintf(stderr, "Usage: %s interface targetIP\n", argv[0]);
        return -1;
    }

    try {
        arp send_pack;
        send_pack.ETH.src.setbydev(argv[1]);
        send_pack.ETH.dst.setbroadcast();
        send_pack.ARPs.operation = arp::operation::request;
        send_pack.ARPs.hwsrc = send_pack.ETH.src;
        send_pack.ARPs.psrc.setbydev(argv[1]);
        send_pack.ARPs.hwdst = send_pack.ETH.dst;
        send_pack.ARPs.pdst = argv[2];

        net_stream net(argv[1], open_mode::netif);

        printf("ARPING %s from %s %s\n", argv[2],
               send_pack.ETH.src.str().c_str(), argv[1]);
        for (int count=0; count<5; ) {
            send_pack.compile();
            net << send_pack;

            uint8_t recvbuf[10000];
            size_t recflen = net.recv(recvbuf, sizeof recvbuf);

            if (detect(recvbuf, recflen) == pgen::packet_type::arp) {
                arp recv_pack(recvbuf, recflen);
                if (recv_pack.ARPs.operation == arp::operation::reply) {
                    recv_pack.ARPs.summary();
                    sleep(1);
                    count++;
                }
            }
        }
    } catch (std::exception& e) {
        printf("%s \n", e.what());
    }
}
```

- ソースコードはこんな感じ

ARPing プログラム

```
#include <pgen2.h>
using pgen::arp;
using pgen::net_stream;
using pgen::module::detect;
using pgen::open_mode;
using pgen::packet_type;

int main(int argc, char** argv) {
    if (argc < 3) {
        fprintf(stderr, "Usage: %s interface targetIP\n", argv[0]);
        return -1;
    }

    try {
        arp send_pack;
        send_pack.ETH.src.setbydev(argv[1]);
        send_pack.ETH.dst.setbroadcast();
        send_pack.ARPs.operation = arp::operation::request;
        send_pack.ARPs.hwsrc = send_pack.ETH.src;
        send_pack.ARPs.psrc.setbydev(argv[1]);
        send_pack.ARPs.hwdst = send_pack.ETH.dst;
        send_pack.ARPs.pdst = argv[2];

        net_stream net(argv[1], open_mode::netif);

        printf("ARPING %s from %s %s\n", argv[2],
               send_pack.ETH.src.str().c_str(), argv[1]);
        for (int count=0; count<5; ) {
            send_pack.compile();
            net << send_pack;

            uint8_t recvbuf[10000];
            size_t recvlen = net.recv(recvbuf, sizeof recvbuf);

            if (detect(recvbuf, recvlen) == pgen::packet_type::arp) {
                arp recv_pack(recvbuf, recvlen);
                if (recv_pack.ARPs.operation == arp::operation::reply) {
                    recv_pack.ARPs.summary();
                    sleep(1);
                    count++;
                }
            }
        }
    } catch (std::exception& e) {
        printf("%s \n", e.what());
    }
}
```

○ソースコードはこんな感じ

○ちゃんと分割して説明します

ARPing プログラム

```
#include <pgen2.h>
using pgen::arp;
using pgen::net_stream;
using pgen::module::detect;
using pgen::open_mode;
using pgen::packet_type;

int main(int argc, char** argv) {
    if (argc < 3) {
        fprintf(stderr, "Usage: %s interface tergetIP\n", argv[0]);
        return -1;
}
```

ここはどうでもいい

ARPing プログラム

```
try {
    arp_send_pack;
    send_pack.ETH.src.setbydev(argv[1]);
    send_pack.ETH.dst.setbcast();
    send_pack.ARPs.operation = arp::operation::request;
    send_pack.ARPs.hwsrc = send_pack.ETH.src;
    send_pack.ARPs.psrc.setbydev(argv[1]);
    send_pack.ARPs.hwdst = send_pack.ETH.dst;
    send_pack.ARPs.pdst = argv[2];
    send_pack.compile();

    net_stream net(argv[1], open_mode::netif);

    printf("ARPING %s from %s %s\n", argv[2],
           send_pack.ETH.src.str().c_str(), argv[1]);
```

ここが重要!

ARPing プログラム

```
for (int count=0; count<5; ) {  
    net << send_pack;  
  
    uint8_t recvbuf[10000];  
    size_t recvlen = net.recv(recvbuf, sizeof recvbuf);  
  
    if (detect(recvbuf, recvlen) == pgen::packet_type::arp) {  
        arp recv_pack(recvbuf, recvlen);  
        if (recv_pack.ARP.operation == arp::operation::reply) {  
            recv_pack.ARP.summary();  
            sleep(1);  
            count++;  
        }  
    }  
}
```

少し重要

ARPing プログラム

どうでry)

```
    } catch (std::exception& e) {
        printf("%s \n", e.what());
    }
}
```

ARPing プログラム

- 使ってみるとこんな感じ

```
# ./arping en0 192.168.179.1
ARPING 192.168.179.1 from 80:e6:50::*:* en0
ARP [192.168.179.1 is at a2:12:42::*::*]
```

ARPing プログラム

- 使ってみるとこんな感じ

```
# ./arping en0 192.168.179.1
ARPING 192.168.179.1 from 80:e6:50::*:* en0
ARP [192.168.179.1 is at a2:12:42::*::*]
```

- プロトコルの知識があれば、ネットワークプログラミングの多少複雑なことを知らなくていい

ARPing プログラム

- 使ってみるとこんな感じ

