Linuxのフォレンジック向けメモリイメージ取得

2.1 Linuxメモリダンプツール

インシデントレスポンスやフォレンジックでメモリ解析を行う場合、私たちはVolatilityを使用しています。本レポートのVol.32の「1.4.1 Volatility Frameworkプロファイルの生成」ではVolatilityのLinux用プロファイルを作成する手順を解説しました[*1]。

今回はVolatilityで解析を行うLinuxのメモリイメージをどのように取得すれば良いか解説します。Linuxのメモリイメージを取得するツールはいくつかありますが、今回はLiME[*2]とcrash[*3]というツールを紹介します。また、ディスクフォレンジックに極力影響がないようなメモリイメージ取得方法も紹介します。動作環境はCentOS 7.7-1908を想定しています。メモリイメージを取得するツールは、これら以外にもLinpmem[*4]がありますが、筆者の検証環境ではうまく動作しなかったため、今回は紹介を見送ることにしました。

なお、本稿では調査が行われる側のコンピュータを「解析対象ホスト」、調査を行う側のコンピュータを「調査用ホスト」と記述します。解析対象ホストのデータを極力、変更しないようにするため、解析ツールのコンパイルなどは解析対象のホストとは別に用意し、そのホスト上で行ってください。

2.2 LiMEとは

LiMEはLinux Memory Extractorの略でVolatilityがメモリイメージの取得に推奨している[*5]ツールです。LiMEはLinuxのカーネルモジュールとして動作するため、解析対象ホストで動作しているカーネルのバージョンに合わせてLiMEをコンパイルする必要があります。

2.3 LiMEのコンパイル

LiMEのコンパイルを行うには、最初に図-1のようにgitコマンドでLiMEのソースコードを取得するか、GitHubのLiMEのページからzipファイルをダウンロードして適当なディレクトリに展開します。ソースコードを展開したディレクトリ内の「src」ディレクトリにMakefileがあるので、このディレクトリでmakeコマンドを実行すればLiMEモジュールが生成されます（図-2の赤枠で囲ったファイル）。LiMEのコンパイルには、カーネルモジュールのコンパイルに必要なパッケージ（kernel-develやgccなど）が必要になるため、makeコマンド実行時にエラー

```
$ git clone https://github.com/504ensicsLabs/LiME.git
```

図-1 LiMEのGitリポジトリをクローン

```
$ cd LiME/src
$ make
$ ls
disk.c  lime-3.10.0-1062.el7.x86_64.ko  lime.o    Makefile.sample  tcp.o
disk.o  lime.h                          main.c    modules.order
hash.c  lime.mod.c                      main.o    Module.symvers
hash.o  lime.mod.o                      Makefile  tcp.c
```

図-2 LiMEモジュールをコンパイル

```
$ sudo yum install kernel-devel gcc
```

図-3 kernel-develパッケージをインストール

*1 Internet Infrastructure Review（IIR）Vol.32 1.4.1 Volatility Frameworkプロファイルの生成（https://www.iij.ad.jp/dev/report/iir/032/01_04.html）。
*2 LiME（https://github.com/504ensicsLabs/LiME）。
*3 crash（https://people.redhat.com/anderson/）。
*4 Velocidex/c-aff4（https://github.com/Velocidex/c-aff4/releases）。
*5 Linux · volatilityfoundation/volatility Wiki（https://github.com/volatilityfoundation/volatility/wiki/Linux#acquiring-memory）。

が発生する場合は図-3のように関連するパッケージをインストールしてみてください。

なお、kernel-develパッケージのインストール時にバージョンを指定しない場合、最新バージョンのパッケージがインストールされます。解析対象ホストで異なるバージョンのカーネルが動作している場合、図-4のようにyumコマンドで検索を行って適切なバージョンのkernel-develパッケージをインストールし、LiMEモジュールのコンパイルを行ってください。この際、makeコマンドにKVERオプションが必要になります。

更にOSのバージョンが異なる場合には、図-5のように該当のkernel-develパッケージを個別にダウンロードし、cpioコマンドでパッケージを展開後、KVERとKDIRオプションを指定したmakeコマンドを実行することで、該当のバージョン用のLiMEモジュールを作成することができます。KVERはカーネルバージョンを、KDIRはkernel-develパッケージを展開したディレクトリをそれぞれ指定します。その他、必要なパッケージがあればインストールします（筆者の環境ではelfutils-libelf-develをインストールする必要がありました）。このように作成したLiMEモジュールはCentOS 8（1905）で動作することを確認しました。

