

輪読会 #1

Attacking Network Protocols #2

Caputuring Application Traffic

ed

なぜ最初にキャプチャをやるのか

- ・ネットワークを通した攻撃から保護までの流れ

1. トラフィックキャプチャ
2. 使用プロトコル解析
3. 脆弱性発見
4. 悪用
5. 保護

データがないと使用プロトコルもわからず攻撃どころじゃない

パケットバイナリが欲しいのではなく

どんなプロトコルで通信をしているのか知りたい

■ トラフィックキャプチャ

- パッシブキャプチャ
 - Wireshark
 - システムコールトレース
 - strace
 - Dtrace
 - Windows でのトレース
- アクティブキャプチャ
 - ネットワークプロキシ
 - ポートフォワーディングプロキシ
 - SOCKS プロキシ
 - HTTP プロキシ
 - リバース HTTP プロキシ

■ パッシブキャプチャ

- ・ クライアントアプリ, サーバーアプリ間動作
- ・ クライアント, サーバーを制御できない場合の唯一のキャプチャ手法
 - 通常はパッシブで怪しいトラフィックを検出し, アクティブでプロトコル分析
- ・ メリット
 - クライアント, サーバー通信が阻害されない
 - アプリケーションの変更, 再設定が不要
- ・ デメリット
 - 低レベル情報のため解釈が困難
 - 暗号化されたプロトコルの復号, 圧縮の無効化, トラフィック変更, 操作などが困難

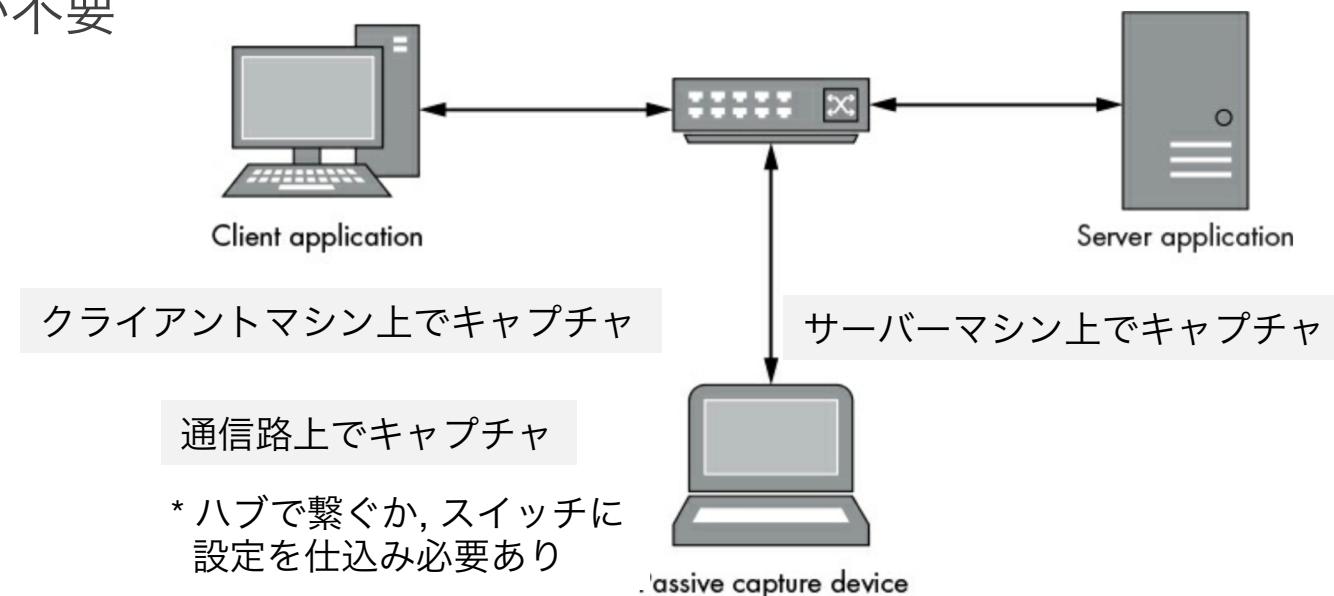
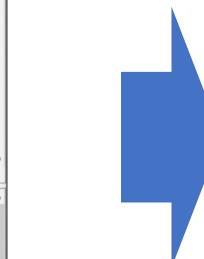
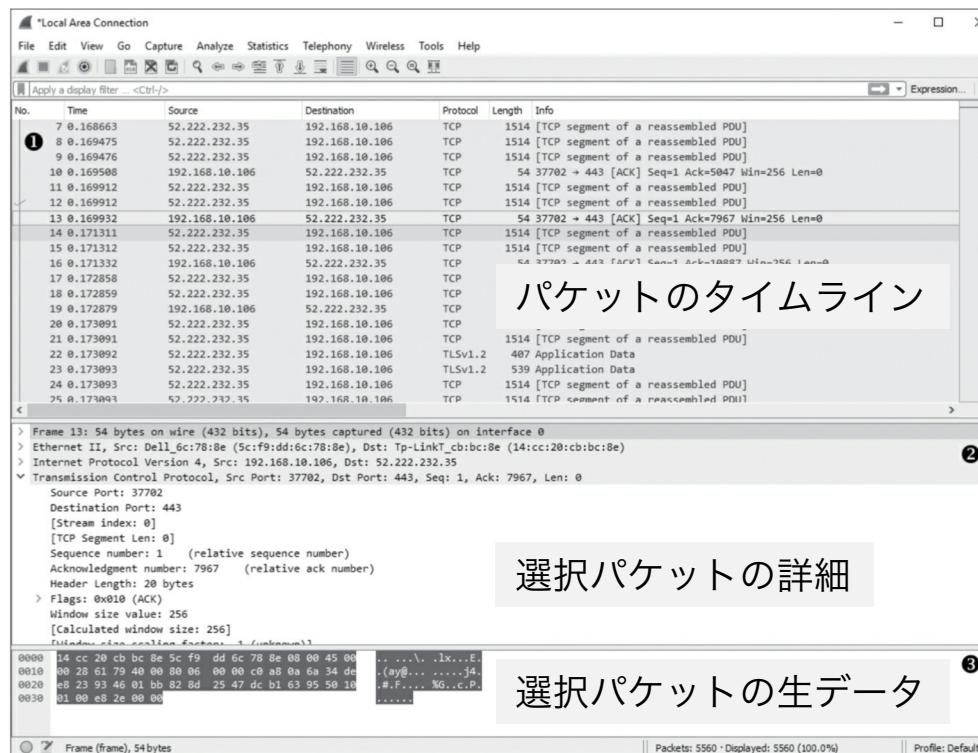