```
# ./arping en0 192.168.179.1
ARPING 192.168.179.1 from 80:e6:50::*:* en0
ARP [192.168.179.1 is at a2:12:42::*::*]
```

- プロトコルの知識があれば、ネットワークプログラミングの多少複雑なことを知らなくていい
- これだけでは他のライブラリとかモジュールでも簡単

Agenda

- LibPGEN

- LibPGENとは、その設計は
- 簡単に使う
- 拡張性

- 少しがっつり使う

- 新たなプロトコルに関するパケット解析

- 今後の展開と総まとめ

LibPGEN: 拡張性

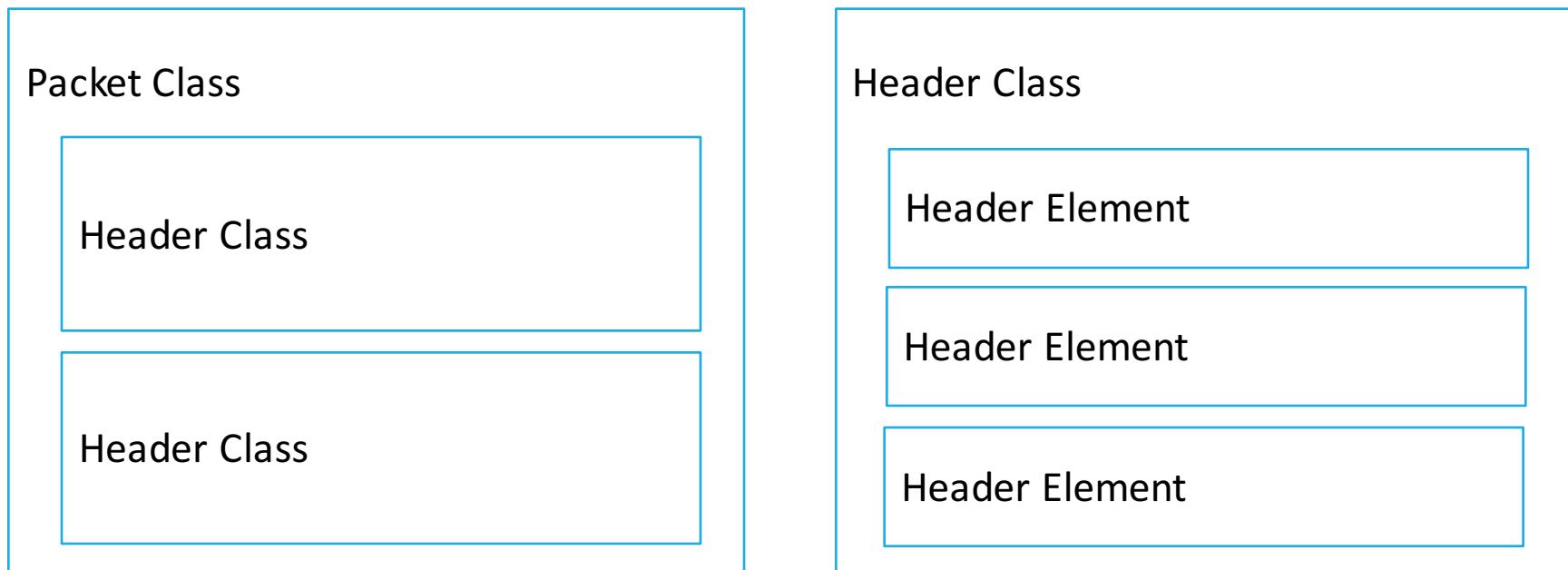
- ライブラリが既知のプロトコルに関する解析は簡単
- ライブラリが未知なプロトコルに対してどうするか
- Wiresharkとかは簡単にdissectorが追加できる
- じゃあそうしよう

拡張性

- 新たなプロトコルに関するパケットクラスの実装が簡単
- ちょっとだけパケットクラスの構造が分かってればOK

拡張性

- パケットクラス
 - プロトコルごとのヘッダのインスタンスを持つ (has-a)
- ヘッダクラス