2.4 外部ドライブへのメモリダンプ

メモリイメージを取得する際、解析対象ホストのディスクに極力書き込みを行わないようにする必要があります。特にメモリイメージは数GB以上のファイルになることがほとんどであり、解析対象ホストのディスクに書き込みを行った場合、多くの未使用領域を上書きすることになるため、ディスクフォレンジックによる調査に大きな影響を及ぼす可能性があります。

そのため、解析対象ホストに物理的アクセスができるのであれば、LiMEモジュールをコピーしたUSBメモリやモバイルSSDを解析対象ホストに接続し、図-6のようにinsmodコマンドでカーネルにLiMEモジュールをロードすることで、解析対象ホストのディスクにほとんど書き込みを行わずにメモリイメージを取得することができます。図-6のダブルクォーテーションで囲っている部分はLiMEモジュールのオプションです。この例の場合、/mediaにマウントされたUSBメモリにlimeフォー

```
$ yum --showduplicates search kernel-devel
$ sudo yum install kernel-devel-3.10.0-1062.1.2.el7.x86_64
$ make KVER=3.10.0-1062.1.2.el7.x86_64
```

図-4 カーネルバージョンを指定したLiMEのコンパイル（1）

```
$ curl -OL http://ftp.iij.ad.jp/pub/linux/centos/8.0.1905/BaseOS/x86_64/os/Packages/kernel-devel-4.18.0-80.11.2.el8_0.x86_64.rpm
$ rpm2cpio ./kernel-devel-4.18.0-80.11.2.el8_0.x86_64.rpm | cpio -id
$ sudo yum install elfutils-libelf-devel
$ make KVER=4.18.0-80.11.2.el8_0.x86_64 KDIR=~/src/usr/src/kernels/4.18.0-80.11.2.el8_0.x86_64/
```

図-5 カーネルバージョンを指定したLiMEのコンパイル（2）

```
$ sudo insmod /media/lime-3.10.0-1062.el7.x86_64.ko "path=/media/centos77.mem format=lime"
```

図-6 LiMEモジュールのロード

マットのダンプファイルを/media/centos77.memというファイル名で保存します。

LiMEモジュールがロードされるとメモリダンプが開始されますが、メモリダンプが完了するまでコマンドプロンプトは返ってきません。モジュールをロードした後に操作ができなくなっても焦らずにダンプが終了するまで待ってください。特に最近のサーバではメモリ容量が大きいため時間がかかることが予想されます。なお、メモリダンプが完了してもLiMEモジュールはロードされたままなので、図-7のようにrmmodコマンドでモジュールをアンロードします。

2.5 ネットワーク経由でのメモリダンプ

上記のように解析対象ホストに物理的にアクセスできない場合（例えば、解析対象ホストが遠隔地にあるため物理的なアクセスが容易ではない、など）、または大容量のUSBメモリがない場合、他のツールと組み合わせてネットワーク経由でメモリイメージを取得します。ここでは、Netcat、NFS、SSHの3種類のツールと組み合わせる方法を紹介します。また、調査用ホストのIPアドレスを192.168.232.131、解析対象ホストのIPアドレスを192.168.232.132とします。

■ Netcat（1）

LiMEモジュールはネットワーク経由でメモリダンプする機能を持っており、この機能とNetcatコマンドを組み合わせてネットワーク経由でメモリイメージを取得します。調査用ホストにNetcatコマンドをインストールし、図-8と図-9のようにコマンドを実行すると、調査用ホストからLiMEモジュールがLISTENしているポート（4444/tcp）に接続し、メモリイメージのデータを取得します。

■ Netcat（2）

上記の手順ではメモリ容量と同じだけのデータをネットワーク経由で受信することになります。調査用ホストがサーバの場合、数十ギガバイト以上のデータになることもあるため、可能な限り圧縮したいところです。解析対象ホストにもNetcatが

```
$ lsmod | grep lime
$ sudo rmmod lime
```

図-7 LiMEモジュールのアンロード

```
$ sudo insmod /media/lime-3.10.0-1062.el7.x86_64.ko "path=tcp:4444 format=lime"
```