Figure 2-1: An example of passive network capture

パッシブキャプチャ例 | Wireshark

- ・パケットスニッフィングアプリケーション(5章で詳細なフィルタ作成)
- ・キャプチャ方法
 - プロミスキヤス(無差別)モードを指定
 - インターフェイスを指定



TCPフロー解析
順序通り処理

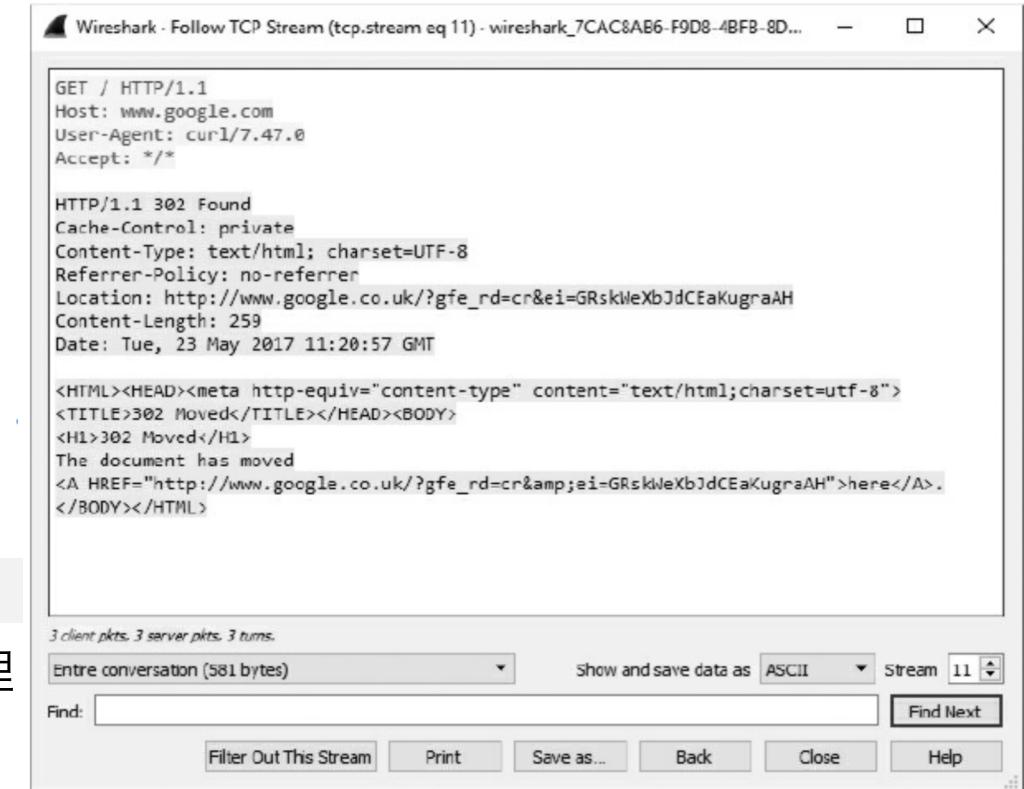


Figure 2-3: Following a TCP stream

Figure 2-2: The default Wireshark view

■ パッシブキャプチャ例 | システムコールトレース

6

- ・システムコールトレースによりアプリケーションのプロトコルを分析
- ・ネットワーク通信にはシステムコールを呼び出す
 - socket | 新規ソケットインターフェイスを作成
 - connect | IP/Portで指定したソケットと接続
 - bind | 自ソケットのIP/Portを固定
 - recv | ソケットを通しデータを受信
 - send | ソケットを通しデータを送信