 - ヘッダの要素を持つ (UDPヘッダならsrc_portとかdst_portとか)



新プロトコルのパケットクラスの実装

新 Packet Class

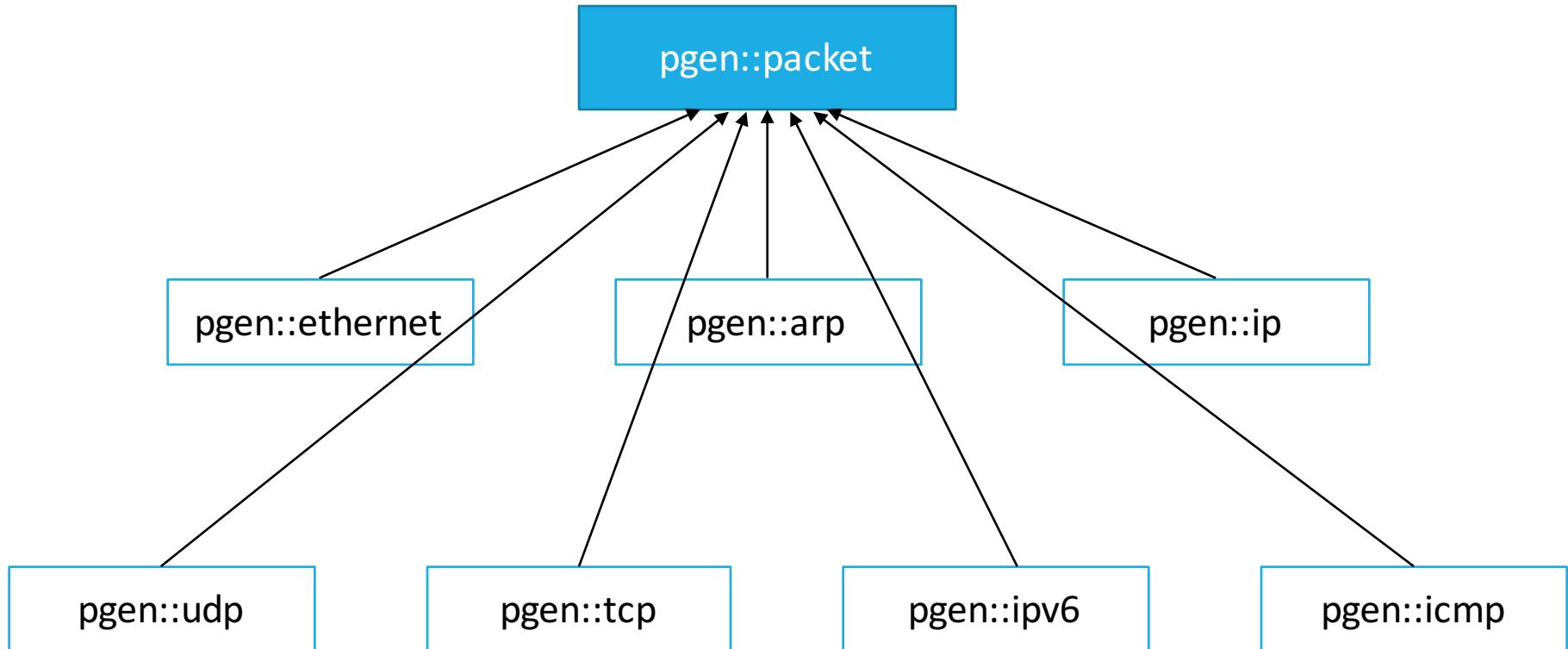
既存のHeader Class

既存のHeader Class

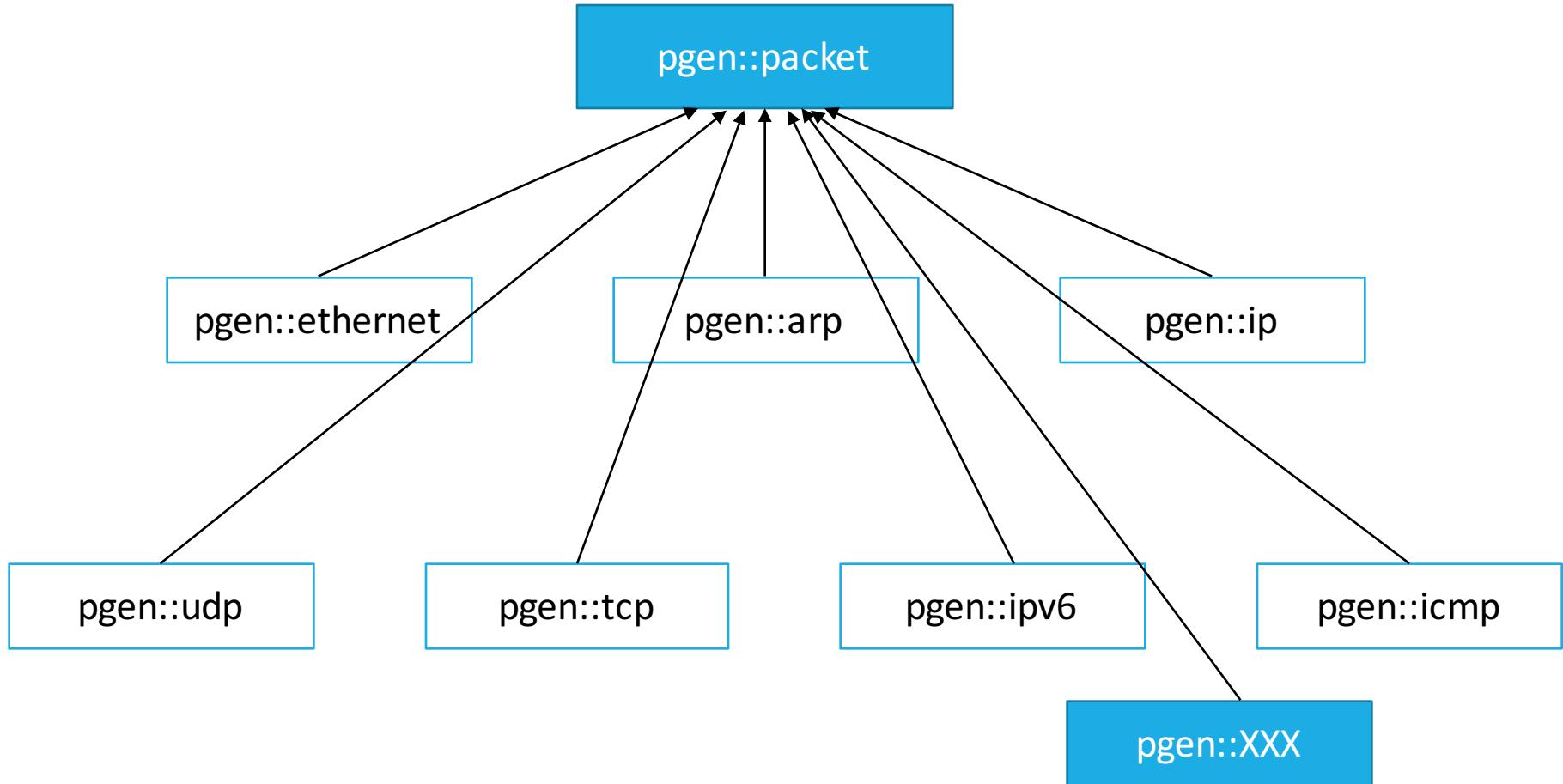
新 Header Class

- 拡張するユーザは新たなヘッダクラスを実装
- そのヘッダのバイナリ解析のコードのみを追加

パケットクラスの継承関係



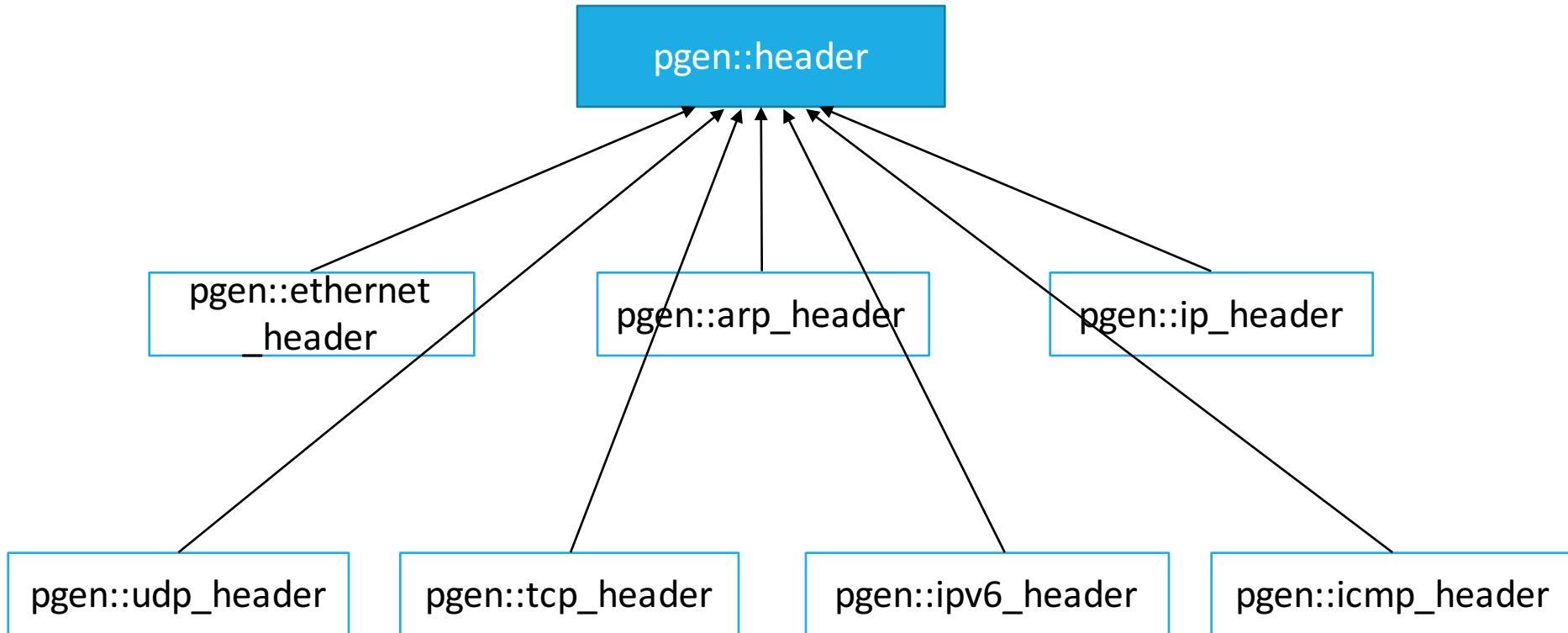
パケットクラスの継承関係



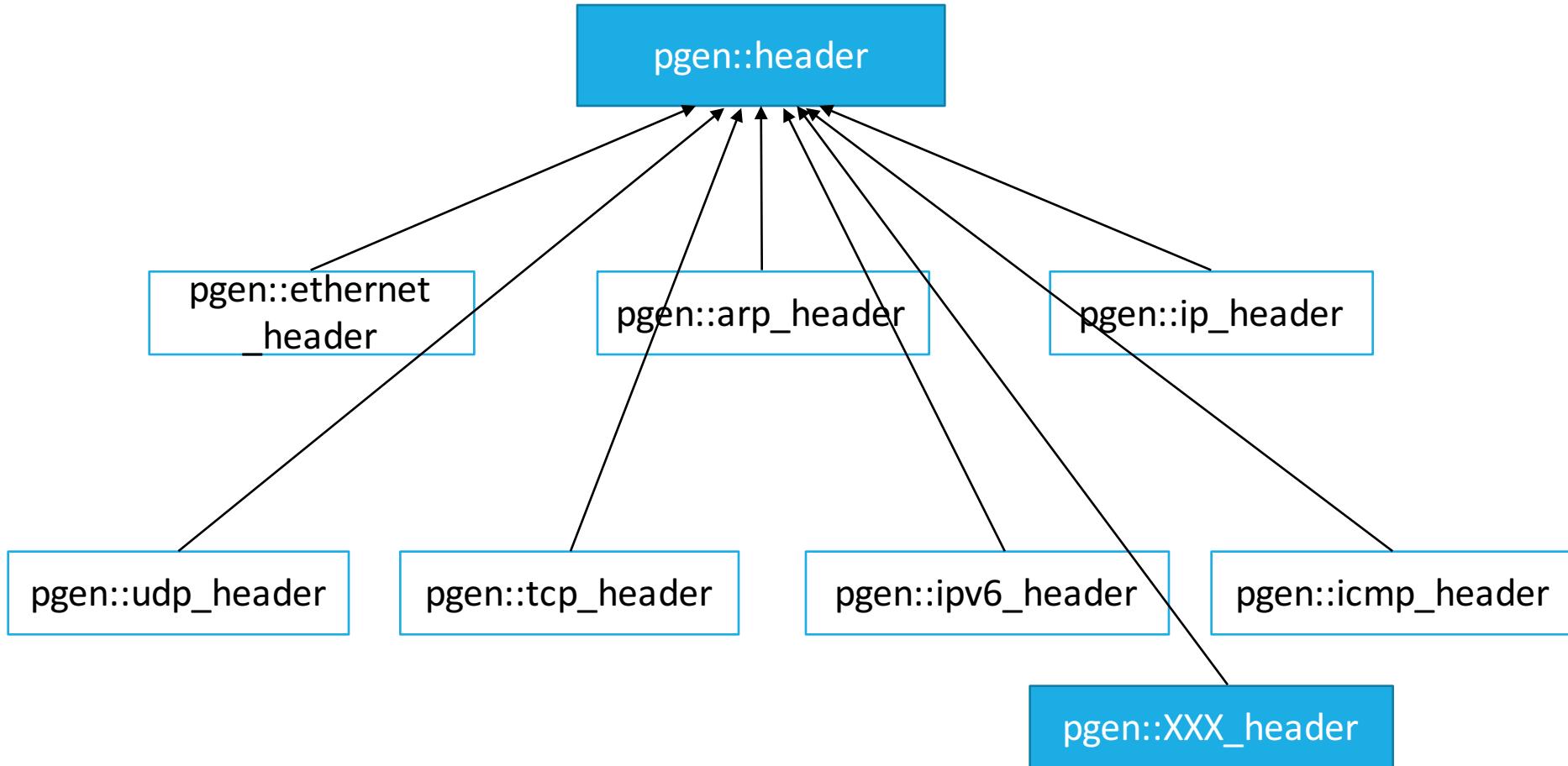
パケットクラスのメンバ関数

- 基底パケットクラスのメンバ関数 (一部)
 - `uint8_t* raw()` 生パケットのバイナリのポインタ
 - `size_t length()` パケットの長さを返す
 - `compile()` バイナリを生成`raw()`に反映する
 - `analyze(buffer, bufferlen)` バイナリを解析
- 拡張する開発者はこれらの実装を意識しなくていい

