

MTD演習（全体の構成 480分（8時間））

1. 事前準備	60分
2. 講義・説明	60分
3. 演習1: カーネルレベルMTD	120分
4. 演習2: ネットワークレベルMTD	120分
5. まとめ, レポート作成	120分

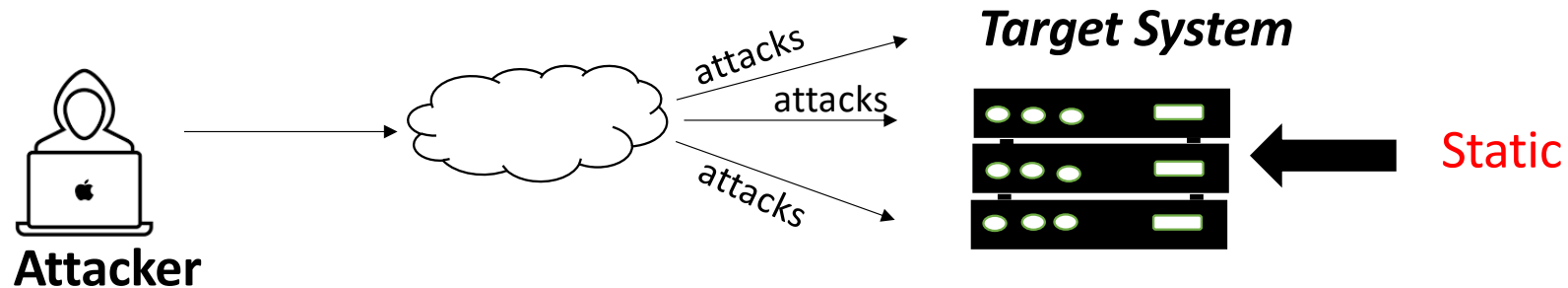
MTD演習（事前準備 60分）

Moving Targer Defense (MTD) の演習については以下のスキル、基礎知識が必要となります。各自で事前学習をお願いします。

- UNIX (Linux) の基本コマンド
- OSの基礎知識, システムコールの基礎知識
- リモートログイン, sshクライアントの利用
- emacs, vi, nano 等のいずれかのエディタの利用
- Python3によるプログラミングの基礎知識
- Webアプリケーションの基礎知識
- VirutalBox, Ubuntu Serverのインストール, 基本操作

MTD
(Moving Target Defense)

サイバー空間の状況 Situation in Cyberspace



攻撃者に有利な状況

- 短時間に攻撃を実行可能
- ひとつ攻撃可能な脆弱性を見つければ良い
- 偵察と準備に多くの時間を掛けることが可能
- 継続的, 適用的, 目的主導的 (最小攻撃コストで最大利益)

いままでの防御技術では不十分

Attackers have advantages.

- can execute attack in a **short time**
- need to find only a **single vulnerable** entry point
- **unlimited time** for reconnaissance / preparation
- They are **persistent, adaptive and incentive-driven** (minimum attack cost with maximum dangerous outcome)

Traditional defense techniques are not sufficient

解決したい課題

Problem statements

- どうすれば攻撃者側の負荷と攻撃に要する時間, コストを増やすことができるか
- どうすれば攻撃成功確率を減らすことができるか
- どのように新しい防御技術追加することによりシステムの防御能力を拡張することができるか
- 複数の多様化を行う方法が最小限の防御コストでセキュアな防御システムとなりうるかどうか
- How can we increase attackers' workload and attack time, costs?
- How can we reduce the *probability of attack success* over time?
- How can we enhance a system's resiliency with additional defense techniques?
- Whether a hybrid diversification approaches can introduce a secure defense system with minimum defense cost or not?

Moving Target Defense Approach

MTD (Moving Target Defense)の基本的な考え方は、複数のシステムの変更を制御することにより、保護したいシステムの情報に関する不確実性を高めて攻撃側を複雑にすること

これにより攻撃者の調査と攻撃にかかるコストを増やし攻撃の機会を減らすことができる

The concept of controlling the alteration of *multiple systems* with the aim to *increase uncertainty* about a protected system's information and give *complexity* for attackers.

By doing so, we can reduce the attack window of opportunity and increase the costs of attackers' probing and attack efforts.



停まっている標的を射撃
Shoot a stationary target



動いている標的を射撃
Shoot a moving target

Goals of This Technique

- 攻撃の**不確実性**を高める
 - 攻撃に必要な**努力とコスト**を増加させる
 - 保護したいシステムは**探査しにくく予測しにくい標的**となる
 - 攻撃成功確率を**低減**する
- To increase the **uncertainty** for the attacker
 - To increase **attacker's effort and cost** in making attacks
 - A protected system will be **hard to be exploited and unpredictable** destination
 - To reduce the **probability of attack success** over time

MTD技術によりランダム性が加わるため、特定の瞬間にシステムがどのような構成と**なっているかわからない**。このランダム性のため攻撃者がゼロデイ攻撃などを成功させるためのコストが増加する可能性がある

MTDフレームワークを設計するときの3つの考慮すべき事項：**(what, when, how)** が変更可能なパラメータ

MTD techniques will have randomness built, so this randomness can increase the cost for an attacker to succeed in using e.g, **zero-day attacks** because it does not necessarily know which configuration of the system is in place at any particular moment.

Three consideration when to design MTD framework: **(what, when, how) to move the elements**

What to Move ?

攻撃者を混乱させるために動的に変更できるシステム構成, 属性, 構成要素 (すなわち attack surface のこと)

例)

- ❖ 命令セット
- ❖ アドレス空間のレイアウト
- ❖ IPアドレス, ポート番号
- ❖ プロキシ
- ❖ 仮想マシン
- ❖ OS
- ❖ ミドルウェア
- ❖ フレームワーク
- ❖ ソフトウェア

what system configuration attribute or elements (components) (i.e., attack surface) can be dynamically changed to confuse attackers.

For example,

- ❖ Instruction sets
- ❖ address space layouts
- ❖ IP addresses, port numbers
- ❖ proxies, virtual machines
- ❖ operating systems
- ❖ software programs

When to Move ?

MTDシステムのある状態から別の状態に変更するのに適切なタイミングを決定し、攻撃者が取得した現在の状態、または攻撃の状況を無効化

- ❖ 反応的な適応
- ❖ 積極的な適応
- ❖ ハイブリッド適応

deciding the optimal time to change from the current state of an MTD system to a new state, invalidating information or progress gained by an attacker in the current state.

- ❖ Reactive adaptation
- ❖ Proactive adaptation
- ❖ Hybrid adaptation

How to Move ?

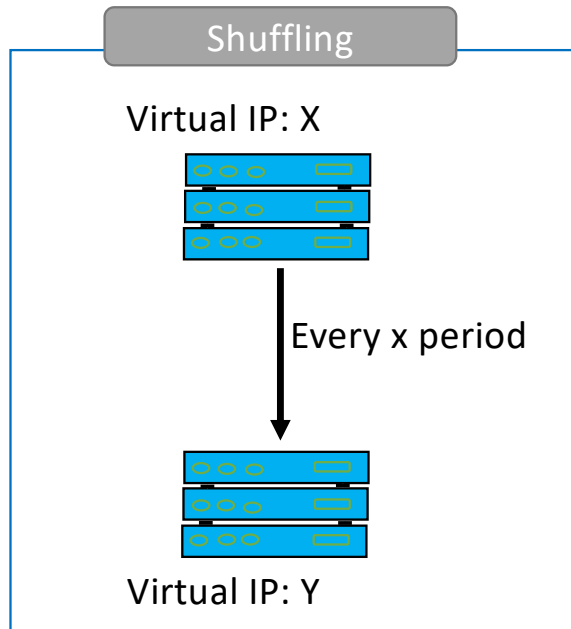
- どのようにシステムの属性や構成要素（すなわち標的）を変化させて、攻撃者側の予測不可能性や不確実性を増加させ、攻撃者を混乱させるか

- ❖ シャッフリング,
- ❖ 多様性,
- ❖ 冗長性

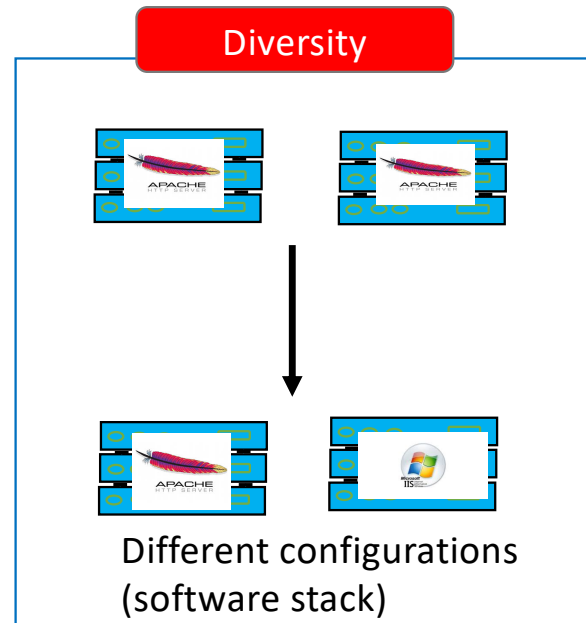
- *how to change the moving attributes or elements (components) (i.e., targets) to increase unpredictability and/or uncertainty, leading to an attacker's high confusion.*

