TEMU installation and user manual

BitBlaze Team

Nov 5th, 2009: Release 1.0 and Ubuntu 9.04

Contents

1 Introduction

This document is a quick start guide for setting up and running TEMU, the dynamic tracing component of the BitBlaze Binary Analysis Framework. It assumes that you have some familiarity with Linux. The instructions are based on the release of TEMU shown in the header, running on a vanilla Ubuntu 9.04 distribution of Linux. We intermix instructions with explanations about utilities to give an overview of how things work. The goal in this exercise is to take a simple program, trace it on some input and treat its keyboard input as symbolic. You can then use the generated trace file in the separate Vine tutorial.

2 Installation

The following script shows the steps for building and installing TEMU and the other software it depends on: (This is also found as `docs/install-temu-release.sh` in the TEMU source,

```
#!/bin/bash
# Instructions for installing TEMU 1.0 on Ubuntu 9.04 Linux 32-bit

# Things that require root access are preceded with "sudo".

# Last tested 2009-10-05
```

```
# This script will build TEMU in a "$HOME/bitblaze" directory,
# assuming that temu-1.0.tar.gz is in /tmp.
cd ~
mkdir bitblaze
cd bitblaze

# TEMU is based on QEMU. It's useful to have a vanilla QEMU for testing
# and image development:
sudo apt-get install qemu
# Stuff needed to compile QEMU/TEMU:
sudo apt-get build-dep qemu

# The KQEMU accelerator is not required for TEMU to work, but it can
# be useful to run VMs faster when you aren't taking traces.
#
# The following commands would build a kqemu module compatible with
# your system QEMU, but in Ubuntu 9.04 that would be too new to work
# with TEMU.
# sudo apt-get install kqemu-common kqemu-source
# sudo apt-get install module-assistant
# sudo module-assistant -t auto-install kqemu

# For the BFD library:
sudo apt-get install binutils-dev

# TEMU needs GCC version 3.4 (neither 3.3 nor 4.x will work)
sudo apt-get install gcc-3.4

# Unpack source
tar xvzf /tmp/temu-1.0.tar.gz

# Build TEMU
# You can select one of several plugins; "tracecap" provides
# tracing functionality.
(cd temu-1.0 && ./configure --target-list=i386-softmmu --proj-name=tracecap \
                            --cc=gcc-3.4 --prefix=`pwd`/install)
(cd temu-1.0 && make)
(cd temu-1.0 && make install)
```

3 Configuring a new VM

While QEMU itself is compatible with almost any guest OS that runs on x86 hardware, TEMU requires more knowledge about the OS to bridge the semantic gap and provide information about OS abstractions like processes. For Linux, we embed knowledge about kernel data structures

directly into TEMU; the same approach could potentially be used for Windows, but TEMU's current Windows support uses an extra driver that runs within the guest. This release of TEMU works out-of-the-box with VMs running Ubuntu Linux 9.04 32-bit. A few extra steps are required to support Windows XP or other versions of Linux.

- **Windows-based VMs** TEMU supports Windows XP (we've tested with SP1, SP2, and SP3), with the installation of a support driver. (We have not tested versions prior to XP, and Windows Vista or Windows 7 are not supported.) The driver is found in both source and binary form in the `testdrv/driver` directory of the TEMU release. To install the driver, first copy the `testdrv.sys` driver file into the `%SYSTEM32%\drivers` directory (i.e., typically `C:\Windows\system32\drivers`). Then, double-click the `testdrv.reg` file to copy its contents into the registry to configure the driver; it will then be loaded on the next reboot. To confirm that the driver is working correctly, look for a `guest.log` file created in the directory where you are running TEMU; it shows some of the data collected by TEMU.

- **Linux-based VMs** Because TEMU's Linux support requires more intimate knowledge of OS internals, it is more version-dependent than Windows support. TEMU's `kernel_table` data structure, found in `shared/read_linux.c` in the source, contains information about the location and layout of kernel global data, and the addresses of functions whose execution to monitor; unfortunately this information is different for different kernel versions and Linux distributions. As distributed, TEMU supports the kernel from a recent version of Ubuntu Linux 9.04, as well as some older ones, but you must collect the information anew to support a new kernel or distribution version.

  Most of this information (all, for some 2.4 kernels) can be collected automatically using a kernel module whose source is found in the `shared/kernelinfo` directory. There are several sample variants for different distribution versions; `procinfo-ubuntu-hardy`, which was originally created for Ubuntu 8.04 and also works for 9.04, would be a good starting point for modern 2.6-based systems. Copy the module source to your guest VM, and compile it there (you should have the kernel header files matching the running kernel installed). Then, load the module using the `insmod` command, and look for its output in the kernel's logs (e.g., `/var/log/kern.log`) or the kernel log ring buffer (displayed by the `dmesg` command). Then copy these entries to `shared/read_linux.c` and recompile TEMU. For 2.6 kernels, we haven't been able to find an appropriate hooking function that is exported to modules, so you'll need to find the address of a function that is called after a new process is created using the kernel's symbol table (usually kept in a file like `/boot/System.map-2.6.28-15-generic`), and add it as the second value in the information structure by hand. For recent kernels, we've found the function `flush_signal_handlers` works well.