ヘッダクラスの継承関係



ヘッダクラスの継承関係

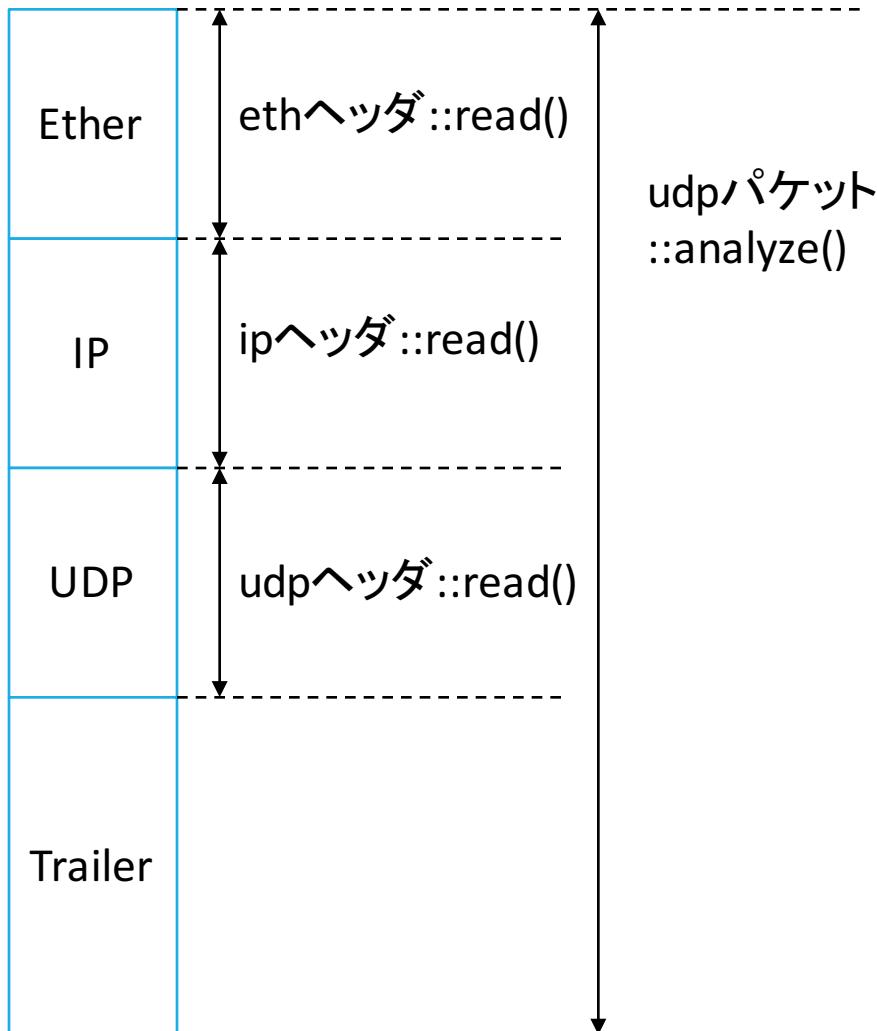


ヘッダクラスのメンバ関数

○ヘッダクラスのメンバ関数(一部)

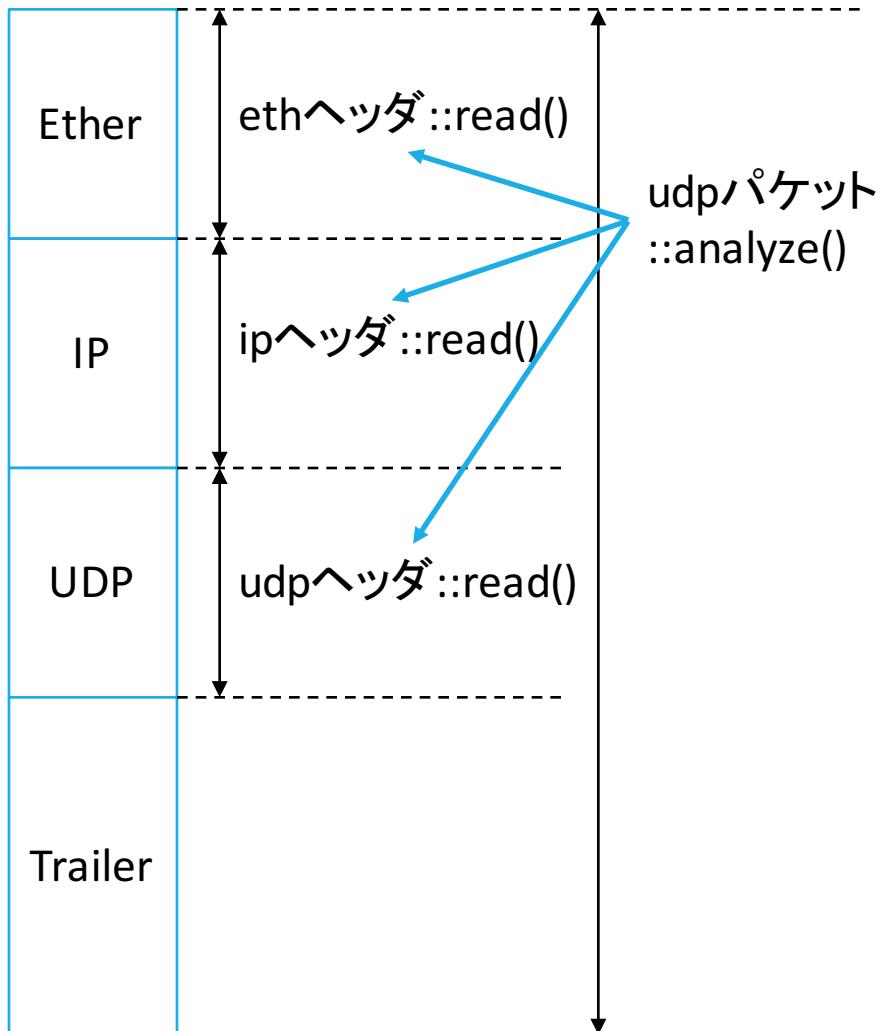
- virtual write(buffer, bufferlen) = 0 ヘッダのバイナリを書き込む
- virtual read(buffer, bufferlen) = 0 ヘッダのバイナリを読み込む
- virtual size_t length() = 0 ヘッダの長さを返す

ヘッダクラスとパケットクラスの関係



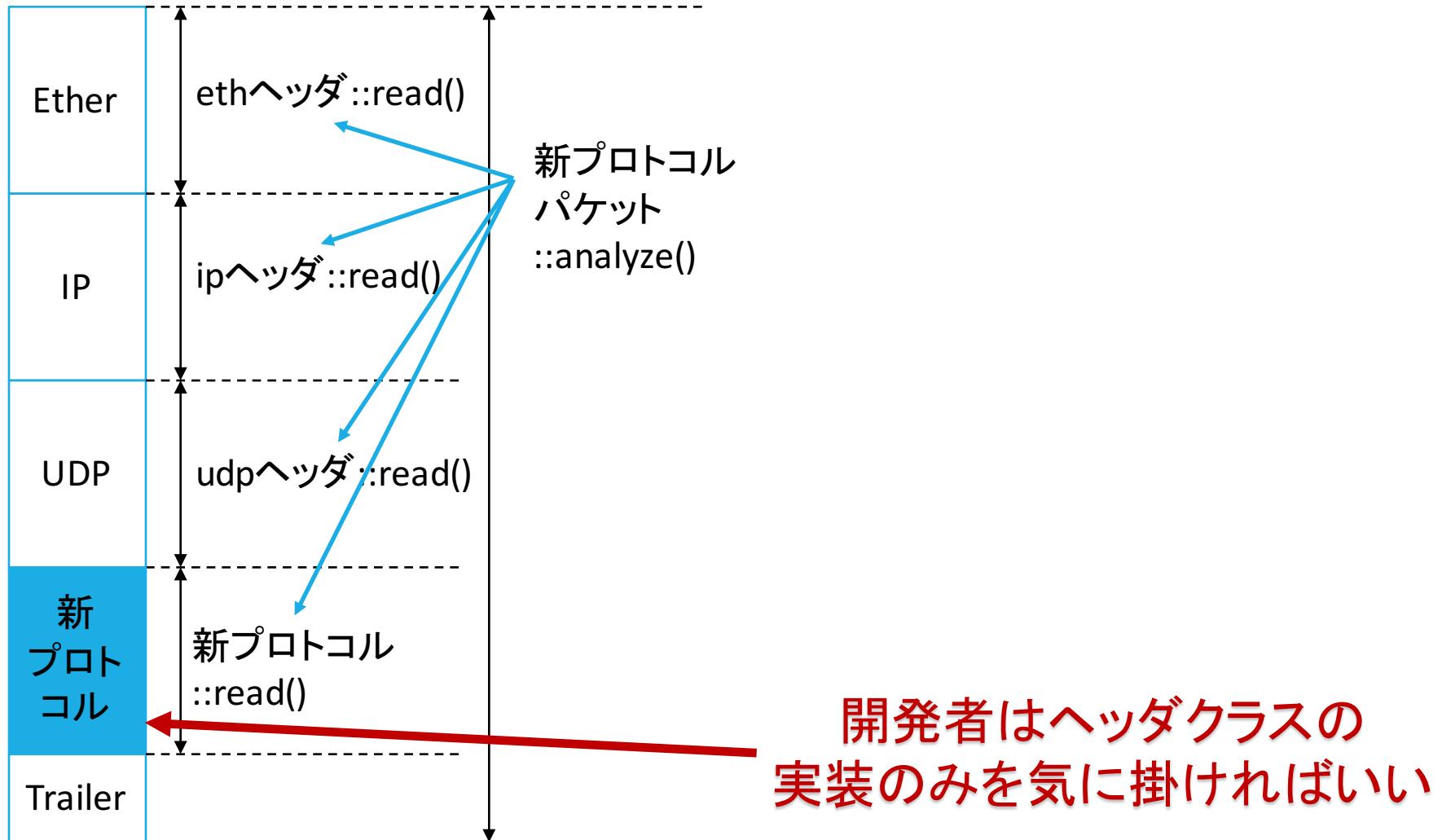
- パケットのバイナリ解析を行う場合
- analyze()は各ヘッダクラス::read()を上から呼び出して解析
- バイナリ生成も同じ構造

ヘッダクラスとパケットクラスの関係



- パケットのバイナリ解析を行う場合
- `analyze()`は各ヘッダクラス`::read()`を上から呼び出して解析
- バイナリ生成も同じ構造

新プロトコルでパケットクラスの実装



UDPパケットクラスの構造

