Pythonで作るネットワークスニッファー

lapis-zero09

ネットワークスニッファー

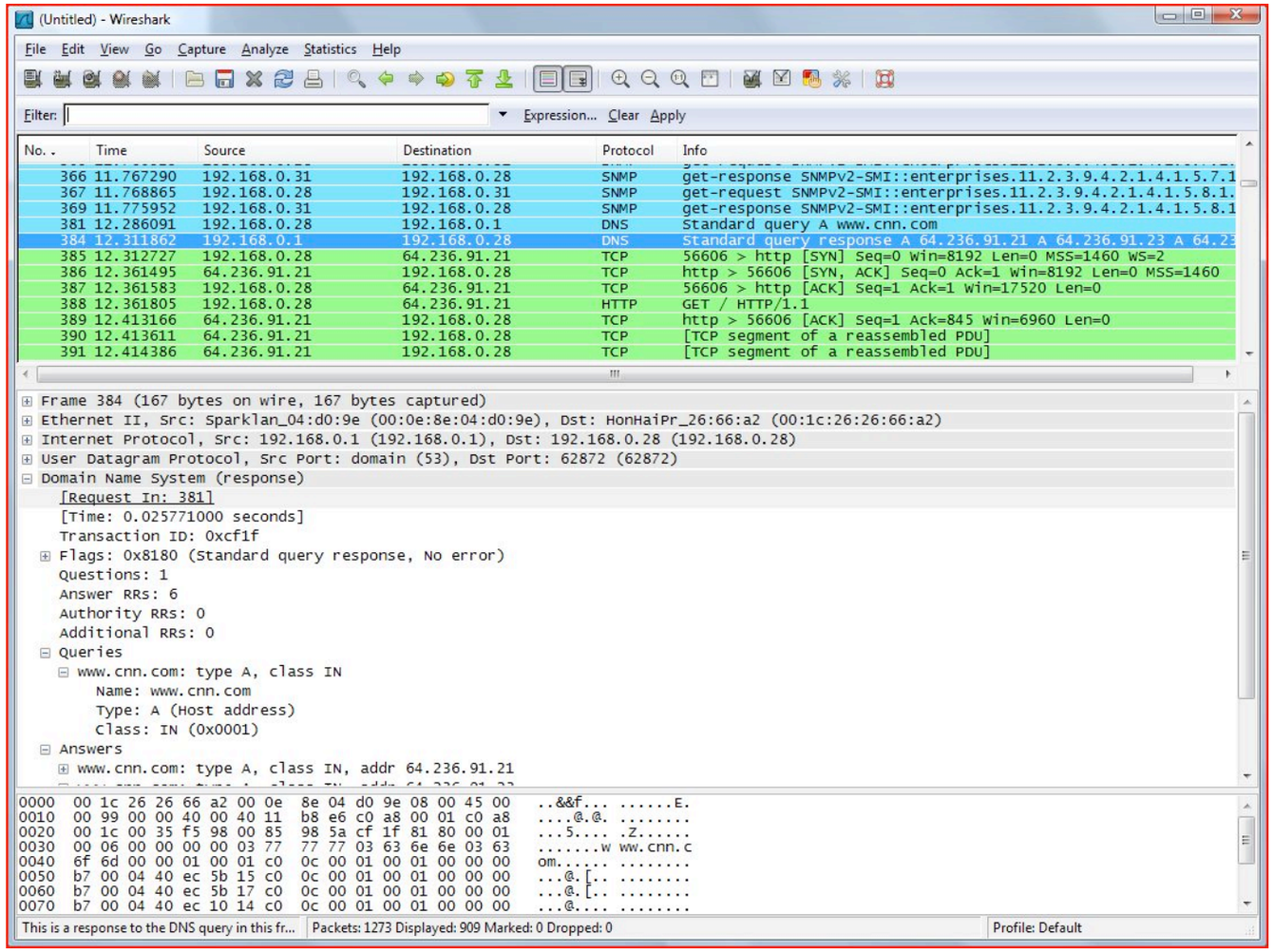
- ネットワークスニッファーとは
  - sniffとは
    - くんくんかぐ，匂いを嗅ぐ.etc..
  - ネットワークを流れるパケットをモニタリングし，トラフィックを調査したり，パケットをキャプチャすることで障害を解析したりする目的で利用
  - 標的マシンが送受信するパケットの観測を可能にするため，攻撃の前後で使用される

→　暗号化されていないパスワードやデータを盗聴することが可能

パケットアナライザ，ネットワークアナライザとも

ネットワークスニッファー

- 通信の監視によく利用されるもの 
  - WireShark(希望があればあとで使ってみせます)
  - Scapy(pythonライブラリ)(後述)



UDPベースのスニッファー

- 標的ネットワークで動作中のホストを発見するUDPベースのスニッファーを作る

- 攻撃者は事前調査を行ったり，攻撃対象を絞り込んだりするため，ネットワーク上に潜む全ての標的を見つけようとする

- 特定のIPアドレスでホストが稼働しているか判断するために，たいていのOSに共通の既知の挙動を利用する

UDPベースのスニッファー

- OSに共通の既知の挙動とは
- UDPデータグラムをホスト上の閉じたポートに送ると，当該ポートに到達できないことを示すICMPメッセージが送信される

→ このICMPを受信できるということはホストが稼働している

- UDPを選択した理由

→ サブネット全体にメッセージを送り，ICMPレスポンスを待つことについて，オーバーヘッドが存在しない

※オーバーヘッド

- システムやプロトコルの処理の負荷
- 本来やりたい事以外にかかってしまう余計なコスト

UDPベースのスニッファー

- なんか取れた

```
% sudo python sniffer.py                                                    [master]
('E\xc0@\x00\x00\x00\x00\x003\x01\xd4Z\xd8:\xc5\xee\ne\n\x01\x00\x00\xd6\x8fW\x1e\x0
0\x00X\xbd\x84\r\x00\x01\n\x83\x08\t\n\x0b\x0c\r\x0e\x0f\x10\x11\x12\x13\x14\x15\x16
\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#$%&\'()*+,-./01234567', ('216.58.197.238', 0
))
```

→ 最初のICMP pingリクエスト

```
% ping google.com
PING google.com (216.58.197.238): 56 data bytes
64 bytes from 216.58.197.238: icmp_seq=0 ttl=51 time=34.337 ms
```

→ それに対するレスポンス

1パケットのみキャプチャできた!!