図-8 LiMEで4444/tcpをListenする（解析対象ホスト）

```
$ nc 192.168.232.132 4444 > centos77.mem
```

図-9 Netcatでネットワーク経由でメモリイメージを取得（調査用ホスト）

```
$ nc -l 5555 > memorydump.lime.gz
```

図-10 調査用ホストで実行するコマンド

```
$ sudo insmod /media/lime-3.10.0-1062.el7.x86_64.ko "path=tcp:4444 format=lime"
別のログインセッションに切り替えて、以下のコマンドを実行する。
$ nc localhost 4444 | gzip -c | nc 192.168.232.131 5555
```

図-11 解析対象ホストで実行するコマンド

```
$ sudo yum install nfs-utils
$ sudo mkdir /mnt/nfsserv/
$ chown -R nfsnobody:nfsnobody /mnt/nfsserv/
$ sudo vi /etc/exports
$ sudo systemctl start nfs.service
$ sudo systemctl status nfs.service
```

図-12 NFSサーバ（調査用ホスト）の設定

```
/mnt/nfsserv/ 192.168.232.132(rw,all_squash)
```

図-13 調査用ホストのNFSエクスポート設定（/etc/exports）

インストールされていれば、図-10と図-11のようなコマンドを実行することでメモリイメージをgzipで圧縮しながら調査用ホストに転送することが可能です。先ほど既述したように、LiMEモジュールを読み込むとコマンドプロンプトは返ってきませんので、「nc」以降のコマンドは別のログインセッションから実行する必要があります。

この例では、まず調査用ホストでNetcatを使用して5555/tcpでLISTENを行います。次に解析対象ホストでNetcatを使ってLiMEモジュールがLISTENしている4444/tcpに接続し、メモリイメージを取得後、gzipで圧縮し、更にNetcat経由で調査用ホストに転送します。

■ NFS

解析対象ホストがNFSボリュームをマウントすることが可能である場合、解析対象ホストがアクセス可能なネットワーク上に調査用ホストを用意します。そして、調査用ホストでNFSボリュームを読み書き可能な設定でエクスポートします。このNFSボリュームにLiMEをコピーしておき、解析対象ホストからNFSマウントすれば、解析対象ホストにファイルを作成することなくメモリイメージを取得することが可能です（図-12、図-13、図-14、図-15）。

■ SSH

NetcatやNFSが使えない場合、代わりにSSHを使用することができます。図-16のようなコマンドを実行することで調査用ホストにメモリイメージをSSH経由で転送することができます。ただし、この手法が使えるのはbashのみになります。また、Netcat（2）の場合と同様に、exec以降のコマンドは別のログインセッションから実行する必要があります。

2.6 crashとは

crashコマンドはLinuxのメモリイメージを解析するためのツールです。解析を行うのが主目的のツールですが、メモリダンプを行うためのモジュールも使用することができます。ただし、crashのメモリダンプモジュールはRPMパッケージ

```
$ sudo firewall-cmd --permanent --add-service=nfs
$ sudo firewall-cmd --reload
$ sudo exportfs -v
```

図-14 調査用ホストのファイアウォール設定とNFSエクスポート確認

```
$ sudo mount -t nfs 192.168.232.131:/mnt/nfsserv/ /mnt/
$ sudo insmod /mnt/lime-3.10.0-1062.el7.x86_64.ko "path=/mnt/centos77.mem format=lime"
```

図-15 NFSマウントの実行とメモリイメージ取得（解析対象ホスト）

```
$ sudo insmod /media/lime-3.10.0-1062.el7.x86_64.ko "path=tcp:4444 format=lime"
別のログインセッションに切り替えて、以下のコマンドを実行する。
$ exec 5<>/dev/tcp/127.0.0.1/4444; cat <&5 | ssh -c user@192.168.232.131 'cat > centos77.mem'
```

図-16 解析対象ホストで実行するコマンド

が用意されていないため、ソースパッケージからコンパイルする必要があります(図-17)。また、crashコマンドの動作にはデバッグシンボル付きのカーネルが必要になるため、デバッグ用のカーネルパッケージをインストールします(図-18)。次に、図-19の3つのファイルをUSBメモリ内の同じディレクトリにコピーします。解析対象ホストでは、図-20のようにcrashを実行してメモリイメージを取得します。