- ❖ shuffling,
- ❖ diversity,
- ❖ Redundancy

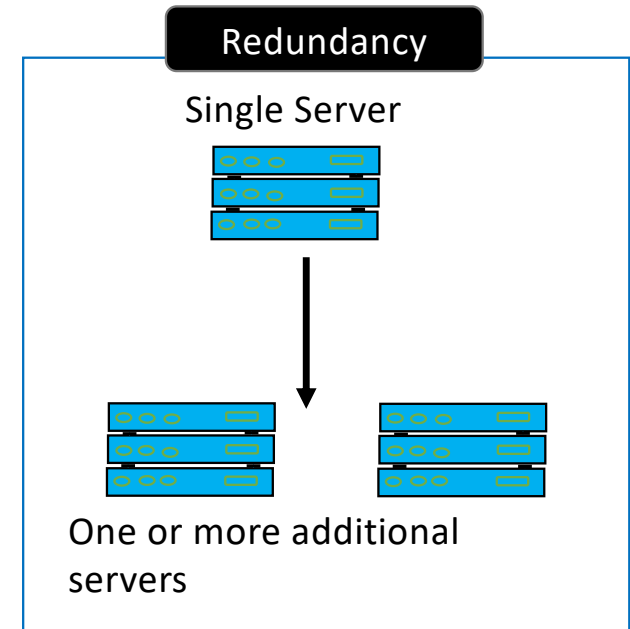
Examples of MTD Techniques



Performance and Efficiency
正規の利用者に対する
Availabilityの確保



Resilience and Robustness



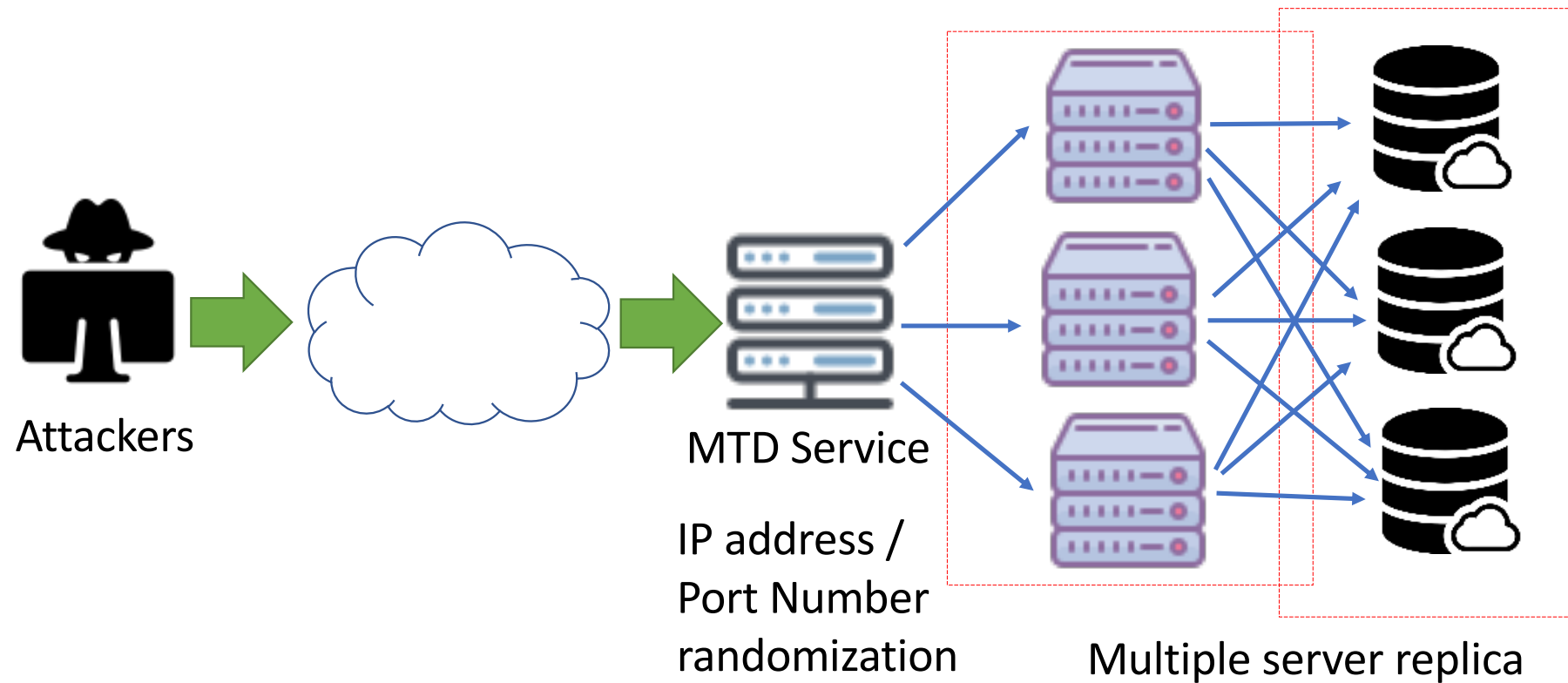
Reliability and Availability

異なる階層におけるMTD技術の変更要素

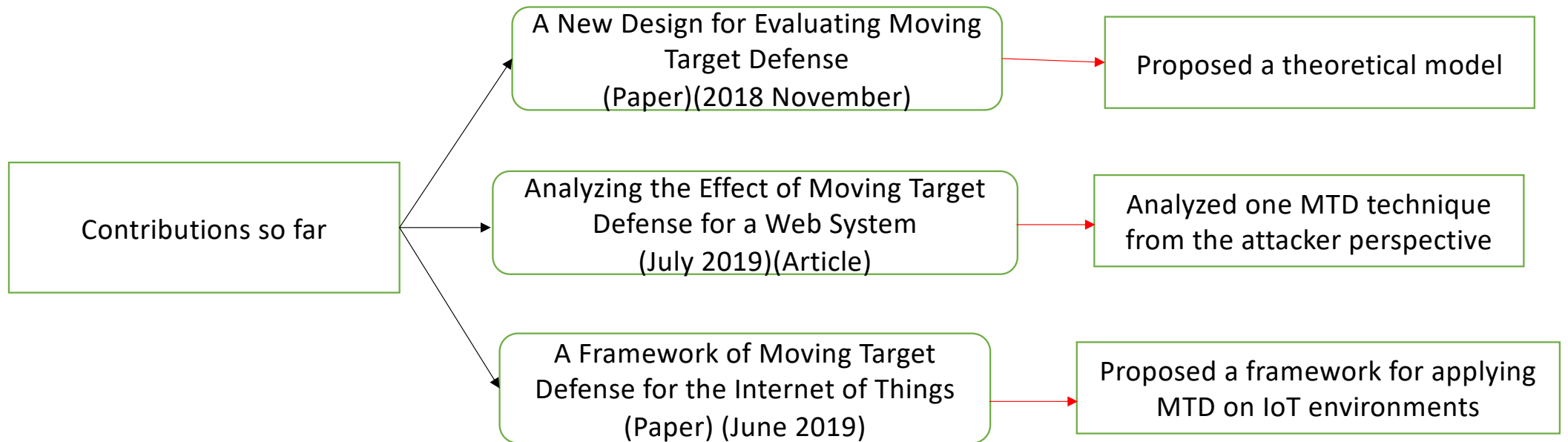
Elements of MTD Techniques at Different Layers

Layers	MTD Techniques		
	Shuffling	Diversity	Redundancy
Application	TCP/UDP port numbers	Web: Apache, IIS, etc. App: Java, PHP, etc. DB: SQL, MySQL, Oracle, etc. Others: mail-server, proxy-server, etc.	Web service replica Application replica Database replica Other service replica
OS-host	IP address, System Call Numbering	Windows server various versions Linux various versions Other Unix various versions	Host OS and VM replica
VM-instance	Virtual IP address		
Virtual Machine Manager	Failover, switchover	VMware, ESXi Other kernel-based VM, Vbox, etc.	Hypervisor's replica

Example of Network-level and Application-level MTD Framework



Overview of Wai Kyi's research



MTD演習1：

カーネルレベルMTD

(システムコール・シャッフリング)

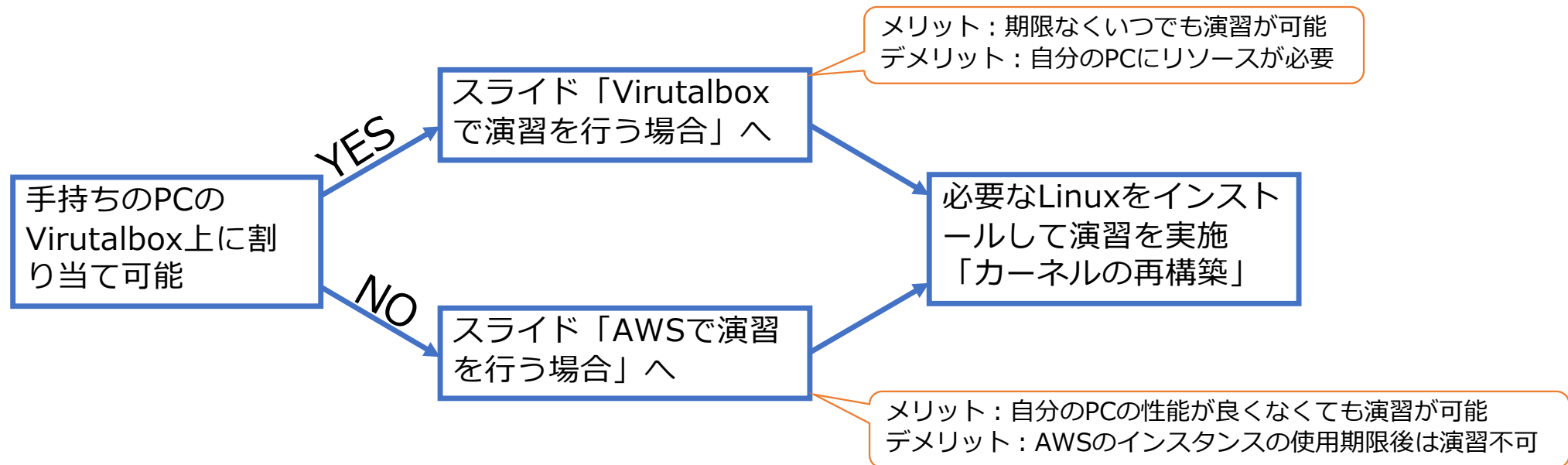
演習の準備

演習用のLinuxの準備

- リソース : CPU2コア以上, メモリ4GB以上, 二次記憶32GB以上
- OS : Ubuntu Server 16.04 LTS x86 64 bit

以下のどちらかの準備を事前に行っておいてください

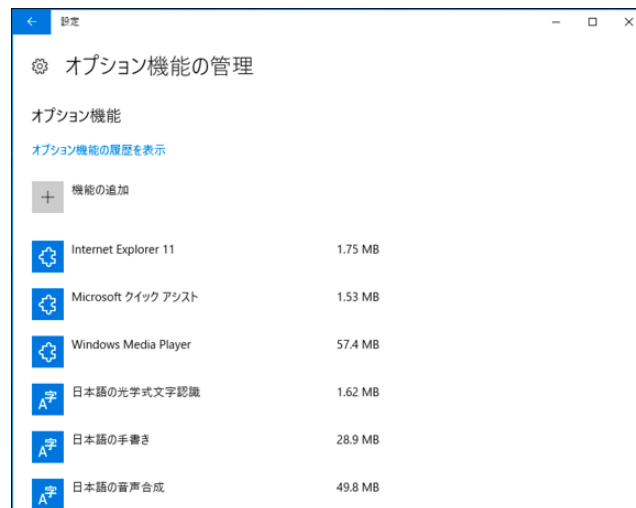
(今回の演習では「AWSで演習を行う場合」で実施します.)