```
class udp : public packet {  
    private:  
        void init_headers() override;  
    public:  
        pgen::udp_header UDP;  
        pgen::ipv4_header IP;  
        pgen::ethernet_header ETH;  
  
        udp();  
        udp(const void* buffer, size_t bufferlen);  
        udp(const pgen::udp& rhs);  
        udp& operator=(const std::udp& rhs);  
        void clear() override const;  
};
```

UDPパケットクラスの構造

```
class udp : public packet {  
    private:  
        void init_headers() override;  
    public:  
        pgen::udp_header UDP;  
        pgen::ipv4_header IP;  
        pgen::ethernet_header ETH;  
  
        // 決まり文句だけの  
        // 関数で1~4行程度  
        udp();  
        udp(const void* buffer, size_t bufferlen);  
        udp(const pgen::udp& rhs);  
        udp& operator=(const std::udp& rhs);  
        void clear() override const;  
};
```

UDPパケットクラスの構造

```
class udp : public packet {  
    private:  
        void init_headers() override;  
    public:  
        pgen::udp_header UDP;  
        pgen::ipv4_header IP;  
        pgen::ethernet_header ETH;  
};
```

決まり文句だけの
関数で1~4行程度

udp();
udp(const void* buffer, size_t bufferlen);
udp(const pgen::udp& rhs);
udp& operator=(const std::udp& rhs);
void clear() override const;

このみを新たに
実装すればいい

UDPヘッダクラスの構造

```
class udp_header : public header {
public:
    static const size_t min_length = sizeof(uint16_t)*4;
    static const size_t max_length = min_length;
    uint16_t src;
    uint16_t dst;
    uint16_t len;
    uint16_t check;
public:
    void clear() override;
    void summary(bool moreinfo=false) const override;
    void write(void* buffer, size_t buffer_len) const override;
    void read(const void* buffer, size_t buffer_len) override;
    size_t length() const override;
    uint16_t calc_checksum(const ipv4_header& ip,
                          const void* data, size_t datalen) const;
};
```

UDPヘッダクラスの構造