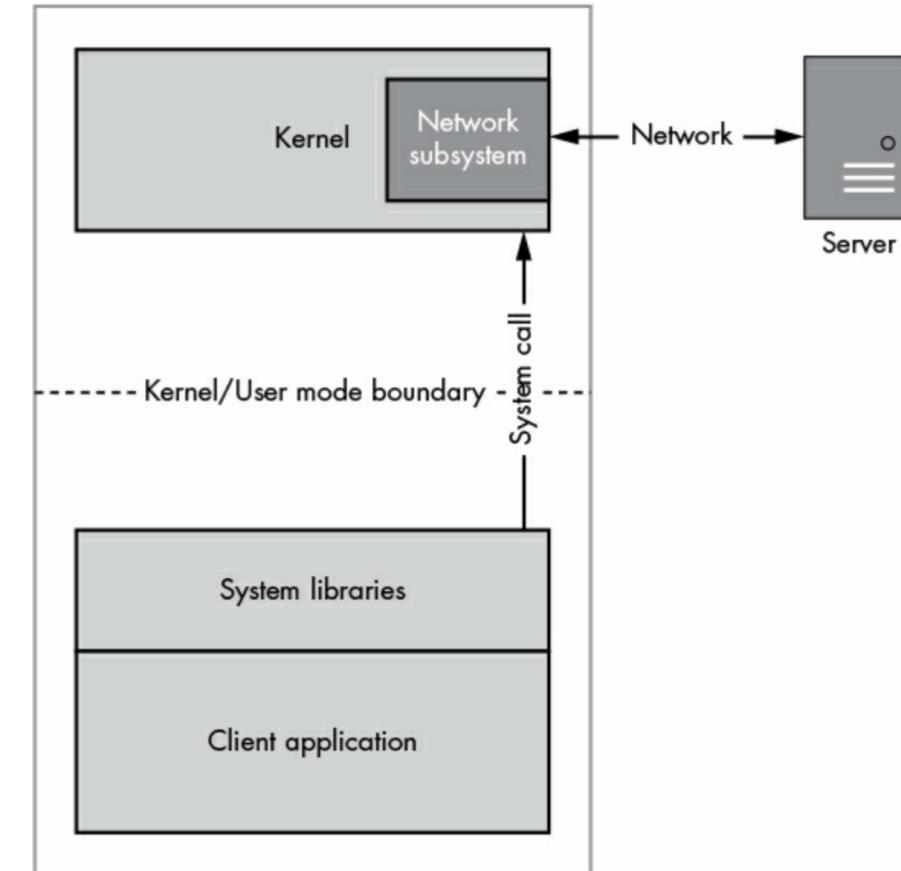


Figure 2-4: An example of user-to-kernel network communication via system calls

■ パッシブキャプチャ例 | システムコールトレース

7

- strace (syscall trace)
 - 特権なしでユーザプログラムのシステムコール監視が可能

```
$ strace -e trace=network,read,write customapp  
--snip--
```

❶ socket(PF_INET, SOCK_STREAM, IPPROTO_TCP) = 3	新規TCPソケットを作成
❷ connect(3, {sa_family=AF_INET, sin_port=htons(5555), sin_addr=inet_addr("192.168.10.1")}, 16) = 0	ポート 5555, IP 192.168.10.1 に TCP 接続確立
❸ write(3, "Hello World!\n", 13) = 13	Hello World! を送信
❹ read(3, "Boo!\n", 2048) = 5	Boo! を受信

Listing 2-1: Example output of the strace utility

パッシブキャプチャ例 | システムコールトレース

8

- Dtrace (syscall trace)
 - スクリプトによってシステム全体のトレースが可能

```
/* traceconnect.d - A simple DTrace script to monitor a connect system call */
❶ struct sockaddr_in {
    short          sin_family;           IPv4で使われるアドレス構造体
    unsigned short  sin_port;
    in_addr_t      sin_addr;
    char           sin_zero[8];
};

❷ syscall::connect:entry
❸ /arg2 == sizeof(struct sockaddr_in)/
{
    ❹ addr = (struct sockaddr_in*)copyin(arg1, arg2);   arg0 ~ 2 にはconnectシステムコールの引数が格納
    ❺ printf("process:'%s' %s:%d", execname, inet_ntop(2, &addr->sin_addr),
            ntohs(addr->sin_port));
}
```

トレースポイントを指定 (dtrace -l でトレースポイント一覧取得)
フィルタを指定 (アドレス構造体の大きさで IPv4 と判断)
プロセス名, 宛先IP, ポート番号を表示



スクリプトを traceconnect.d に保存
dtrace -s traceconnect.d を実行

```
process:'Google Chrome' 173.194.78.125:5222
process:'Google Chrome' 173.194.66.95:443
process:'Google Chrome' 217.32.28.199:80
process:'ntpd'           17.72.148.53:123
process:'Mail'            173.194.67.109:993

process:'syncdefaultsd'  17.167.137.30:443
process:'AddressBookSour' 17.172.192.30:443
```