sshクライアントの準備

SSHクライアント

- Windows10, MacOSでは標準
- Windows 10ではいってないとき
 - 「スタートメニュー」→「オプション機能の管理」
 - →「オプション機能の追加」



PCからLinuxにsshによる接続

❖ MacからLinuxにssh接続

- 純正のsshクライアントを使用（お勧め！）

❖ Windows10からLinuxにssh接続（以下の選択肢からひとつ）

- 純正のsshクライアントを使用（お勧め！）
- TeraTerm等を使用
(<http://ecs.kyushu-u.ac.jp/VDI/4-b-Win-TeraTerm+2Linux.pdf>)
- WSL（Windows Ubuntu）のsshを使用

Windows10 純正sshクライアント

C:\¥ Windows ¥ System32 ¥ OpenSSH の下にファイルが置かれている

```
ファイルが見つかりません
C:\¥Users¥koide>dir ¥Windows¥System32¥OpenSSH
ドライブ C のボリューム ラベルがありません。
ボリューム シリアル番号は AC7A-06AB です

C:\¥Windows¥System32¥OpenSSH のディレクトリ

2018/08/18  11:24    <DIR>          .
2018/08/18  11:24    <DIR>          ..
2018/08/18  11:24    <DIR>          Logs
2017/09/28  15:49             322,048 scp.exe
2017/09/28  15:49             390,656 sftp.exe
2017/09/28  15:49             521,728 ssh-add.exe
2017/09/28  15:49             612,352 ssh-agent.exe
2017/09/28  15:49             599,552 ssh-keygen.exe
2017/09/28  15:49             851,456 ssh.exe
2017/09/28  15:49              3,520 sshd_config
              7 個のファイル             3,301,312 バイト
              3 個のディレクトリ 14,814,961,664 バイトの空き領域

C:\¥Users¥koide>
```

AWSで演習を行う場合

(今回の演習ではこの方法で実施)

カーネルレベルのMTD演習（準備）

インスタンス作成

Amazon EC2でインスタンスを作成

- AMI: Ubuntu Server 16.04 LTS (HVM), SSD Volume Type 64 ビット (x86)
- t2.medium 以上
- ストレージ追加：32GiB 以上

aws サービス リソースグループ

1. AMI の選択 2. インスタンスタイプの選択 3. インスタンスの設定 4. ストレージの追加 5. タグの追加 6. セキュリティグループの設定 7. 確認

ステップ 7: インスタンス作成の確認
インスタンスの作成に関する詳細を確認してください。各セクションの変更に戻ることができます。[作成] をクリックして、インスタンスにキーペアを割り当て、作成処理を完了します。

⚠️ お客様のインスタンス設定は無料利用枠の対象ではありません
無料利用枠の対象であるインスタンスを起動するには、選択している AMI、インスタンスタイプ、設定オプション、ストレージデバイスをチェックします。無料利用枠の利用枠と使用制限に関する詳細情報をご覧ください。

AMI の詳細 [AMI の編集](#)

Ubuntu Server 16.04 LTS (HVM), SSD Volume Type - ami-078648cce0d33c256
Ubuntu Server 16.04 LTS (HVM), EBS General Purpose (SSD) Volume Type. Support available from Canonical (<http://www.ubuntu.com/cloud/services>).
ルートデバイスタイプ: ebs 仮想化タイプ: hvm

インスタンスタイプ [インスタンスタイプの編集](#)

インスタンスタイプ	ECU	vCPU	メモリ (GiB)	インスタンス ストレージ (GB)	EBS 最適化利用	ネットワークパフォーマンス
t2.medium	可変	2	4	EBS のみ	-	Low to Moderate

セキュリティグループ [セキュリティグループの編集](#)

セキュリティグループ名 launch-wizard-25
説明 launch-wizard-25 created 2020-07-05T13:23:15.676+09:00

タイプ ①	プロトコル ①	ポート範囲 ①	ソース ①	説明 ①
-------	---------	---------	-------	------

このセキュリティグループにはルールがありません。

[キャンセル](#) [戻る](#) [起動](#)

より高性能なAWSのEC2インスタンスを演習用に配布します。必要な受講生はア
ナウンスの指示に従いログインに必要な情報を得て下さい。

→スライド「カーネルの再構築」までスキップしてください

今回の演習では. .

- SECKUN事務局からAWSのインスタンスを配布します.
- SECKUN事務局からAWSのインスタンスのIPアドレスとsshの秘密鍵ファイルを送付します.
- ダウンロードした秘密鍵のファイルのパーミッションを変更してください

% chmod 600 秘密鍵ファイル MacOSの場合

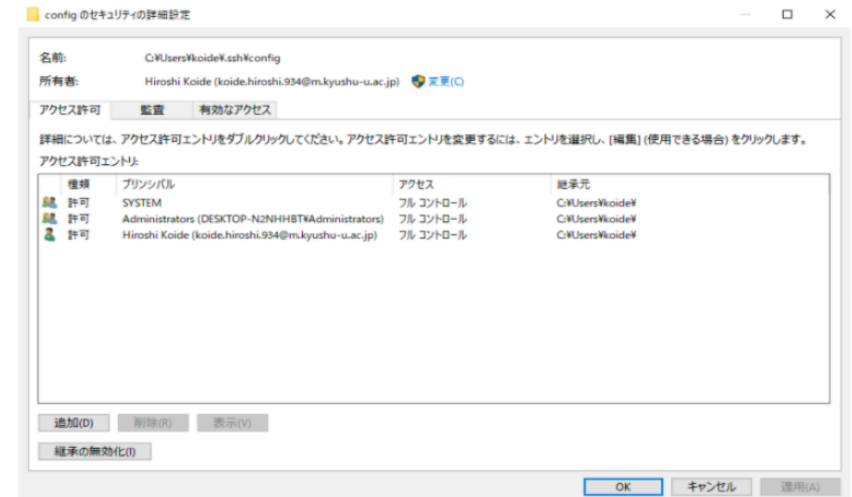
Windows の場合, 秘密鍵ファイルのプロパティを右図のように変更してください.

(参考 : <https://qastack.jp/superuser/1296024/windows-ssh-permissions-for-private-key-are-too-open>)

- ssh クライアントを使い以下のコマンドでAWSのインスタンスに接続してください.

% ssh ubuntu@IPアドレス -i 秘密鍵ファイル

→スライド「カーネルの再構築」に進んでください.



VirtualBoxで演習を行う場合

VirtualBoxにLinuxをインストール

- <https://www.virtualbox.org>

VirtualBox

search...
Login Preferences

Download VirtualBox

Here you will find links to VirtualBox binaries and its source code.

VirtualBox binaries

By downloading, you agree to the terms and conditions of the respective license.

If you're looking for the latest VirtualBox 6.0 packages, see [VirtualBox 6.0 builds](#). Please also use version 6.0 if you need to run VMs with software virtualization, as this has been discontinued in 6.1. Version 6.0 will remain supported until July 2020.

If you're looking for the latest VirtualBox 5.2 packages, see [VirtualBox 5.2 builds](#). Please also use version 5.2 if you still need support for 32-bit hosts, as this has been discontinued in 6.0. Version 5.2 will remain supported until July 2020.

VirtualBox 6.1.12 platform packages

- ⇒ Windows hosts
- ⇒ OS X hosts
- ⇒ Linux distributions
- ⇒ Solaris hosts

The binaries are released under the terms of the GPL version 2.

See the [changelog](#) for what has changed.

You might want to compare the checksums to verify the integrity of downloaded packages. *The SHA256 checksums should be favored as the MD5 algorithm must be treated as insecure!*

- SHA256 checksums, MD5 checksums

Note: After upgrading VirtualBox it is recommended to upgrade the guest additions as well.

VirtualBox 6.1.12 Oracle VM VirtualBox Extension Pack

- ⇒ All supported platforms

Support for USB 2.0 and USB 3.0 devices, VirtualBox RDP, disk encryption, NVMe and PXE boot for Intel cards. See [this chapter from the User Manual](#) for an introduction to this Extension Pack. The Extension Pack binaries are released under the [VirtualBox Personal Use and Evaluation License \(PUEL\)](#). Please install the same version extension pack as your installed version of VirtualBox.