なお、解析用ホストと解析対象ホストのカーネルバージョンが異なる場合は、図-4を参考に適切なバージョンのkernel-debuginfoパッケージを検索してください(yum --showduplicates search)。更に図-5を参考にパッケージをダウンロード及び展開(curl、rpm2cpio、cpioコマンド)し、vmlinuxファイルをコピーしてください。crashコマンドもOSバージョンに合わせたバージョンを使った方が良いでしょう(例えば、CentOS 8.0ではcrash-7.2.3-18が提供されています)。

2.7 メモリイメージの解析

Volatility[*6]でLinuxのメモリイメージを解析するには、Linuxカーネルのバージョンに応じたプロファイルが必要になります。冒頭で述べたように、Volatilityプロファイルの作成手順はIIR Vol.32[*1]で取り上げましたので、手順が分からない方は参考にしてください。

また、本稿の執筆中にLiMEモジュールとVolatility用Linuxプロファイルを自動生成して公開するBitbucketリポジトリをLorenzo Martínez氏が公開しました[*7]。このリポジトリは新しいバージョンのLinuxカーネルパッケージがリリースされるたびに更新されます。ただし、自動生成する対象のOSはCentOS 5、6、7、8及びUbuntu 14.04 LTS、16.04 LTS、18.04 LTSになります。リポジトリのWebページでカーネルバージョンでフィルタリングを行えば、該当するLiMEモジュールとVolalityプロファイルをダウンロードすることができます。

```
$ sudo yum install crash crash-devel
$ yumdownloader --source crash
$ rpm -ivh crash-7.2.3-10.el7.src.rpm
$ cd rpmbuild/SPECS
$ rpmbuild -bp crash.spec
$ cd ../BUILD/crash-7.2.3
$ make extensions
```

図-17 crashのソースパッケージインストールとモジュールのコンパイル

```
・/usr/bin/crash
・rpmbuild/BUILD/crash-7.2.3/extensions/snap.so
・/usr/lib/debug/usr/lib/modules/3.10.0-1062.el7.x86_64/vmlinux
```

図-19 メモリダンプに必要なファイル

```
$ sudo yum install --enablerepo=base-debuginfo kernel-debuginfo-3.10.0-1062.el7.x86_64
```

図-18 デバッグシンボル付きカーネルインストール

*6 The Volatility Foundation - Open Source Memory Forensics(https://www.volatilityfoundation.org/)。
*7 Lorenzo Martínez氏のツイート(https://twitter.com/lawwait/status/1181469996821700609)。

なお、x86_64以外のアーキテクチャには対応していないため、自動生成の対象ではないLinuxディストリビューションやアーキテクチャを使っている場合は自身でそれらを用意する必要があります。また、Linuxカーネルをカスタマイズして使用している場合も同様です。crashを使うのであれば、自身でデバッグシンボル付きのカスタマイズカーネルも用意する必要があります。

2.8 メモリイメージ取得時のTips

Linuxカーネル2.4からtmpfsというファイルシステムを使うことができます。tmpfs内のデータはメモリ上のみに保持され、ホストのシャットダウンや再起動を行うと保存内容が消えるため、通常はテンポラリディレクトリなどファイルが消えても問題ない用途に使用されます。しかし、この特性を利用して、攻撃者はアンチディスクフォレンジックのためにこの領域をファイル置き場として使用することが確認されています。

そのため、Volatilityにはlinux_tmpfsというLinux専用のコマンドが用意されています。これは、tmpfs内のファイルを復元するコマンドです。図-21では、/home/user/tmpディレクトリにマウントされたtmpfsのファイルを復元しています。コマンドの結果、「tmpfs_example.txt」というファイルが復元され、ファイル内容が「hello!!」であることが確認できます。

しかし、メモリ空き容量が少なくなるとtmpfsの内容はスワップアウトされてしまうため、メモリダンプを行ってもtmpfs内のデータを復元することはできません（図-22）。このような場合の対策として一時的にスワップを無効化することが考えられます。つまり、スワップアウトしたデータを強制的にスワップインさせるということです。ただし、この操作を行うことができるか否かは、解析対象ホストのメモリ使用状況に依存します。一時的にメモリ使用量が上がったことが原因でスワップアウトし、その後、メモリが解放されたような状況であればス