パッシブキャプチャ例 | システムコールトレース

9

- Windows のトレース
 - ネットワーク機能はドライバを介して呼び出し、イベントを生成
 - Process Monitor を利用

Time ...	Process Name	PID	Operation	Path	Result	Detail
12:26...	spoolsv.exe	3212	UDP Send	onyx:55084 -> 192.168.10.70:snmp	SUCCESS	Length: 78, seqnum: 0, connid: 0
12:26...	CDASrv.exe	10672	UDP Send	onyx:51358 -> 192.168.10.70:snmp	SUCCESS	Length: 50, seqnum: 0, connid: 0
12:26...	spoolsv.exe	3212	UDP Send	onyx:55084 -> 192.168.10.70:snmp	SUCCESS	Length: 112, seqnum: 0, connid: 0
12:26...	msvsmon.exe	6580	TCP Receive	onyx:37105 -> onyx:37106	SUCCESS	Length: 221, seqnum: 0, connid: 0
12:26...	devenv.exe	18088	TCP Send	onyx:37106 -> onyx:37105	SUCCESS	Length: 221, starttime: 118739281.
12:26...	devenv.exe	18088	TCP Receive	onyx:37106 -> onyx:37105	SUCCESS	Length: 86, seqnum: 0, connid: 0
12:26...	msvsmon.exe	6580	TCP Send	onyx:37105 -> onyx:37106	SUCCESS	Length: 86, starttime: 118739281.
12:26...	CDASrv.exe	10672	UDP Send	onyx:51363 -> 192.168.0.19:snmp	SUCCESS	Length: 46, seqnum: 0, connid: 0
12:26...	devenv.exe	18088	TCP Disconnect	onyx:37775 -> onyx:49154	SUCCESS	Length: 0, seqnum: 0, connid: 0
12:26...	CDASrv.exe	10672	UDP Send	onyx:51368 -> onyx:snmp	SUCCESS	Length: 46, seqnum: 0, connid: 0
12:26...	googledrivesyn...	9552	TCP Send	onyx:35685 -> 66.102.1.125:5222	SUCCESS	Length: 53, starttime: 118739354.
12:26...	CDASrv.exe	10672	UDP Send	onyx:51373 -> 192.168.10.70:snmp	SUCCESS	Length: 50, seqnum: 0, connid: 0
12:26...	chrome.exe	12792	TCP Receive	onyx:37738 -> 104.19.194.102:https		
12:26...	chrome.exe	12792	TCP Receive	onyx:37738 -> 104.19.194.102:https		
12:26...	chrome.exe	12792	TCP Disconnect	onyx:37738 -> 104.19.194.102:https		
12:26...	CDASrv.exe	10672	UDP Send	onyx:51378 -> 192.168.10.105:snmp		
12:26...	chrome.exe	12792	TCP Receive	onyx:37736 -> 104.20.92.43:https		
12:26...	chrome.exe	12792	TCP Receive	onyx:37736 -> 104.20.92.43:https		
12:26...	chrome.exe	12792	TCP Disconnect	onyx:37736 -> 104.20.92.43:https		
12:26...	svchost.exe	1992	UDP Send	c0a8:a6:300:0:3071:9f3a:82e7ffff:65454 -> 808:808:5f57:7261:7070:6572:2		
12:26...	svchost.exe	1992	UDP Receive	onyx:65454 -> google-public-dns-a.google.com:domain		
12:26...	devenv.exe	18088	TCP Disconnect	onyx:37776 -> onyx:49154		
12:26...	devenv.exe	18088	TCP Disconnect	onyx:37777 -> onyx:49154		
12:26...	svchost.exe	1992	UDP Send	c0a8:a6:300:0:3071:9f3a:82e7ffff:62005 -> 808:808:5f57:7261:7070:6572:2		
12:26...	svchost.exe	1992	UDP Send	c0a8:a6:300:0:3071:9f3a:82e7ffff:57688 -> 808:808:5f57:7261:7070:6572:2		

Figure 2-5: An example Process Monitor capture

ネットワークイベントのみを取得

HTTP 接続をフィルタ

Process Name	Operation	Path	Detail
IEXPLORE.EXE	TCP Connect	onyx:home:3103 -> 2.22.133.163:http	Length: 0, mss: 1412, sackopt: 1, tsopt: 0, wsopt: 1, rcvwin: 66364, rcvwinscale: 2, sndwinscale: 1, seqnum: 0,
IEXPLORE.EXE	TCP Send	onyx:home:3103 -> 2.22.133.163:http	Length: 133, starttime: 65221, endtime: 65221, seqnum: 0, connid: 0
IEXPLORE.EXE	TCP Receive	onyx:home:3103 -> 2.22.133.163:http	Length: 2297, seqnum: 0, connid: 0
IEXPLORE.EXE	TCP Receive	onyx:home:3103 -> 2.22.133.163:http	Length: 0, seqnum: 0, connid: 0
IEXPLORE.EXE	TCP Disconnect	onyx:home:3103 -> 2.22.133.163:http	Length: 0, seqnum: 0, connid: 0

Figure 2-6: A single captured connection

■ トラフィックキャプチャ

10

- パッシブキャプチャ
 - Wireshark
 - システムコールトレース
 - strace
 - Dtrace
 - Windows でのトレース
- アクティブキャプチャ
 - ネットワークプロキシ
 - ポートフォワーディングプロキシ
 - SOCKS プロキシ
 - HTTP プロキシ
 - リバース HTTP プロキシ