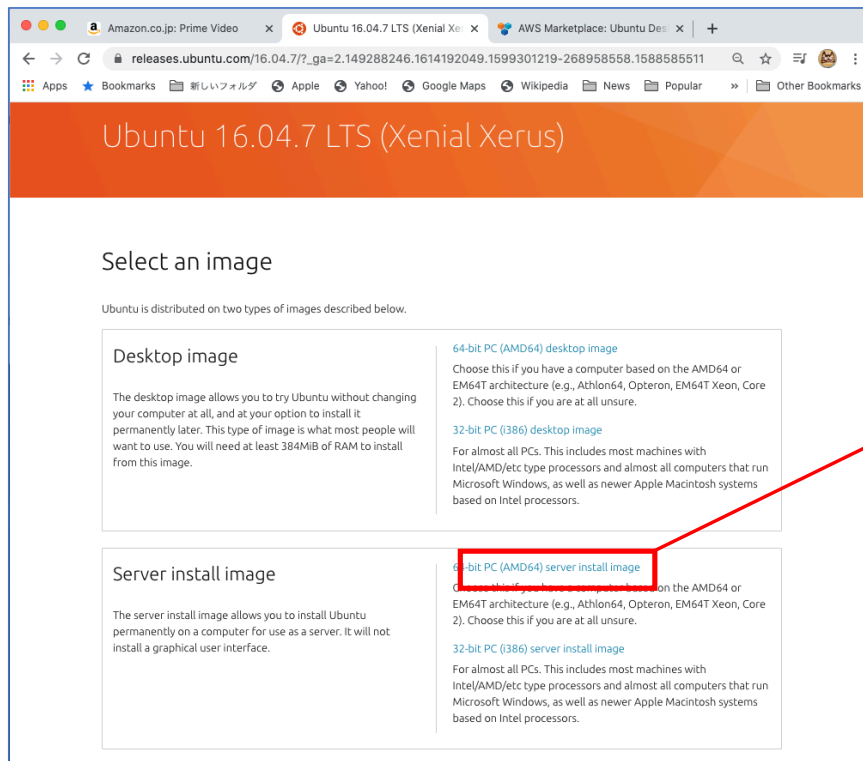
自分のPCのパッケージを選ぶ

拡張パックもインストールしておく

ダウンロードしたパッケージをインストールし、拡張パックもインストール

VirtualBoxにLinuxをインストール

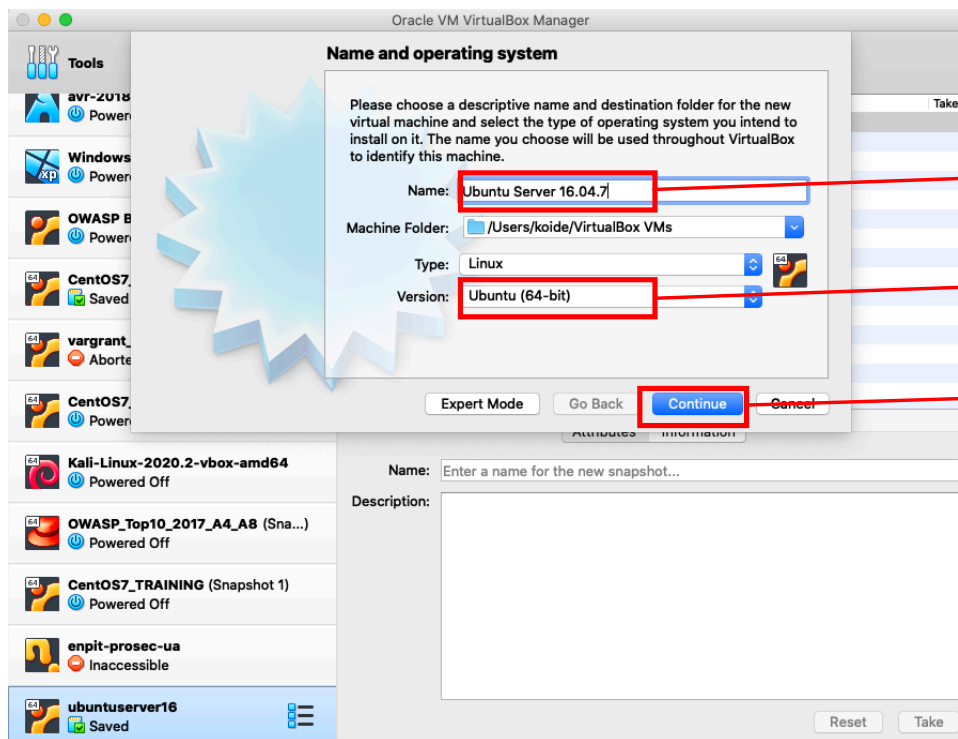
- https://releases.ubuntu.com/16.04.7/?_ga=2.149288246.1614192049.1599301219-268958558.1588585511



64-bit PC (AMD64) server install image
をダウンロードしておく

VirtualBoxにLinuxをインストール

「Machine」 → 「New」

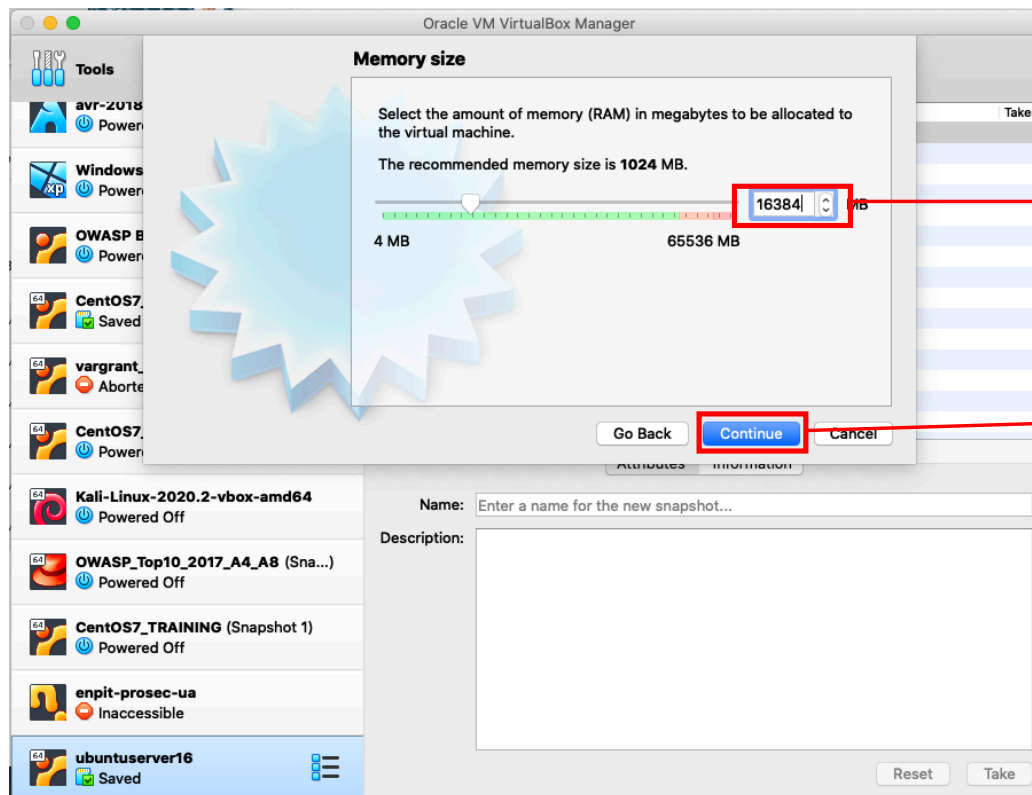


分かり易い名前を付けておく

Ubuntu (64-bit)を選択

最後にContinueを押す

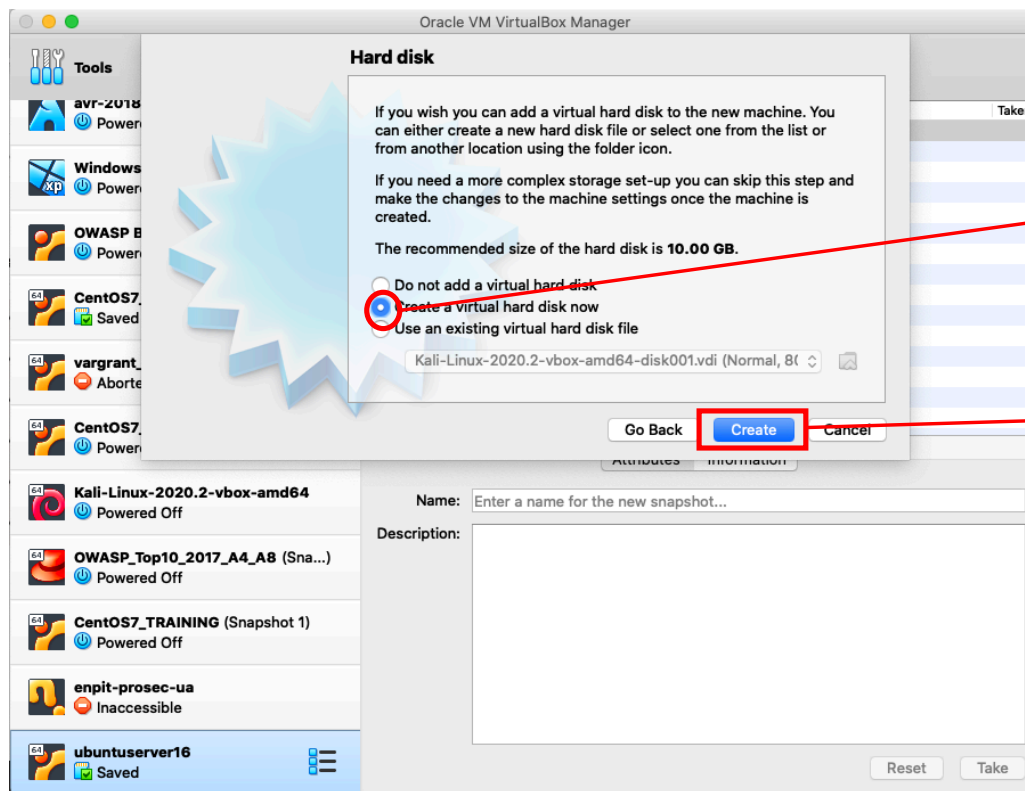
VirtualBoxにLinuxをインストール



割り当てるメモリサイズを入力
ここでは16GBを指定
(8GB以上を割り当ててください)