UDPベースのスニッファーの強化

- 1パケットのみできてもあまり意味がない．．．

→ 複数のパケットを処理できるようにする

→ 中身をデコードする

  - 欲しい情報
    - プロトコルタイプ(TCP, UDP, ICMP)，　destination, source
    - 以下の典型的なIPv4ヘッダの構造に基づいてデコード

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
バージョン				ヘッダ長				サービス種別								全長															
識別子																フラグ			断片位置											
生存時間								プロトコル								チェックサム															
送信元アドレス																															
宛先アドレス																															
拡張情報																															
データ																															

UDPベースのスニッファーの強化

IPヘッダのC定義

```
struct ip {
    u_char ip_hl:4;
    u_char ip_v:4;
    u_char ip_tos;
    u_short ip_len;
    u_short ip_id;
    u_short ip_off;
    u_char ip_ttl;
    u_char ip_p;
    u_char ip_sum;
    u_char ip_src;
    u_char ip_dst;
}
```

Pythonのctypesで再定義

```
_fields_ = [
    ("ihl",          c_uint8, 4),
    ("version",      c_uint8, 4),
    ("tos",          c_uint8),
    ("len",          c_uint16),
    ("id",           c_uint16),
    ("offset",       c_uint16),
    ("ttl",          c_uint8),
    ("protocol_num", c_uint8),
    ("sum",          c_uint16),
    ("src",          c_uint32),
    ("dst",          c_uint32)
]
```

UDPベースのスニッファーの強化

- 取れた

```
% sudo python sniffer_ip_header_decode.py
Password:
Protocol: ICMP 216.58.197.174 -> 10.101.10.1
Protocol: ICMP 216.58.197.174 -> 10.101.10.1
Protocol: ICMP 216.58.197.174 -> 10.101.10.1
Protocol: ICMP 216.58.197.174 -> 10.101.10.1
Protocol: ICMP 216.58.197.174 -> 10.101.10.1
Protocol: ICMP 216.58.197.174 -> 10.101.10.1
Protocol: ICMP 216.58.197.174 -> 10.101.10.1
Protocol: ICMP 216.58.197.174 -> 10.101.10.1
```

- IPヘッダをデコードできた
- 同じようにICMPメッセージのデコードをする
  - ICMPメッセージはタイプ，コード，チェックサムの3フィールドを持つ
  - タイプとコードは受信したICMPメッセージの種類を表している

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
タイプ								コード								チェックサム															
未使用								長さ								未使用															
IPヘッダ + 元データグラムの先頭部分																															

UDPベースのスニッファーの強化

- 欲しい情報
  - 閉じたポートから返ってくる，当該ポートに到達できないことを示すICMPメッセージ
- それを満たすICMPのフィールド条件
  - タイプ3・・・宛先到達不可能(Destination Unreachable Message)
  - コード3・・・ポート到達不能(Destination port unreachable)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
タイプ (3)								コード								チェックサム															
未使用								長さ								次HopのMTU															
IPヘッダ + 元データのデータグラムの先頭部分																															

- 最初の8ビットにタイプ，次の8ビットにコードが格納されている
- 元メッセージのIPヘッダとデータの先頭8バイトが含まれている

→ スキャナによって生成されたICMPレスポンスか確認可能

UDPベースのスニッファーの強化

- できた

```
% sudo python sniffer_with_icmp.py
Protocol: ICMP 172.217.25.110 -> 10.101.10.1
ICMP -> Type: 0 Code: 0
Protocol: ICMP 172.217.25.110 -> 10.101.10.1
ICMP -> Type: 0 Code: 0
Protocol: ICMP 172.217.25.110 -> 10.101.10.1
ICMP -> Type: 0 Code: 0
Protocol: ICMP 130.158.3.235 -> 10.101.10.1
ICMP -> Type: 3 Code: 9
Protocol: ICMP 130.158.3.235 -> 10.101.10.1
ICMP -> Type: 3 Code: 9
Protocol: ICMP 130.158.3.235 -> 10.101.10.1
ICMP -> Type: 3 Code: 9
Protocol: ICMP 130.158.3.235 -> 10.101.10.1
ICMP -> Type: 3 Code: 9
```

- スキャナもついでに作る
- https://github.com/lapis-zero09/sniffer/blob/master/scanner.py
- 悪用厳禁
- よく使われるスキャナ
  - nmap

暗号化されていない通信を盗聴する

暗号化されていない通信を盗聴する

- telnet
  - 汎用的な双方向8ビット通信を提供する端末間およびプロセス間の通信プロトコル
  - RFC 854
  - リモートにあるサーバを端末から操作できるようにする
  - 基本的にポート番号23番
  - 認証も含めすべての通信を暗号化せずに平文のまま送信する

- FTP
  - ネットワークでファイルの転送を行うための通信プロトコル
  - インターネット初期から存在する古いプロトコル
  - セキュア（安全）なプロトコルとして設計されていない
  - すべての通信内容を暗号化せずに転送する

攻撃者がIPを見つけた後にどうするか