```
class udp_header : public header {  
public:  
    static const size_t min_length = sizeof(uint16_t)*4;  
    static const size_t max_length = min_length;  
  
    uint16_t src;  
    uint16_t dst;  
    uint16_t len;  
    uint16_t check;    UDPヘッダの要素  
  
public:  
    void clear() override;  
    void summary(bool moreinfo=false) const override;  
    void write(void* buffer, size_t buffer_len) const override;  
    void read(const void* buffer, size_t buffer_len) override;  
    size_t length() const override;  
    uint16_t calc_checksum(const ipv4_header& ip,  
                          const void* data, size_t datalen) const;  
};
```

ヘッダ長の
最大値と最小値

UDPヘッダクラスの構造

```
class udp_header : public header {  
public:  
    static const size_t min_length = sizeof(uint16_t)*4;  
    static const size_t max_length = min_length;  
    uint16_t src;  
    uint16_t dst;  
    uint16_t len;  
    uint16_t check;  
  
public:  
    void clear() override;  
    void summary(bool moreinfo=false) const override;  
    void write(void* buffer, size_t buffer_len) const override;  
    void read(const void* buffer, size_t buffer_len) override;  
    size_t length() const override;  
    uint16_t calc_checksum(const ipv4_header& ip,  
                          const void* data, size_t datalen) const;  
};
```

決まり文句だけの
関数で1~4行程度

UDPはチェックサム
があるので、それ用の関数

バイナリ解析など
のコードでここだけ
しっかり実装

まとめると

- パケットクラスの実装はほとんど以下の作業のみ
 - XXX_header::read(const void* buf, size_t buflen);
 - XXX_header::write(void* buf, size_t buflen);
- 新たなプロトコルのパケットクラスを実装するときも同じ
- クラス設計である程度開発者を束縛して、必要な仕事のみに専念できるように設計

Agenda

- LibPGEN
 - LibPGENとは、その設計は
 - 簡単に使う
 - 拡張性
- 少しがっつり使う
 - 新たなプロトコルに関するパケット解析
- 今後の展開と総まとめ

少しがっつり使う

- 新たなプロトコルのパケット解析
- プロトコルを自己定義
- その通信を解析できるパケットクラスを実装する

少しがっつり使う



てくも
@kumiromilk



Follow

Javaの講義、試験が「自作関数を作り記述しなさい」って問題だったから
「ズン」「ドコ」のいずれかをランダムで出力
し続けて「ズン」「ズン」「ズン」「ズン」
「ドコ」の配列が出たら「キ・ヨ・シ！」って
出力した後終了って関数作ったら満点で単位
貰ってた

[View translation](#)

RETWEETS

16,064

LIKES

12,859

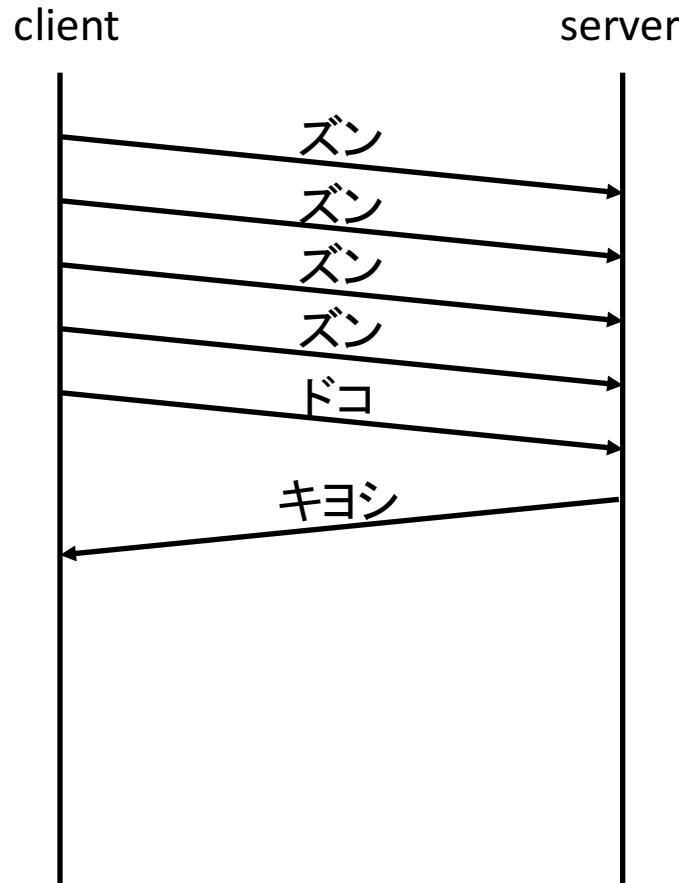


9:28 PM - 8 Mar 2016



...

ズンドコプロトコル



- ズンドコプロトコル over UDP
- 勝手にいろいろ改変して
- ズン、ズン、ズン、ズン、ドコの順に受け取ると”キヨシ“を返すプロトコル

ズンドコプロトコル

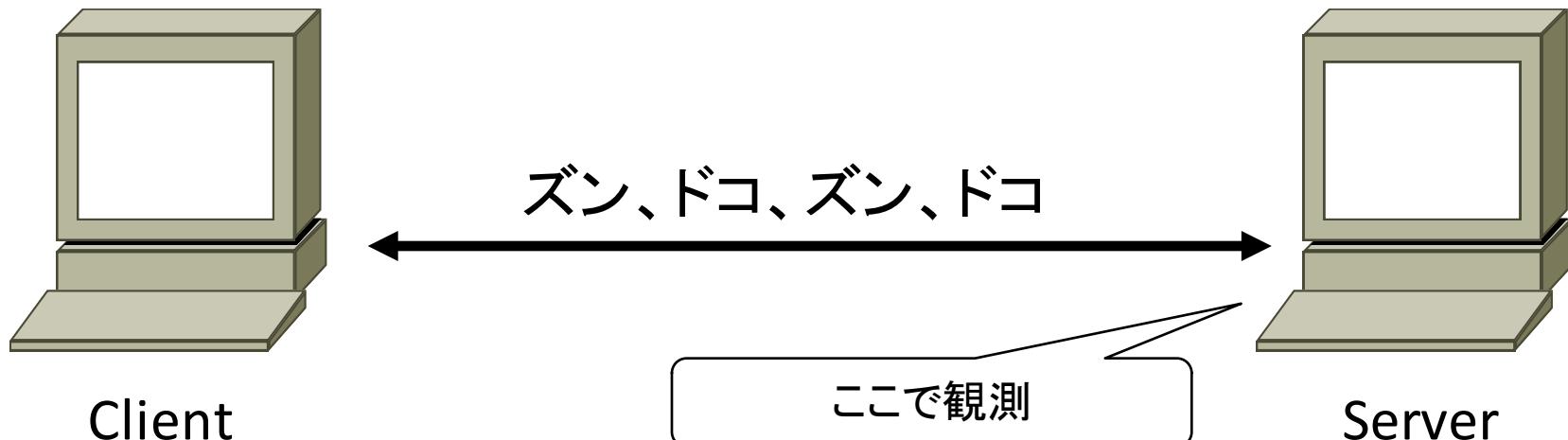
- サーバはズンズンズンズンドコの順に通信を受け付けたら最後のドコを送ったホストにキヨシを返信
- それ以外は受け取ったメッセージをエコーする
- ヘッダフィールドは以下とする

type (16bit)	msglen (16bit)
message (msglen byte)	

type
0 : ズン
1 : ドコ
2 : キヨシ

内容

- デモ用にこのプロトコルで通信するサーバを用意
- サーバ側で別のプロセスで”キヨシ”を当てた端末を表示



内容

- あらかじめパケットをキャプチャ, pcapngで保存
- そのパケットを解析して、
キヨシを当てているホストを羅列する

準備

- pgen::zundoko パケットクラスの実装
 - 要するに pgen::zundoko_header クラスの実装
- サーバクライアントの設計実装 (省略)
- 解析するスクリプトを LibPGEN を使って作成

準備