最後にContinueを押す

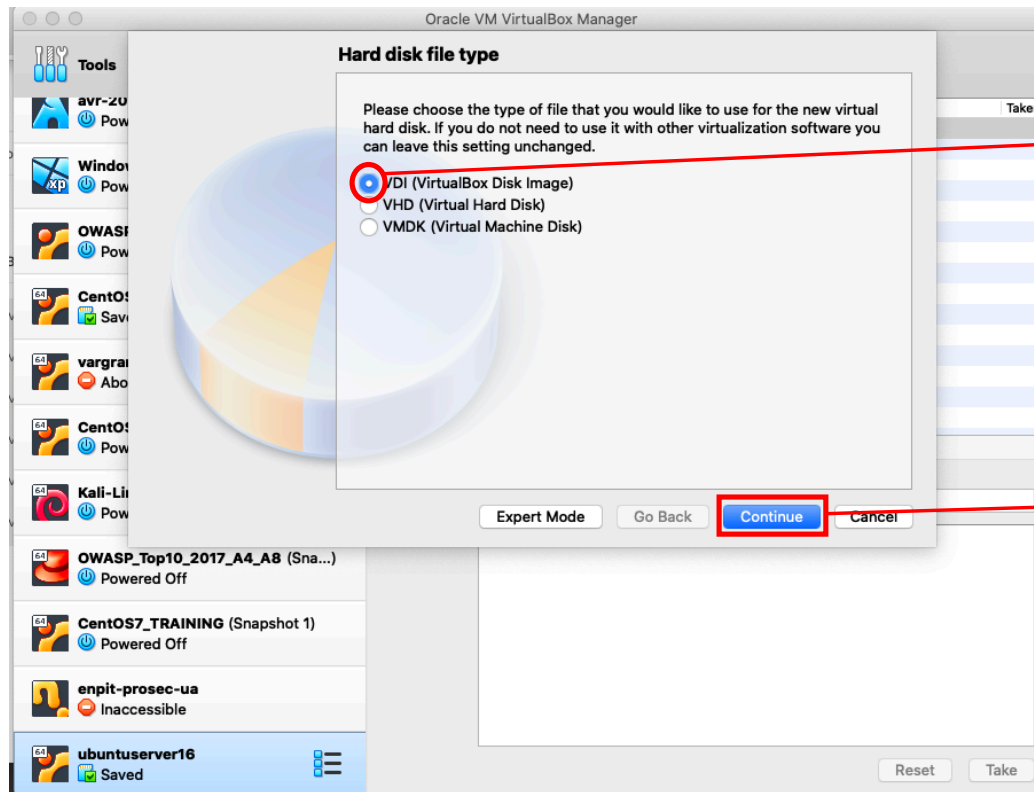
VirtualBoxにLinuxをインストール



Create a virtual hard disk now を選択

最後にContinueを押す

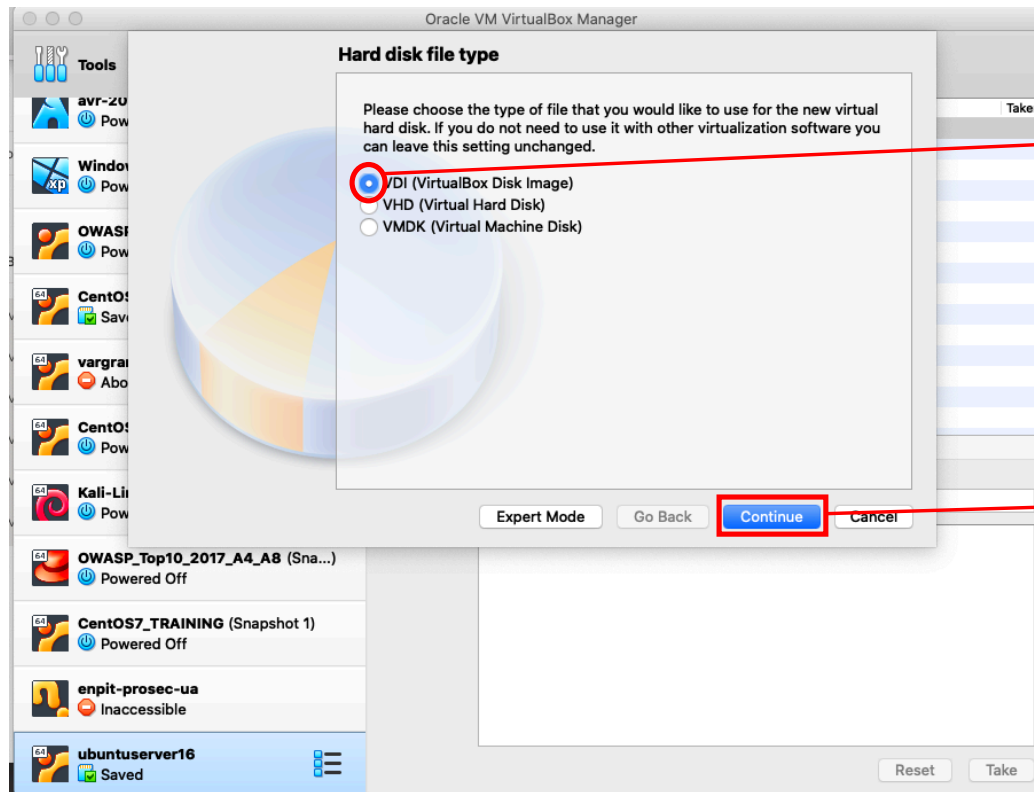
VirtualBoxにLinuxをインストール



VDI (VirtualBox Disk Image) を選択

最後にContinueを押す

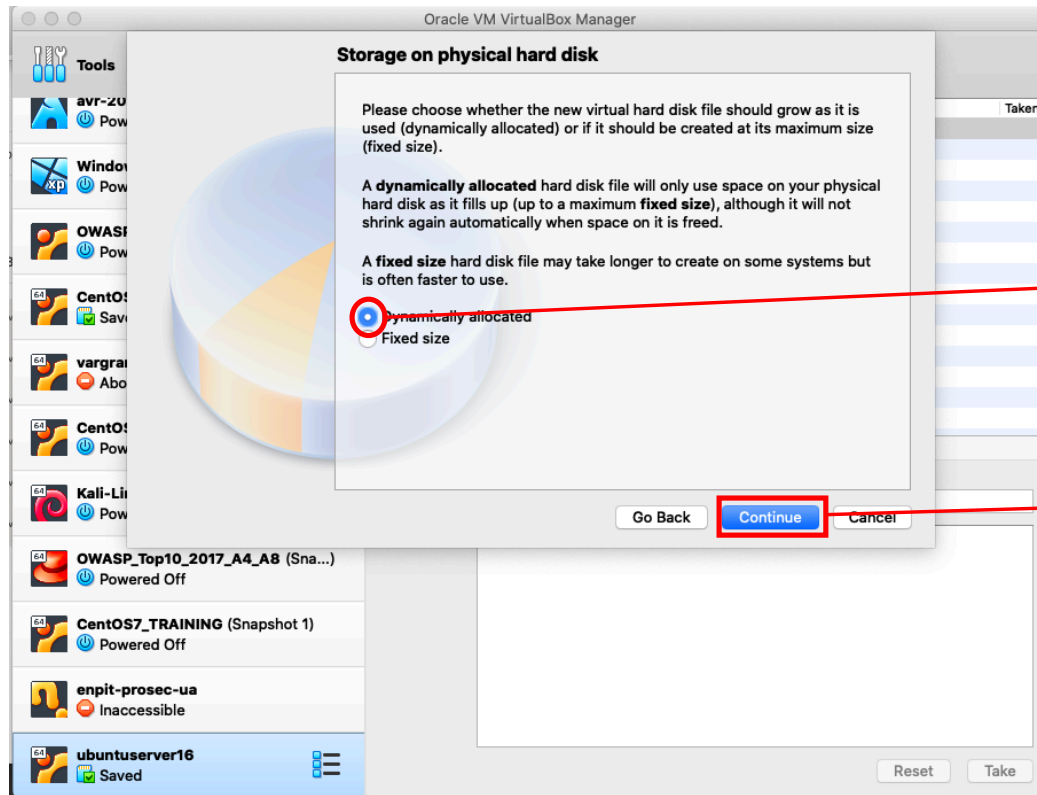
VirtualBoxにLinuxをインストール



VDI (VirtualBox Disk Image) を選択

最後にContinueを押す

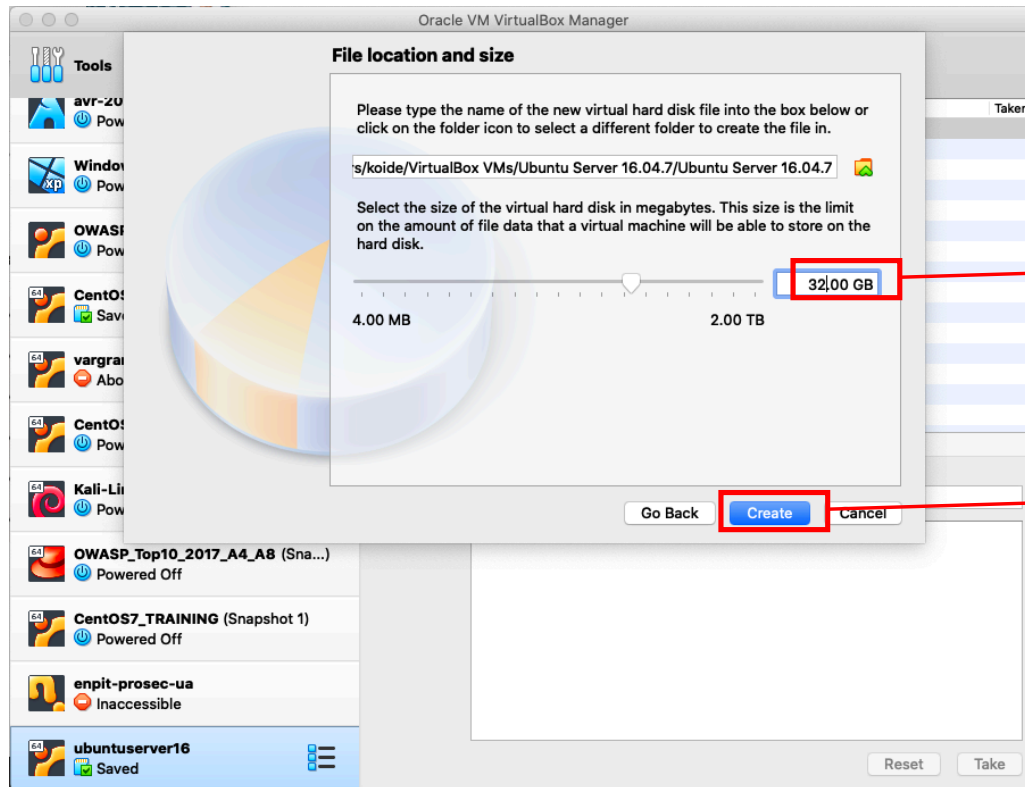
VirtualBoxにLinuxをインストール



Dynamically allocatedを選択

最後にContinueを押す

VirtualBoxにLinuxをインストール