パケット, システムコール呼び出しのキャプチャにより
アプリケーションの使用プロトコルを判別

アクティブキャプチャ

11

- ・ キャプチャデバイスはクライアント、サーバー間のブリッジとして機能
 - 中間者攻撃としてトラフィックに影響を与える
- ・ Man-in-the-middle Attack
 - サーバークライアント間に第三者Aが介入
 - 一般的には、アプリケーションにプロキシサービスの経由を強制し実現
- ・ プロキシサーバー
 - クライアントの代わりに通信を担うサーバー
- ・ メリット
 - プロトコル分析や悪用が容易
 - トラフィックの変更
 - 暗号化、圧縮機能の無効化
- ・ デメリット
 - 意図しない影響の発生

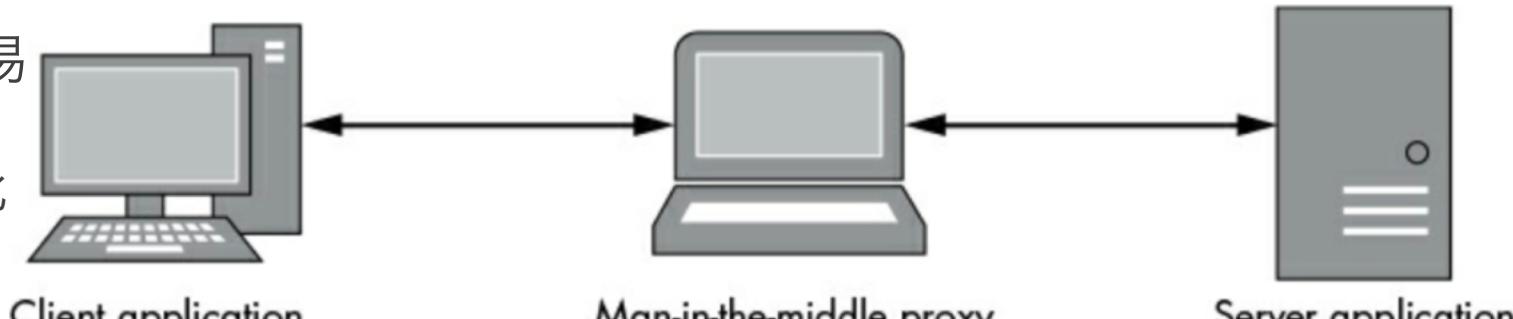


Figure 2-7: A man-in-the-middle proxy

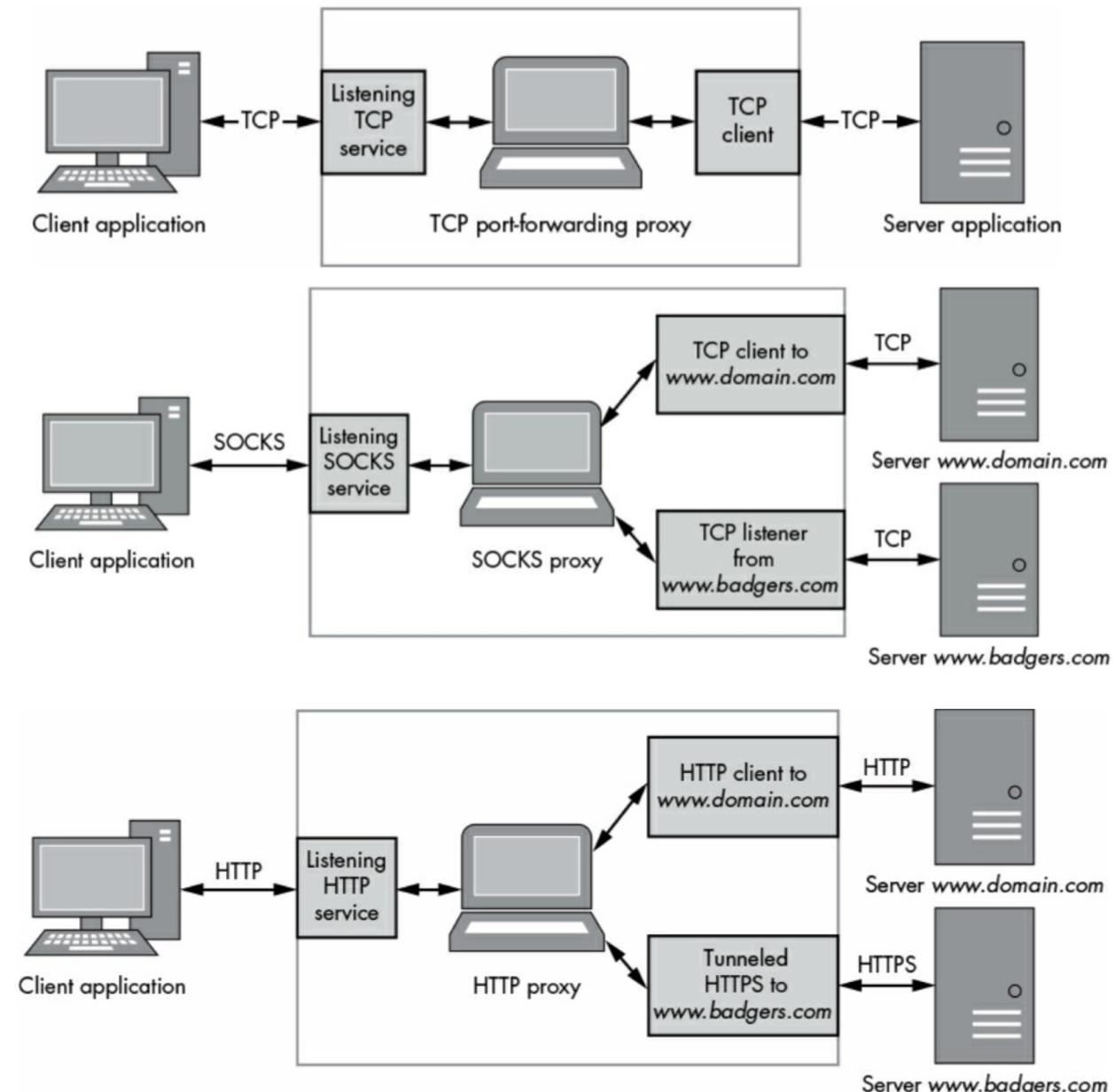
プロキシサーバーを用いてキャプチャ

アクティブキャプチャ | ネットワークプロキシ

12

- ポートフォワーディングプロキシ
 - サーバーは1ポートのみ指定可能
 - 静的設定
- SOCKS プロキシ
 - 1プロキシで複数の接続を処理可能
 - SOCKS メッセージを使用
- HTTP プロキシ
 - アウトバウンドへのプロキシ
 - HTTP リクエストパラメータを利用
- リバース HTTP プロキシ
 - インバウンドへのプロキシ

プロキシサーバーの要件と作成方法は?
プロキシサーバーへのリダイレクト方法は?



- プロキシサーバーの作成
 - Canape Core ライブラリを使用

- リダイレクト方法
 - ウェブブラウザ
 - URL に localhost:port を指定
 - IPアドレスのみ変更可能なアプリ
 - 宛先を localhost に変更
 - ポート番号はアプリを調査し
プロキシサーバーを調整
 - 複数の接続を持つアプリ
 - パッシブキャプチャで使用ポートを調査した後、プロキシインスタンスを複数生成
 - 宛先変更を許可しないアプリ (ただしホスト名指定)
 - カスタム DNS サーバーを用意、ローカルの ホスト名解決テーブルを変更
Unix系: /etc/hosts, Windows: C:\Windows\System32\Drivers\etc\hosts
 - * ただし、これはマルウェアの挙動に近いためウィルス対策ソフトに検知されるかも

PortFormat
Proxy.csx

```
// PortFormatProxy.csx - Simple TCP port-forwarding proxy
// Expose methods like WriteLine and WritePackets
using static System.Console;
using static CANAPE.Cli.ConsoleUtils;

// Create proxy template
var template = new ❶FixedProxyTemplate();
template.LocalPort = ❷LOCALPORT;
template.Host = ❸"REMOTEHOST";
template.Port = ❹REMOTEPORT;

// Create proxy instance and start
❺ var service = template.Create();
service.Start();