ファイルをコピーしたディレクトリに移動後、以下のコマンドを実行する。

```
$ sudo ./crash ./vmlinux
（以降、crashコマンドのプロンプト）
extend ./snap.so
snap centos77.mem
```

図-20 メモリイメージの取得

```
$ hexdump -C ~/vol_output/swapout/tmpfs_example.txt
00000000  00 00 00 00 00 00 00 00                           |........|
00000008
```

図-22 スワップアウトによりtmpfs内のデータ復元に失敗した例

```
$ python2 ./vol.py --profile=LinuxCentOS77x64 -f ~/tmpfs_swapoff.mem linux_tmpfs -L
Volatility Foundation Volatility Framework 2.6.1
1 -> /sys/fs/cgroup
2 -> /run
3 -> /home/user/tmp
4 -> /dev/shm

$ python2 ./vol.py --profile=LinuxCentOS77x64 -f ~/tmpfs_swapoff.mem linux_tmpfs -S 3 -D ~/vol_output/
$ hexdump -C ~/vol_output/tmpfs_example.txt
00000000  68 65 6c 6c 6f 21 21 0a                           |hello!!.|
00000008
```

図-21 tmpfsに保存されたファイルの復元と内容の確認

ワップ無効化ができる可能性があります。例えば、図-23のようにSwapのusedの値がMemのFreeよりも小さい場合が該当します。スワップ無効化後に取得したメモリイメージを解析すると、図-24のようにtmpfsのデータが復元できることが確認できます。

ただし、この方法はメモリの未使用領域にあるデータを上書きしてしまいます。未使用領域にも有用なデータが残っている可能性があるため、スワップ無効化の前後で1回ずつメモリダンプを行うのが、Linuxのメモリフォレンジックではベターな方法です。

2.9 Volatility 3

Volatilityは長い間バージョン2系が使われてきましたが、本稿執筆中にVolatility 3のパブリックベータ版がリリースされました[*8]。すべての形式のメモリイメージで確認したわけではありませんが、筆者の環境ではWindowsのRAW形式のメモリイメージとLiMEで取得したCentOSのメモリイメージは解析することができました。大きな変更点として、Volatility 2では必須であったプロファイルを指定するオプションがなくなりました。代わりに、Volatility 3では解析対象のメモリイメージからOSの種類とバージョンを推測し、適切なシンボルテーブルを参照します。例えば、Windowsのメモリイメージを読み込ませると、マイクロソフトからPDBファイルを自動的にダウンロードおよび解析してシンボル情報を参照します。しかし、macOSとLinuxのシンボルテーブルについては、Volatility 2と同様に、事前に用意する必要があります。Volatilityの開発元が用意したシンボルテーブルを使用することもできますが、WindowsやmacOSと比べ、Linux向けのシンボルテーブルは情報が不足しているため、ほとんどの場合、ユーザが用意する必要があります。

シンボルテーブルを作成するには、dwarf2json[*9]というツールを使用します。dwarf2jsonはGo言語で書かれているため、まず、golangパッケージをインストールし、dwarf2jsonをビルドします。また、シンボル情報が付与されたLinuxカーネルも必要であるため、kernel-debuginfoパッケージをインストールします。シンボル付きのLinuxカーネルを指定してdwarf2jsonを実行すると、シンボルテーブルが作成されます

```
$ free
              total        used        free      shared  buff/cache   available
Mem:        1863248       75920      307192         768     1480136     1591860
Swap:       2097148       56776     2040372
$ sudo swapoff -a
$ free
              total        used        free      shared  buff/cache   available
Mem:        1863248      118804      247652        9800     1496792     1539936
Swap:             0           0           0
（メモリダンプ後、再度、スワップを有効化）
$ sudo swapon -a
```

図-23 メモリ使用状況の確認とスワップ無効化および有効化

```
$ hexdump -C ~/vol_output/swapoff/tmpfs_example.txt
00000000  68 65 6c 6c 6f 21 21 0a                           |hello!!.|
00000008
```