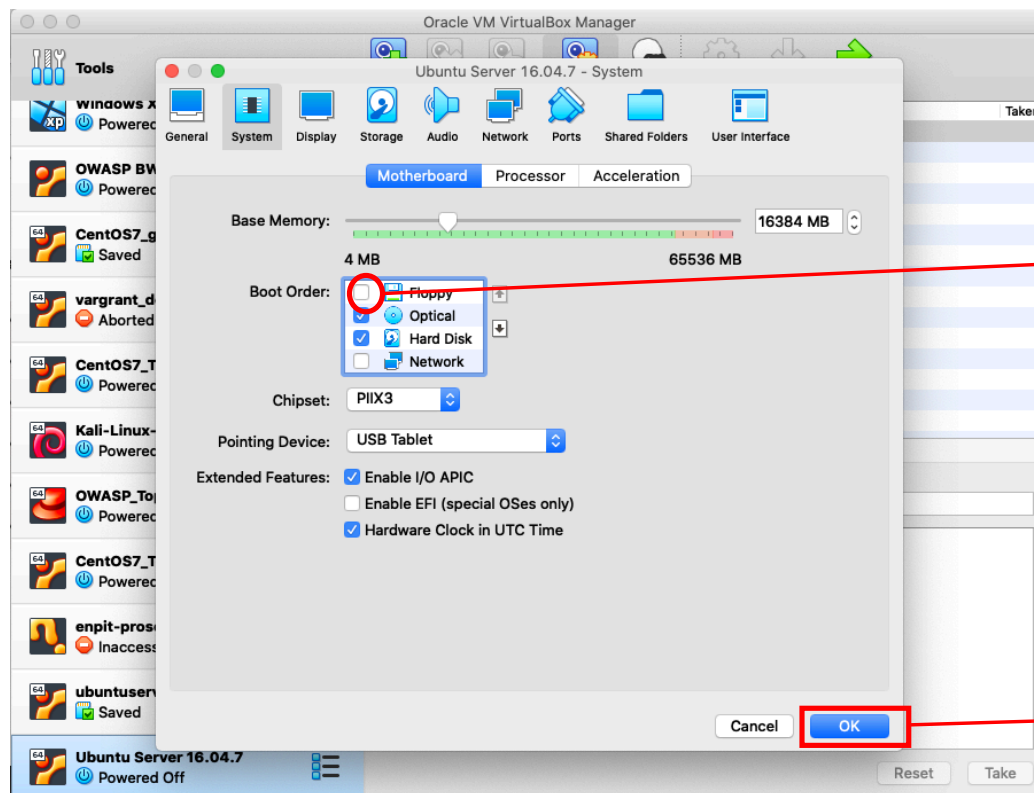


32GB以上を指定してください

最後にCreateを押す

VirtualBoxにLinuxをインストール

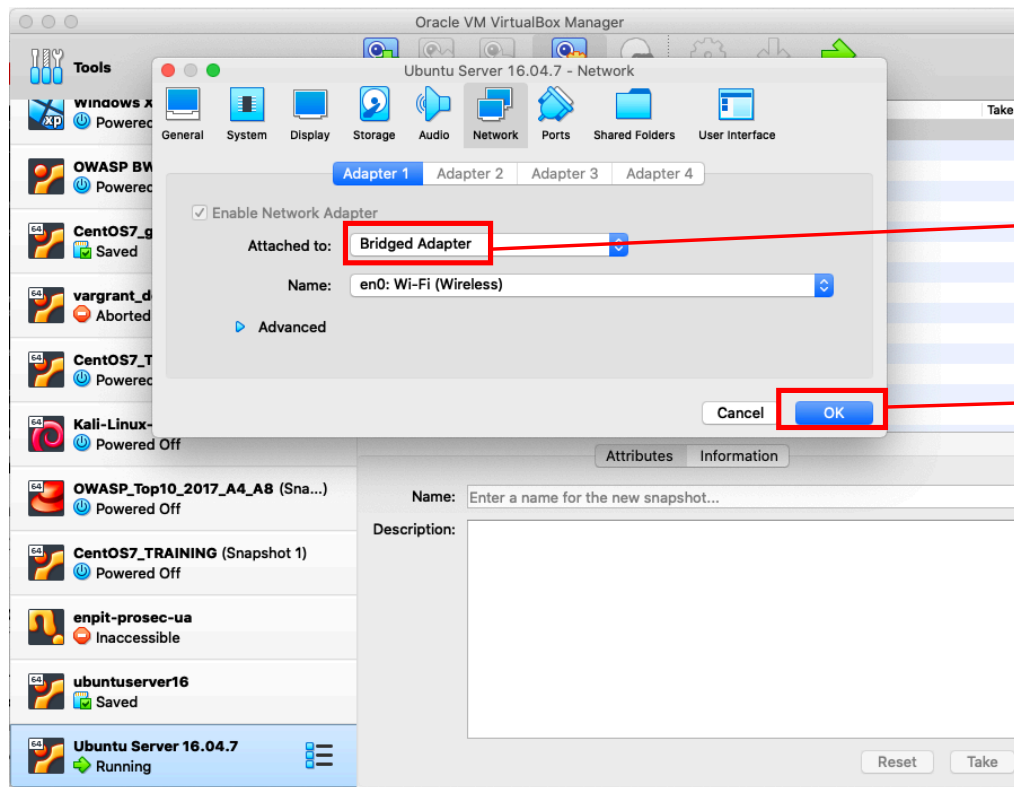
「Ubuntu Server 16.04.7」 → 「Settings」 → 「System」 → 「Motherboard」



Floppyのチェックを外す

最後にOKを押す

VirtualBoxにLinuxをインストール

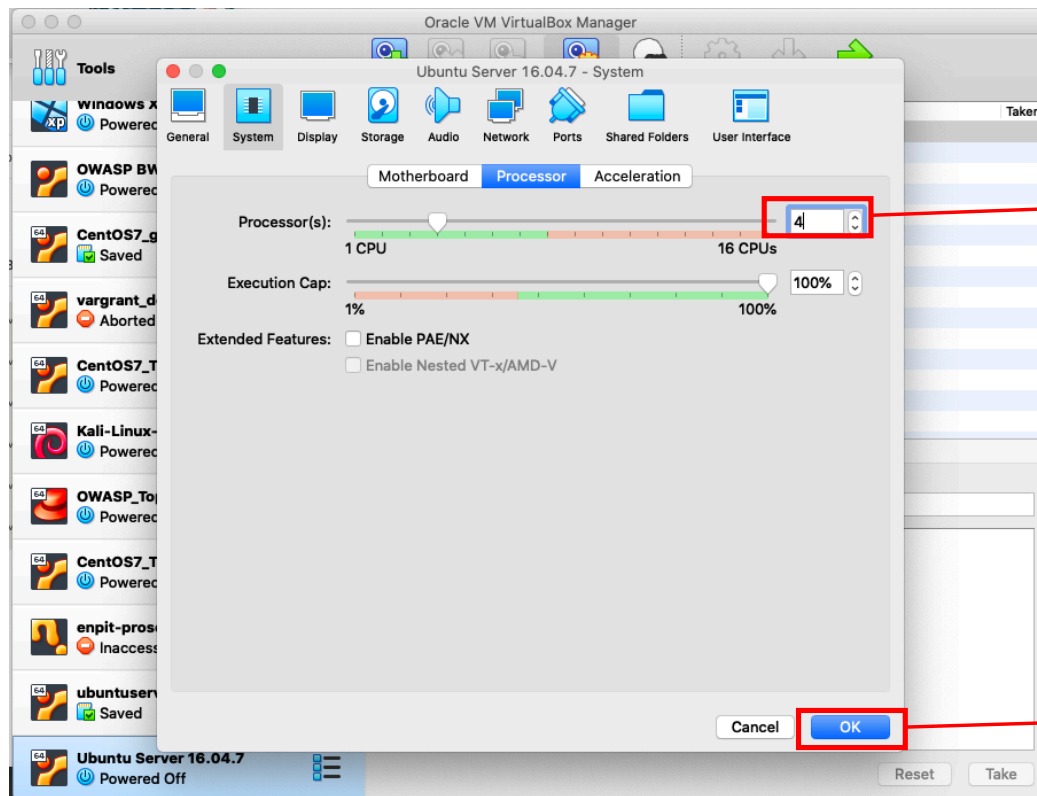


Bridged Adapter

最後にOKを押す

VirtualBoxにLinuxをインストール

「Ubuntu Server 16.04.7」 → 「Settings」 → 「System」 → 「Processor」

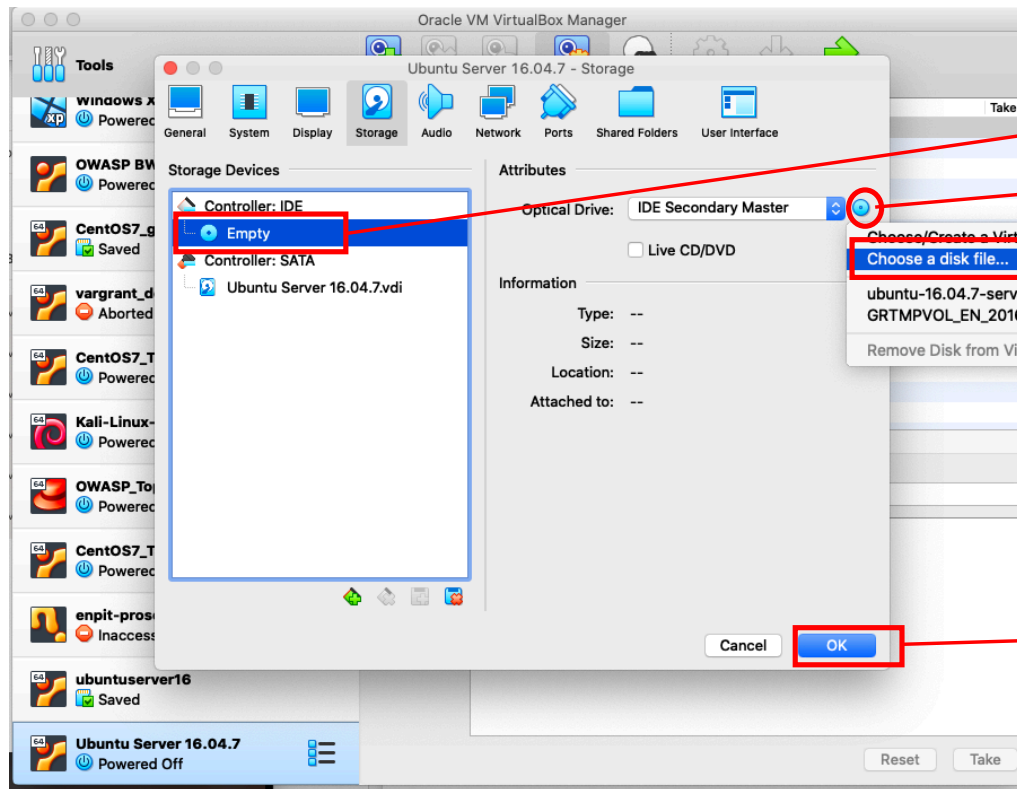


割り当てる仮想CPU数を入力
ここでは4を指定
(2以上を割り当ててください)