WriteLine("Created {0}", service);
WriteLine("Press Enter to exit...");
ReadLine();
❻ service.Stop();

// Dump packets
var packets = service.Packets;
WriteLine("Captured {0} packets:", packets.Count);
❼ { WritePackets(packets);
```

プロキシサーバーのポート指定
リモートサーバーのホスト指定
リモートサーバーのポート指定

プロキシサーバー作成

プロキシサーバー終了

キャプチャ結果を表示

Listing 2-4: A simple TCP port-forwarding proxy example

プロキシインスタンスを複数生成

ホスト名解決テーブルを変更

Unix系: /etc/hosts, Windows: C:\Windows\System32\Drivers\etc\hosts

* ただし、これはマルウェアの挙動に近いためウィルス対策ソフトに検知されるかも

- プロキシサーバーの作成
 - Canape Core ライブラリを使用
- リダイレクト方法
 - ウェブブラウザ
 - URL に localhost:port を指定
 - IPアドレスのみ変更可能なアプリ
 - 宛先を localhost に変更
 - ポート番号はアプリを調査しプロキシサーバーを調整
 - 複数の接続を持つアプリ
 - パッシブキャプチャで使用ポートを調査した後、プロキシインスタンスを複数生成
 - 宛先変更を許可しないアプリ (ただしホスト名指定)
 - カスタム DNS サーバーを用意、ローカルのホスト名解決テーブルを変更
Unix系: /etc/hosts, Windows: C:\Windows\System32\Drivers\etc\hosts
 - * ただし、これはマルウェアの挙動に近いためウィルス対策ソフトに検知されるかも

PortFormat
Proxy.csx

```
// PortFormatProxy.csx - Simple TCP port-forwarding proxy
// Expose methods like WriteLine and WritePackets
using static System.Console;
using static CANAPE.Cli.ConsoleUtils;

// Create proxy template
var template = new ❶FixedProxyTemplate();
template.LocalPort = ❷LOCALPORT;
template.Host = ❸"REMOTEHOST";
template.Port = ❹REMOTEPORT;

// Create proxy instance and start
❺ var service = template.Create();
service.Start();

WriteLine("Created {0}", service);
WriteLine("Press Enter to exit...");
ReadLine();
❻ service.Stop();

// Dump packets
var packets = service.Packets;
WriteLine("Captured {0} packets:",
packets.Count);
❼ { WritePackets(packets);
```