```
#include <pgen2.h>
#include <stdio.h>
#define ZDPORT 9988
const char* in = "in.pcapng";
int main() {
    try {
        pgen::pcapng_stream pcapng(in, pgen::open_mode::pcapng_read);
        while (1) {
            uint8_t buf[10000];
            size_t recvlen = pcapng.recv(buf, sizeof buf);

            pgen::packet_type type = pgen::module::detect(buf, recvlen);
            if (type == pgen::packet_type::udp) {
                pgen::udp udp(buf, recvlen);
                if (udp.UDP.src==ZDPORT || udp.UDP.dst==ZDPORT) {
                    pgen::zundoko pack(buf, recvlen);
                    if (pack.ZUNDOKO.type == pgen::zundoko::type::kiyoshi) {
                        printf("Success %s msg=\"%s\" \n", pack.IP.dst.str().c_str(),
                               pack.ZUNDOKO.msg.c_str());
                    }
                }
            }
        } catch (std::exception& e) {
            printf("%s \n", e.what());
        }
    }
}
```

KIYOSHIを当てた端末を
表示するスクリプト

pge::zundoko_headerクラス

```
class zundoko_header : public header {
public:
    static const size_t min_length = sizeof(uint16_t)*2;
    static const size_t max_length = min_length + 100;

    uint16_t type;
    uint16_t msg_len;
    std::string msg;

public:
    void clear() override;
    void summary(bool moreinfo=false) const override;
    void write(void* buffer, size_t buffer_len) const override;
    void read(const void* buffer, size_t buffer_len) override;
    size_t length() const override;

};
```

pge::zundoko パケットクラス

```
class zundoko : public packet {
private:
    void init_headers() override;

public:
    struct type {
        static const uint16_t zun      = 0;
        static const uint16_t doko     = 1;
        static const uint16_t kiyoshi  = 2;
    };

    pgen::zundoko_header ZUNDOKO;
    pgen::udp_header UDP;
    pgen::ipv4_header IP;
    pgen::ethernet_header ETH;

    zundoko();
    zundoko(const void* buffer, size_t bufferlen);
    zundoko(const pgen::zundoko& rhs);
    void clear() override;
};
```

pigen::zundoko_header::read()

```
void pundoko_header::read(const void* buffer, size_t buffer_len) {
    if (buffer_len < min_length) {
        throw pigen::exception("pigen::zundoko_header::read: buflen is too small");
    }

    const uint8_t* buffer_point =
        reinterpret_cast<const uint8_t*>(buffer);

    const struct pundoko_struct* zd_head =
        reinterpret_cast<const pundoko_struct*>(buffer_point);
    type      = ntohs(zd_head->type      );
    msg_len = ntohs(zd_head->msg_len);

    const char* message =
        reinterpret_cast<const char*>(buffer_point+sizeof(pundoko_struct));
    if (msg_len > 0)
        msg = message;
}
```

←この辺にバグがありました。
ごめんなさい。

pigen::zundoko_header::write()

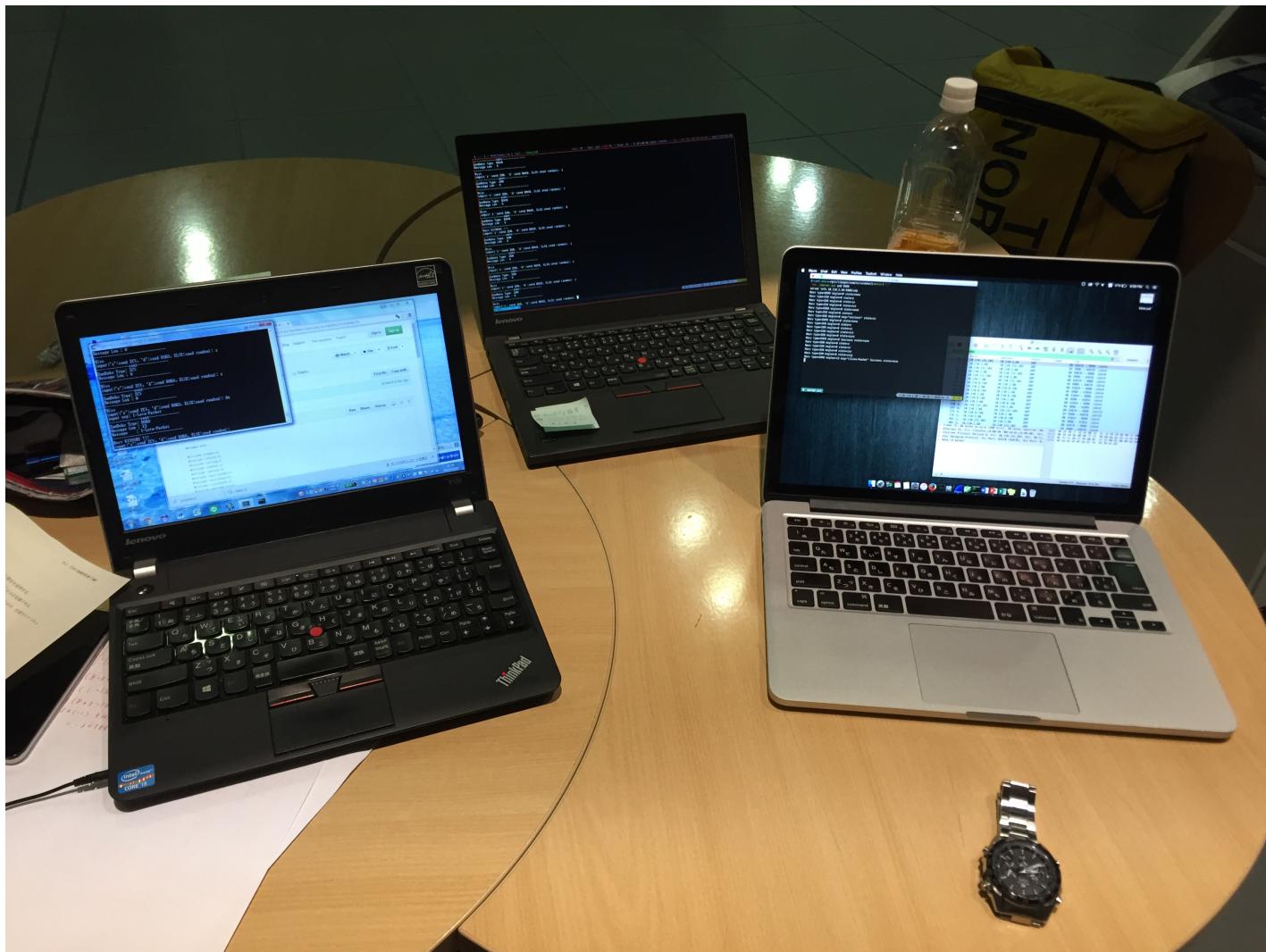
```
void pundoko_header::write(void* buffer, size_t buffer_len) const {
    if (buffer_len < min_length) {
        throw pigen::exception("pigen::zundoko_header::write: buflen is too small");
    }

    uint8_t* buffer_point =
        reinterpret_cast<uint8_t*>(buffer);