最後にOKを押す

VirtualBoxにLinuxをインストール

「Ubuntu Server 16.04.7」 → 「Settings」 → 「Storage」



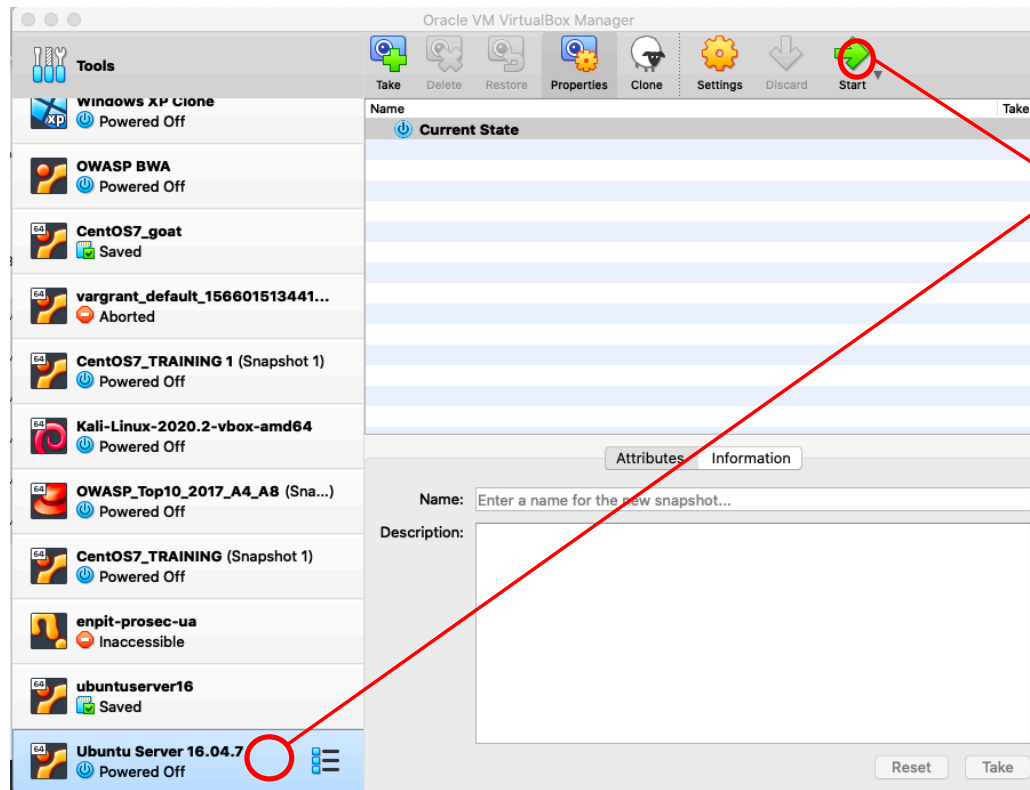
光ディスクのマークを選択

さらに光ディスクのマークを選択

ここを選ぶとファイル選択がで
てくるので、ダウンロードして
きたLinuxのインストールディス
クを選択してください

最後にOKを押す

VirtualBoxにLinuxをインストール



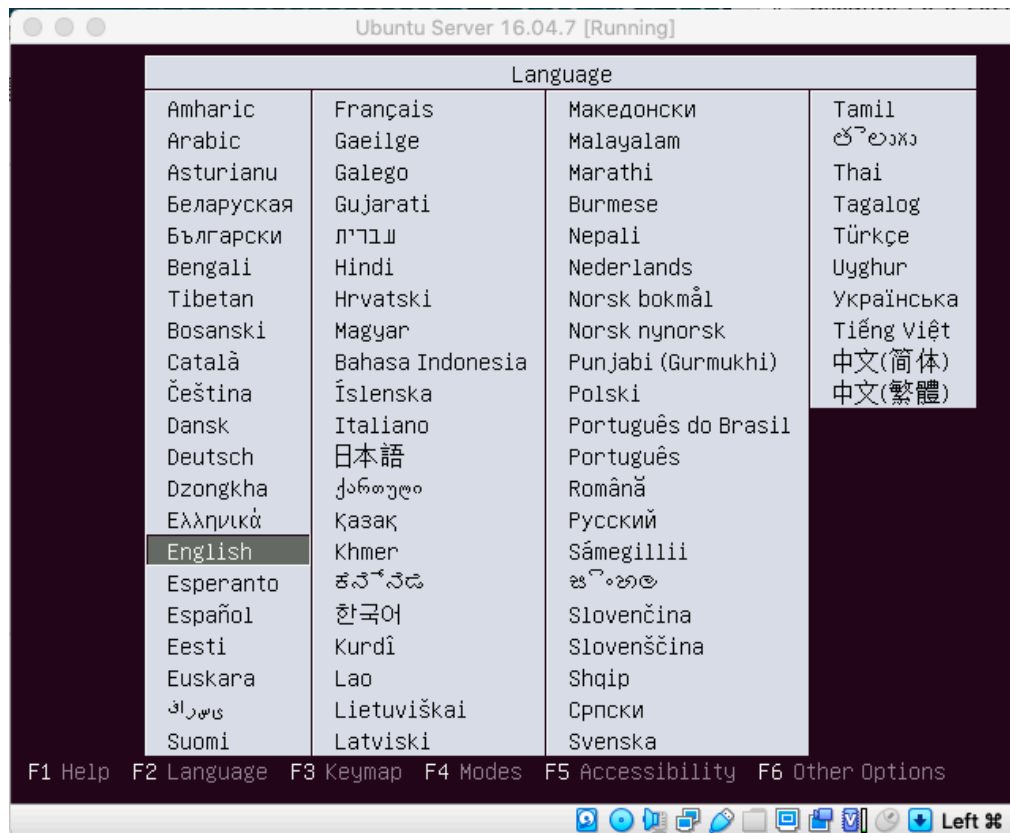
作成したインスタンス指定する

スタートボタンを押す



スタートボタンを押す

VirtualBoxをLinuxをインストール



<https://youtu.be/eGxtKtsmjSU>

カーネルの再構築

カーネルレベルのMTD演習（準備）

必要なパッケージのインストール

```
sudo apt-get -y update↓  
sudo apt-get -y install build-essential \  
libncurses-dev fakeroot kernel-package \  
linux-source libssl-dev bison flex↓
```

ソースのコピー・展開

```
mkdir ~/src↓  
cp /usr/src/linux-source-4.4.0.tar.bz2 ~/src/↓
```

参考 : <https://mongonta.com/f258-howto-build-linux-kernel/>

カーネルレベルのMTD演習（カーネルのリビルド）

カーネルソースの解凍

```
cd ~/src↓  
tar xvf linux-source-4.4.0.tar.bz2↓  
cp /boot/config-4.4.0-1112-aws ../config↓  
（または cp /boot/config-4.4.0-186-generic ../config）  
cd linux-source-4.4.0/↓  
make oldconfig↓
```

カーネルビルド（時間節約したい場合は、「システムコールテーブルの改変」までスキップ） ↓

```
export CONCURRENCY_LEVEL=3↓  
make-kpkg -j 4 --rootcmd fakeroot --initrd \  
--append_to_version=-koide --revision=001 kernel_image kernel_headers↓  
（時間がかかります, 4コア割り当てたVirtualboxで40分程度）
```

参考 : <https://mongonta.com/f258-howto-build-linux-kernel/>

カーネルレベルのMTD演習（カーネルのリビルド）

ビルドされたカーネルをインストール

```
cd ..↓
```

```
sudo dpkg -i linux-image-4.4.*-koide_001_amd64.deb↓
```

```
sudo dpkg -i linux-headers-4.4.*-koide_001_amd64.deb↓
```

リブートしてバージョンを確認

```
sudo reboot↓
```

（しばらく待ってログインし直す）

```
uname -r↓
```

（「4.4.223-koide」等と表示されたらOK）

参考：<https://mongonta.com/f258-howto-build-linux-kernel/>

カーネルレベルのMTD演習（システムコールの追加）

システムコールテーブルの改変

```
cd ~/src/linux-source-4.4.0/arch/x86/entry/syscalls/↓
```

```
vi syscall_64.tbl↓
```

(以下を追加してセーブ)

600	common	read600	sys_read
601	common	write601	sys_write
602	common	open602	sys_open
603	common	close603	sys_close
660	common	exit660	sys_exit

再度カーネルをビルド

```
cd ~/src/linux-source-4.4.0/ ↓
```

```
sudo -s ↓
```

```
export CONCURRENCY_LEVEL=3 ↓
```

```
make-kpkg -j 4 --rootcmd fakeroot --initrd --append_to_version=-koide2 --  
revision=002 kernel_image kernel_headers ↓
```

カーネルレベルのMTD演習（カーネルの再度リビルド）

ビルドされたカーネルを再度インストール

```
cd ..↓
```

```
sudo dpkg -i linux-image-4.4.*-koide2_002_amd64.deb↓
```

```
sudo dpkg -i linux-headers-4.4.*-koide2_002_amd64.deb↓
```

リブートしてバージョンを確認

```
sudo reboot↓
```

（しばらく待ってログインし直す）

```
uname -r↓
```

（「4.4.223-koide2」等と表示されたらOK）

参考：<https://mongonta.com/f258-howto-build-linux-kernel/>

カーネルレベルのMTD演習（不要なカーネルパッケージの削除）

インストールされている一覧を表示

```
dpkg --get-selections | grep linux- | grep -v deinstall ↓
```

不要なパッケージを指定して削除

```
sudo apt-get -y autoremove --purge linux-headers-4.4.223-koide ↓
```

```
sudo apt-get -y autoremove --purge linux-image-4.4.223-koide ↓
```