プロキシサーバーのポート指定
リモートサーバーのホスト指定
リモートサーバーのポート指定

プロキシサーバー作成

プロキシサーバー終了

キャプチャ結果を表示

Listing 2-4: A simple TCP port-forwarding proxy example

Listing 2-4: A simple TCP port-forwarding proxy example

Listing 2-4: A simple TCP port-forwarding proxy example

アクティブキャプチャ | SOCKS Proxy

15

- プロキシサーバーの仕組み
 - アプリは **SOCKS リクエスト** をプロキシサーバーに送信
 - 宛先IP/Port/Username を含む
 - プロキシは宛先サーバーに接続要求
 - プロキシはクライアントに **SOCKS レスポンス** を送信
 - IP/Port はバインディング要求時のみ利用 (プロキシサーバーでリッスンが可能に)
- プロキシサーバーの作成
 - Canape Core ライブラリを使用
 - リモートサーバーの指定が不要
- リダイレクト方法
 - 次スライド

Size in octets	1	1	2	4	VARIABLE	1
VER 0x04	CMD 0x01	TCP PORT 12345	IP ADDRESS 0x10000001	USERNAME "james"	NULL 0x00	
VER 0x04	RESP 0x5A	TCP PORT 0	IP ADDRESS 0			

Size in octets 1 1 2 4

IP/Port はバインディング要求時のみ利用 (プロキシサーバーでリッスンが可能に)

SocksProxy.csx

```
// SocksProxy.csx - Simple SOCKS proxy
// Expose methods like WriteLine and WritePackets
using static System.Console;
using static CANAPE.Cli.ConsoleUtils;

// Create the SOCKS proxy template
❶ var template = new SocksProxyTemplate();
template.LocalPort = ❷ LOCALPORT;

// Create proxy instance and start
var service = template.Create();
service.Start();

WriteLine("Created {0}", service);
WriteLine("Press Enter to exit...");
ReadLine();
service.Stop();

// Dump packets
var packets = service.Packets;
WriteLine("Captured {0} packets:",
packets.Count);
WritePackets(packets);
```

プロキシサーバーのポート指定

プロキシサーバー作成

プロキシサーバー終了

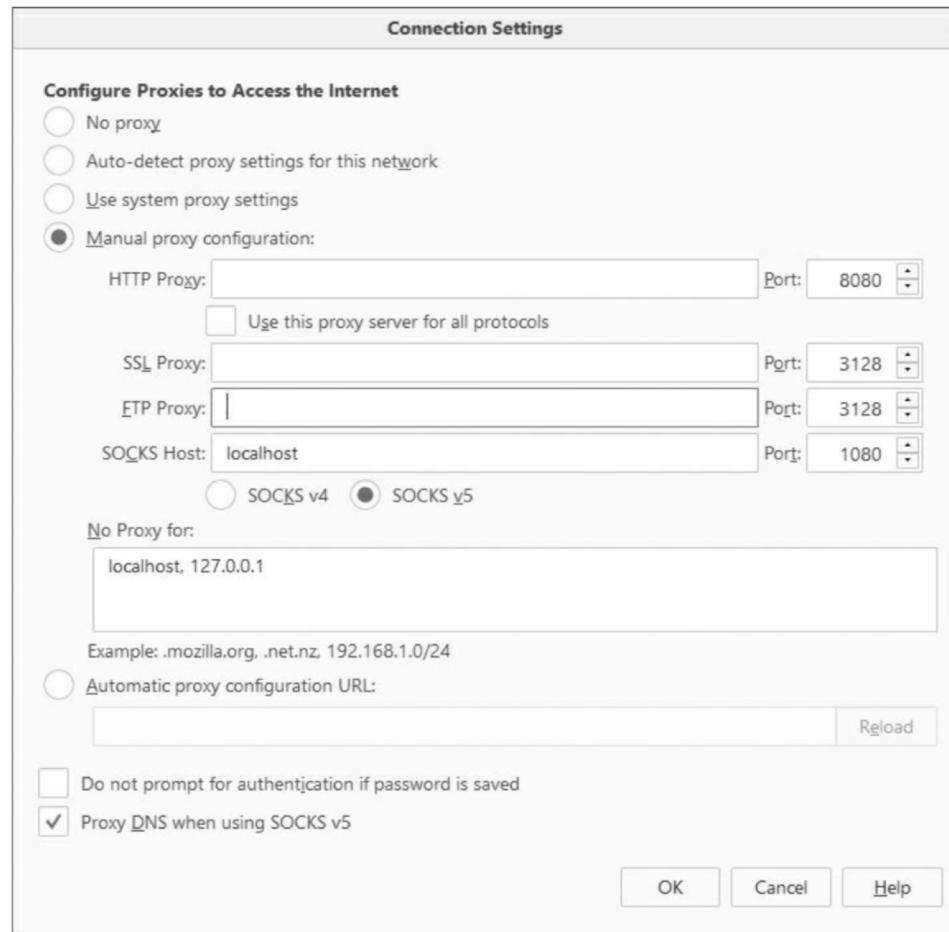
キャプチャ結果を表示

Listing 2-6: A simple SOCKS proxy example

アクティブキャプチャ | SOCKS Proxy

16

- リダイレクト方法
 - ブラウザなら直接設定可能



- プロキシ設定がない場合
 - Javaアプリケーション
 - JavaランタイムにはSOCKSサポートを有効にするコマンドラインパラメータが存在
 - \$ java -DsocksProxyHost=localhost \ -DsocksProxyPort=1080 TCPClient
 - OSのデフォルトプロキシを設定
 - システム全体の通信をSOCKSプロキシに通す
- SOCK version
 - 4: IPv4のみサポート, ホスト名は使用不可
 - 4a: ホスト名による接続が許可
 - 5: IPv6, UDP通信, 認証メカニズムの改善が導入