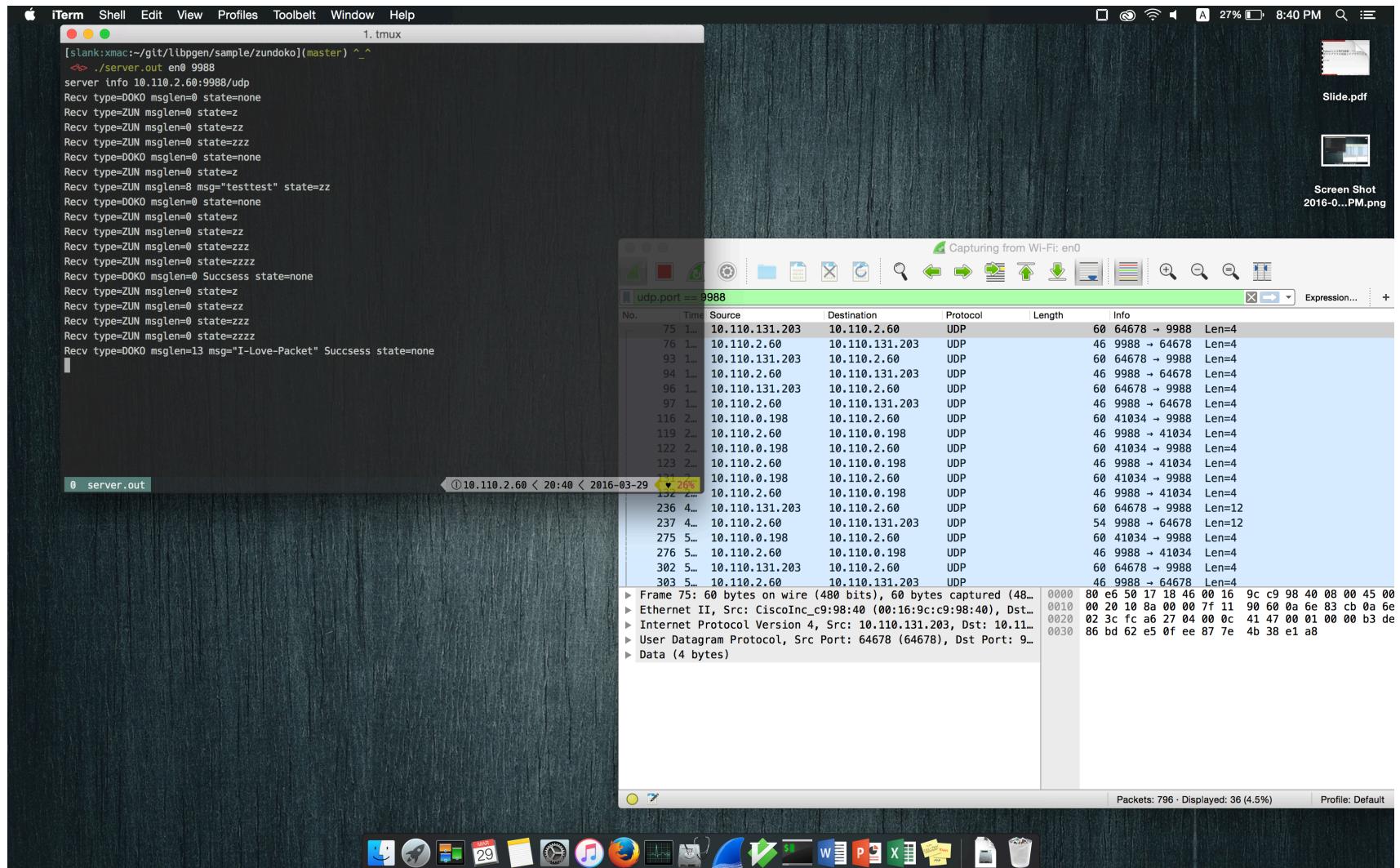
    struct pundoko_struct* zd_head =
        reinterpret_cast<pundoko_struct*>(buffer_point);
    zd_head->type      = htons(type);
    zd_head->msg_len   = htons(msg_len);

    char* message =
        reinterpret_cast<char*>(buffer_point+sizeof(pundoko_struct));
    memcpy(message, msg.c_str(), msg_len);
}
```

実験中



実験中



実行結果

- 以下の2つの端末がKIYOSHIを当てたらしい。

```
[slank:xmac:~/git/libpgen/sample/zundoko](+master) ^_~  
<--> ./a.out  
Success 10.110.0.198 msg=""  
Success 10.110.131.203 msg="I-Love-Packet"  
pgen::pcapng_stream::recv: is end of file  
[slank:xmac:~/git/libpgen/sample/zundoko](+master) ^_~  
<-->
```

- 本題はここじゃ無い

まとめ

- 新たなプロトコルの解析が簡単にできる**
- くだらない?いや楽しい。

Agenda

- LibPGEN
 - LibPGENとは、その設計は
 - 簡単に使う
 - 拡張性
- 少しがっつり使う
 - 新たなプロトコルに関するパケット解析
- 今後の展開と総まとめ

今後の展開

- 他にもパケットで遊ぶために必要な機能を検討中
 - 使う系
 - ARPテーブルの実装
 - TCPプロトコルスタックの実装 ←
 - ルータの実装 ← 絶対この辺楽しい
 - 様々なプロトコルのサポート
 - より拡張しやすい設計
 - Winでも使えるように
 - 高速にIOとか
 - 同期、非同期、なにそれ

ラボニュースまとめ



herumi
@herumi



Following

バグ特定そのものも難しかったけど、長期戦だったので諦めないようにするのも大変でした（私は鬼軍曹と呼ばれているらしい）。

@starpozさんお疲れさまでした。

Cybozu InsideOut 「半年かかったバグ調査の顛末は」

[blog.cybozu.io/entry/2016/01/ ...](http://blog.cybozu.io/entry/2016/01/)

View translation

Cybozu Inside Out
サイボウズエンジニアのブログ

半年かかったバグ調査の顛末は - Cybozu Inside Out | サイボ...

サイボウズ・ラボの光成です。 今回は原因究明に半年以上かかったバグ調査の紹介をいたします。

blog.cybozu.io

RETWEETS
28 LIKES
18



○ビシバシたくさん指摘してくださるので学ぶことが多い

○鬼軍曹...

ラボユースまとめ

- 光成さんのコードレビュー、設計の考えが最も大きな収穫
- 前にも増して無力さ痛感
 - コードを読む力
 - 英語力
 - エディタ力
 - スマートなコードを書く力

ラボユースまとめ

- サイボウズ・ラボ社様のサポートで楽しく春休みを過ごすことができました
- 特にメンターとしてご指導をしてくださった光成さん、ありがとうございました。
- セキュリティ・キャンプ講師陣、修了生の方にも意見やアドバイスをいただきました、ありがとうございます。
- Github: slankdev/libpgen

ありがとうございました

aabb	ccdd	eeff	0011	2233	4455	0800	4500
2800	0001	0000	4006	b9a2	c0a8	b302	c0a8
6501	3039	3039	0000	0000	0000	0000	5001
2000	961c	0000					