参考：<https://mongonta.com/f258-howto-build-linux-kernel/>

テストプログラム

read600

write601

exit660

```
// echo -e 'Hello, world\n' | strace ./a.out
```

```
#include <unistd.h>
```

```
int main()
```

```
{
```

```
    unsigned long read600 = 600L;
```

```
    unsigned long fd = 0L;
```

```
    char buf[100];
```

```
    syscall(read600, fd, buf, 14);
```

```
    unsigned long write601 = 601L;
```

```
    fd = 1L;
```

```
    long len = 14L;
```

```
    syscall(write601, fd, buf, len);
```

```
    unsigned long exit660 = 660L;
```

```
    long st = 42;
```

```
    syscall(exit660, st);
```

```
    return 0;
```

```
}
```

テストプログラムの実行結果

```
gcc test003.c ↓
```

```
echo -e "Hello, world\n" | strace ./a.out ↓
```

```
execve("./a.out", ["/a.out"], [/* 23 vars */]) = 0
```

```
<< 中略 >>
```

```
syscall_600(0, 0x7fffb5addb80, 0xe, 0x4006f0, 0x7fdcdb4d4ac0, 0x258) = 0xe ← read600が読み込んだ文字数
```

```
syscall_601(0x1, 0x7fffb5addb80, 0xe, 0x7fdcdb4d4ac0, 0x258, 0x258Hello, world
```

```
) = 0xe ← write601が読み込んだ文字数
```

```
syscall_660(0x2a, 0x2a, 0x7fdcdb1fb4d9, 0x258, 0x258, 0x258
```

```
<unfinished ...>
```

```
+++ exited with 42 +++ ← exit650が返した42
```

テストプログラム

open602

read600

close603

```
// echo -e 'Hello, world\n' > ./data ; strace ./a.out
```

```
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
```

```
int main()
{
    unsigned long open602 = 602L;
    char *filename = "./data";
    unsigned long flag = O_RDONLY;
    int fd = syscall(open602, filename, flag);

    unsigned long read600 = 600L;
    char buf[100];
    syscall(read600, fd, buf, 14);

    unsigned long close603 = 603L;
    syscall(close603, fd);
}
```

テストプログラムの実行結果

gcc test004.c ↓

echo -e 'Hello, world\n' > ./data ; strace ./a.out ↓

```
execve("./a.out", ["./a.out"], [/* 26 vars */]) = 0
<< 中略 >>
syscall_602(0x4006e4, 0, 0x4006e4, 0x4006d0, 0x7f26c9613ac0, 0) =
0x3 ← open602が返したfd
syscall_600(0x3, 0x7ffc3ce382c0, 0xe, 0x7f26c9613ac0, 0,
0x300000000) = 0xe ← read600が読み込んだ文字数
syscall_603(0x3, 0x3, 0x7f26c933a4d9, 0, 0x300000000,
0x300000000) = 0 ← close603が返した結果
exit_group(0) = ?
+++ exited with 0 +++
```

MTD演習1(課題 120分)

1. AWS EC2 あるいはVirtualBoxのインスタンスを使い、スライドに従い独自の番号付けのシステムコールを追加せよ。
2. この演習では既存のシステムコールの番号付けを入れ換えるのではなく、空いている番号に既存のシステムコールの追加のみを行っている。その理由を考察せよ。
3. (本演習のように単に独自の番号付けのシステムコールを追加するのではなく) 既存のシステムコールの番号付けを入れ換えたLinuxシステムのインスタンスを構築する方法について考察せよ。
4. 本方法によりシステムコールを入れ換えたLinuxシステムのインスタンスをひとつ構築できることが分かった。定期的にシステムコールの番号付けを変更するMTD手法を構築する方法を考察せよ。
5. 4のMTD手法を施したLinuxシステム上のWebシステムクライアントからバイナリ実行ファイルをインジェクションし、実行できる未知の脆弱性があると仮定する。その平均攻撃成功時間間隔について評価する実験方法を考察せよ。

MTD演習2：

ネットワークレベルMTD (URLシャッフルリング)

ネットワークレベルのMTD

ex1)

Webシステムにおけるネットワーク識別子を常に変更

- IPアドレス, ポート番号, パラメータ識別子

ex. <https://192.168.1.100:8080/kaikei?userid=u1010>

192.168.1.100, 8080, kaikei, userid はいずれも変更可能

when : 正規のユーザのアクセス時

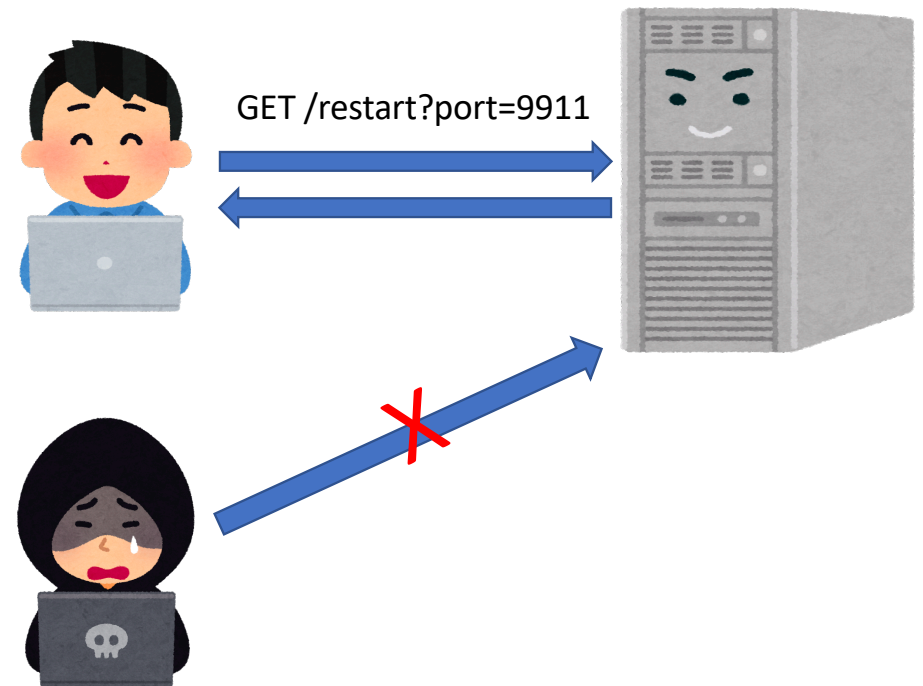
what : URLの構成要素

how : 正規のユーザ, 正規のクライアントのアルゴリズムで

MTD機能を持つWebアプリケーション

Webアプリケーションにクライアント側からポート番号を指定できる次の機能を追加

1. 最初ユーザは指定されたポート番号（初期値）でアクセス可能
2. 「GET /restart?port=新しいポート番号」により次のアクセス時の新しいポート番号に変更可能
3. それ以外は普通のWebアプリケーションの機能を持つ



MTD機能を持つ簡単なWebアプリケーションを作成

```
1 import threading
2 from urllib.parse import urlparse, parse_qs
3 from http.server import BaseHTTPRequestHandler, HTTPServer
4
5 class MyHandler(BaseHTTPRequestHandler):
6
7     def set_port(self, port = 8123):
8         self.port = port
9
10    def set_server(self, server):
11        self.server = server
12
13    def contents(self):
14        # Write content here.
15        body = b'<<This is a response.>>\n'
16        self.send_response(200)
17        self.send_header('Content-type', 'text/html; charset=utf-8')
18        self.send_header('Content-length', len(body))
19        self.end_headers()
20        self.wfile.write(body)
21
22    def do_GET(self):
23        parsed_path = urlparse(self.path)
24        queries = parse_qs(parsed_path.query)
25        self.contents()
26        if parsed_path.path == '/restart':
27            next_port = int(queries.get('port')[0])
28            self.server.running = False
29            m = MainServer(next_port)
30            m.start()
31        elif parsed_path.path == '/shutdown':
32            self.server.running = False
```

```
34 class MainServer:
35
36    def __init__(self, port = 8123):
37        handler = MyHandler
38        handler.set_port(self)
39        self.server = HTTPServer(('0.0.0.0', port), handler)
40        handler.set_server(self, self.server)
41        self.thread = threading.Thread(target=self.run)
42        self.thread.daemon = True
43
44    def set_port(self, port = 8123):
45        self.port = port
46
47    def get_port(self):
48        return self.port
49
50    def run(self):
51        self.server.running = True
52        while self.server.running:
53            self.server.handle_request()
54
55    def start(self):
56        self.thread.start()
57
58    def shut_down(self):
59        self.server.shutdown()
60
61 m = MainServer()
62 m.start()
```

サンプル実装 <https://gist.github.com/koide55/09198d833d0e9c6444c7d1d73cd126c3>

MTD機能を持つ簡単なWebアプリケーションを作成

実行例

```
$ curl localhost:8123 ↓ ← 当初はポート番号8123でアクセス可能
<<This is a response.>>
$ curl localhost:8123/restart?port=9999 ↓ ← ポート番号9999に変更
<<This is a response.>>
$ curl localhost:8123 ↓ ← ポート番号8123ではアクセスできない
^C
$ curl localhost:9999 ↓ ← ポート番号9999でアクセス可能
<<This is a response.>>
$ curl localhost:9999/restart?port=8111 ↓ ← ポート番号8111に変更
<<This is a response.>>
$ curl localhost:8111 ↓ ← ポート番号8111でアクセス可能
<<This is a response.>>
```

サンプル実装

<https://gist.github.com/koide55/09198d833d0e9c6444c7d1d73cd126c3>

MTD演習2(課題 120分)

1. スライドに従いひとつのPythonプログラムだけで試せるネットワークレベルのMTDを試してみよ（ハンズオン）。
2. 複数のサーバがネットワークレベルのMTDを用いて互いに通信しあう実験環境をデザイン，実装せよ。
3. 2の実験環境でMTD手法の評価を行うとしたら，その評価指標は何か．それらを測定するにはどのような実験を行う必要があるか．実験方法について考察せよ。
4. MTDでは正規のユーザが利用するときの余分な負担が少ないことが必要である．この立場から利用者が次のURLを指定する方法はどのように評価すべきか検討せよ。
5. 正規のユーザから指定してURLを変更するより安全な方法を考察せよ。
6. MTD演習1と組み合わせると「平均攻撃成功時間間隔」はどう変化するか考察せよ。
7. 企業内に多くのサーバが存在しており，クライアントやサーバがお互いに通信をしている状況を想定する．このときネットワークレベルのMTDは企業内部の情報システムにマルウェアやツールが侵入していることを検知するのに役立てることができるが，それはなぜか考察せよ。

課題発表，レポートの提出

- 発表形式としてオンラインで議論
- レポート提出