Figure 2-12: Firefox proxy configuration

アクティブキャプチャ | HTTP Proxy

17

- ・プロキシサーバーの仕組み

- アプリは絶対 URIを指定しプロキシにリクエスト送信
- プロキシは URI のホスト部分を用いてサーバーにリクエスト送信

GET /image.jpg HTTP/1.1 → GET <http://www.domain.com/image.jpg> HTTP/1.1

- ・HTTPS の場合ペイロードは暗号化 (URIが参照できない)

- HTTP1.1以降導入された HTTP CONNECT メッセージを利用

CONNECT www.domain.com:443 HTTP/1.1 HTTP/1.1 200 Connection Established



アクティブキャプチャ | HTTP Proxy

18

- ・プロキシサーバーの作成
 - Canape ライブラリを利用
 - プロキシサーバーのローカルポートのみ指定
- ・リダイレクト方法
 - HTTPを用いるアプリケーションはプロキシ設定を持つ
 - ブラウザ, OS 以外にコマンドラインツールも環境変数による設定が可能
 - e.g., curl, wget, apt

HttpProxy.csx

```
// HttpProxy.csx - Simple HTTP proxy
// Expose methods like WriteLine and WritePackets
using static System.Console;
using static CANAPE.Cli.ConsoleUtils;

// Create proxy template
❶ var template = new HttpProxyTemplate();
template.LocalPort = ❷ LOCALPORT;

// Create proxy instance and start
var service = template.Create();
service.Start();

WriteLine("Created {0}", service);
WriteLine("Press Enter to exit...");
ReadLine();
service.Stop();

// Dump packets
var packets = service.Packets;
WriteLine("Captured {0} packets:", packets.Count);
WritePackets(packets);
```

Listing 2-8: A simple forward HTTP proxy example

- ・完全なHTTPパーサーを実装する必要があるためプロキシの作成は複雑
- ・HTTP プロキシはアウトバウンドトラフィックに対するプロキシ
 - 代わりにサーバーに問い合わせてくれる

- ・特定のサーバへの要求を経由するように設定されたプロキシ
 - インバウンドトラフィックに対するプロキシ
 - ロードバランシング, セキュリティ上の理由 (サーバーを直接公開しない) で利用
 - ・HTTP ヘッダのホスト名を利用
-

```
GET /image.jpg HTTP/1.1
User-Agent: Super Funky HTTP Client v1.0
Host: ❶www.domain.com
Accept: */*
```

Listing 2-9: An example HTTP request

- ・リダイレクト方法
 - クライアントの hosts ファイルの変更
 - DNS サーバーを構築
 - ホスト名から引かれる IP アドレスをプロキシに設定
 - Canape を用いて DNS サーバーを構築

アクティブキャプチャ | Reverse HTTP Proxy

20

- DNS サーバーの作成
 - すべてのリクエストに対して設定したIPアドレスに偽装

DnsServer.csx

```
// DnsServer.csx - Simple DNS Server
// Expose console methods like WriteLine at global level.
using static System.Console;

// Create the DNS server template
var template = new DnsServerTemplate();

// Setup the response addresses
template.ResponseAddress = ①"IPV4ADDRESS";
template.ResponseAddress6 = ②"IPV6ADDRESS";
template.ReverseDns = ③"REVERSEDNS";

// Create DNS server instance and start
var service = template.Create();
service.Start();
WriteLine("Created {0}", service);
WriteLine("Press Enter to exit...");
ReadLine();
service.Stop();
```

- プロキシサーバーの作成

ReverseHttp
Proxy.csx

```
// ReverseHttpProxy.csx - Simple reverse HTTP proxy
// Expose methods like WriteLine and WritePackets
using static System.Console;
using static CANAPE.Cli.ConsoleUtils;

// Create proxy template
var template = new HttpReverseProxyTemplate();
template.LocalPort = ①LOCALPORT;

// Create proxy instance and start
var service = template.Create();
service.Start();

WriteLine("Created {0}", service);
WriteLine("Press Enter to exit...");
ReadLine();
service.Stop();

// Dump packets
var packets = service.Packets;
WriteLine("Captured {0} packets:", packets.Count);
WritePackets(packets);
```

■ トラフィックキャプチャ

21

- パッシブキャプチャ
 - Wireshark
 - システムコールトレース
 - strace
 - Dtrace
 - Windows でのトレース
- アクティブキャプチャ
 - ネットワークプロキシ
 - ポートフォワーディングプロキシ
 - SOCKS プロキシ
 - HTTP プロキシ
 - リバース HTTP プロキシ

パケット, システムコール呼び出しのキャプチャにより
アプリケーションの使用プロトコルを判別

プロキシを通すことでパケットをキャプチャ
パケットの偽装や復号など割となんでもできる
ただし, アプリケーションにあったプロキシの構築と
リダイレクト方法の設定が必要