図-24 スワップ無効化後のメモリイメージから復元したtmpfsデータ

```
$ sudo yum install epel-release
$ sudo yum install golang
$ git clone https://github.com/volatilityfoundation/dwarf2json.git
$ cd dwarf2json
$ go build
$ sudo yum install --enablerepo=base-debuginfo kernel-debuginfo-3.10.0-1062.1.2.el7.x86_64
$ ./dwarf2json linux --elf /usr/lib/debug/usr/lib/modules/3.10.0-1062.1.2.el7.x86_64/vmlinux > centos77-3.10.0-1062.1.2.el7.x86_64.json
$ xz -z centos77-3.10.0-1062.1.2.el7.x86_64.json
$ wget https://downloads.volatilityfoundation.org/volatility3/symbols/linux.zip
$ zip ./linux.zip ./centos77-3.10.0-1062.1.2.el7.x86_64.json.xz
```

図-25 dwarf2jsonのビルドとシンボルテーブルの作成

*8 Volatility Labs：Announcing the Volatility 3 Public Beta!（https://volatility-labs.blogspot.com/2019/10/announcing-volatility-3-public-beta.html）。
*9 dwarf2json（https://github.com/volatilityfoundation/dwarf2json）。

ので、これを開発元が配布しているシンボルテーブルが納められたファイル(linux.zip)に追加します。具体的なコマンドは図-25を参照してください。作成したシンボルテーブルはVolatility 3の所定のディレクトリにコピーします(図-26)。

Volatility 3は、「python3 vol.py -f <メモリイメージファイル> <プラグイン>」のような形式のコマンドラインで実行します。図-27はCentOS 7.7のメモリイメージをpstreeプラグインで解析した際の実行結果です。プロセスのネストを表す記号が「*」となっており、Volatility 2から若干フォーマットが変更されています。また、実行するプラグインの指定方法も変更されています。pstreeプラグインの場合、Volatility 2では「linux_pstree」と指定していましたが、Volatility 3では「linux.pstree.PsTree」と指定します。使用できるプラグインの一覧は「python3 vol.py -h」を実行すると分かります。

正常に動作することが確認できた一方で、いくつかの不具合も確認しました。先ほど書いたように、OSの種類とバージョンはコマンドラインで指定したメモリイメージの内容から推測されますが、メモリイメージの状態によっては、Linuxカーネルバージョンが正しく認識されず解析が失敗してしまう場合がありました。また、Windowsのメモリイメージではダウンロードしたpdbファイルの解析に失敗してしまい、メモリイメージの解析まで処理が進まない場合がありました。

Volatilityの開発チームはVolatility 3の正式バージョンを2020年8月に公開するとアナウンスしています。Volatility 2はその1年後の2021年8月までサポートを続けるとしていますが、今後、主流となるVolatility 3のリリースに備えて、早めに新しい使い方や設定方法などを確認しておくとよいでしょう。

```
$ cd ..
$ sudo yum install python3
$ git clone https://github.com/volatilityfoundation/volatility3.git
$ pip3 install --user pefile yara-python capstone
$ cp ./dwarf2json/linux.zip ./volatility3/volatility/symbols/
```

図-26 Volatility 3のインストールと
作成したシンボルテーブルファイルのコピー

```
$ cd volatility3
$ python3 vol.py -f ~/centos77.mem linux.pstree.PsTree
Volatility 3 Framework 1.0.0-beta.1
Progress:  23.13               Scanning LimeLayer using RegExScanner
PID     PPID    COMM

1       0       systemd
* 833   1       login
** 1695 833     bash
* 841   1       firewalld
* 843   1       NetworkManager
** 992  843     dhclient
* 558   1       systemd-journal
* 593   1       systemd-udevd
* 818   1       dbus-daemon
* 1171  1       sshd
** 1720 1171    sshd
*** 1724        1720    sshd
**** 1725       1724    bash
***** 1751      1725    sudo
****** 1753     1751    insmod
* 1172  1       tuned
* 821   1       systemd-logind
* 822   1       irqbalance
* 823   1       polkitd
* 1174  1       rsyslogd
* 1397  1       master
** 1402 1397    pickup
** 1405 1397    qmgr
(以下略)
```

図-27 Linuxメモリイメージに対してpstreeプラグインを実行

執筆者:
小林 稔(こばやし みのる)
IIJセキュリティ本部セキュリティ情報統括室フォレンジックインベスティゲーター。
IIJ-SECTメンバーで主にデジタルフォレンジックを担当し、インシデントレスポンスや社内の技術力向上に努める。
Black HatやFIRST TC、JSAC、セキュリティキャンプなどの国内外のセキュリティ関連イベントで講演やトレーニングを行う。