After performing the above steps, you can check that things are OK by running the `guest_ps` (Windows) or `linux_ps` (Linux) command, and verifying that the current processes are correctly displayed; an error in the configuration will likely cause this command to output garbage, or cause TEMU to crash/hang.

4 Setting up TEMU network

Running QEMU by itself should be the first step, before you try to run TEMU. There are many platform specific tweaks that you may need in order to get QEMU usable for your project. Though not needed for this excercise, you will often need to set up a network inside the QEMU image that you use. You may skip this network setup section, if you will not need this.

This document does not intend to go into great depth in setting up QEMU itself. But we describe some mechanisms that have worked for us. You may need a bit Googling to set this up on your specific platform and network configuration.

- **Method 1** - User-level network emulation

  The simplest kind of network emulation, which QEMU performs by default, uses just user-level network primitives on the host side, and simulates a private network for the virtual machine. This is sufficient for many utility purposes, such as transferring files to and from the virtual machine, but it may not be accurate enough for some kinds of malicious network use. The QEMU options for enabling this mode explicitly are `-net nic -net user,hostname=mybox`, where `mybox` is the hostname for the virtual DHCP server to provide to the VM.

  If you want to connect to well-known services on the VM, you'll need to redirect them to alternate ports on the host with the `-redir` option. For instance, to make it possible to SSH to a server on the VM, give QEMU the option `-redir tcp:2022::22`, then tell your SSH client to connect to port 2022 on the local machine.

- **Method 2** - Use tap network interface.

```
Create a script /etc/qemu-ifup, including the following lines. Be sure to make
this script executable.
#!/bin/sh
sudo /sbin/ifconfig $1 192.168.10.1

You must then setup a tap interface. This step can be skipped if you
are willing to run QEMU as root.
$ sudo apt-get install uml-utilities
$ sudo /usr/sbin/tunctl -b user -t tap0

Start the Windows VM. The host machine will have the IP address
192.168.10.1, as is specified in the above script.
$ sudo chmod 666 /dev/net/tun
$ qemu -kernel-kqemu -snapshot -net nic,vlan=0 \
  -net tap,vlan=0,script=/etc/qemu-ifup \
  -monitor stdio /path/to/qemu/image

If you don't want to type these commands each time you start TEMU,
you can create a wrapper script which initializes the network,
starts TEMU with desired command-line arguments, then removes the
tap interface once TEMU exits.
```

Once QEMU is set up and running, TEMU should run in the same way. You can run TEMU's `qemu` as root, just the same way as you run QEMU using the installed `qemu` in the PREFIX directory.

5 Taking traces

Assuming that you have compiled TEMU and you have identified the command line to launch your QEMU session, we can now go ahead and try out a simple example trace. Here we demonstrate the procedure for a Ubuntu 9.04 Linux image; the commands are mostly the same for a Windows image.

The command-line options for TEMU are mostly the same as for QEMU. Besides whatever options are needed for your virtual machine to run correctly, the example below adds two more. `-snapshot` tells QEMU not to write changes to the virtual hard disk back to the disk image file unless explicitly requested, so you don't have to worry about messing up your VM with experiments gone awry. `-monitor stdio` tells QEMU to put up a command-line prompt on your terminal, which we will use to give TEMU commands.

A command line to launch TEMU looks like:

```
% cd ~/bitblaze/temu
% ./tracecap/temu  -snapshot -monitor stdio ~/images/ubuntu904.qcow2
```

The output on the console is:

```
QEMU 0.9.1 monitor - type 'help' for more information
(qemu)
```

You may also see a warning indicating that `kqemu` is disabled for one reason or another; these may mean that your VM will run more slowly, but can otherwise be ignored.

1. *Generate a simple program in the QEMU image:* In the guest Linux session, create a `foo.c` program as follows, and start it:

```
$ cat foo.c
#include <stdio.h>

int main(int argc, char **argv)
{
  int x;
  scanf("%d", &x);
  if (x != 5)
      printf("Hello\n");
  return 0;
}
```

```
$ gcc foo.c -o foo
$ ./foo
```

2. *Load the TEMU plugin*

   At the `(qemu)` prompt, say:

```
(qemu) load_plugin tracecap/tracecap.so
Cannot determine file system type
tracecap/tracecap.so is loaded successfully!
(qemu) enable_emulation
Emulation is now enabled
```

   The warning about `Cannot determine file system type` applies to functionality we won't be using, and can be ignored. `enable_emulation` is required to activate any of TEMU's per-instruction tracing hooks; without it, later steps won't see any of the instructions executed.

3. *Find out the process id you the program you want to trace:*

   In the `(qemu)` prompt, run the `linux_ps` command to find the process id of the `./foo` application running in the guest Linux image.

```
(qemu) linux_ps
    0  CR3=0x00000000  swapper
    1  CR3=0xC7DEA000  init
         0x08048000 -- 0x0804E000 init
         0x0804E000 -- 0x0804F000 init
         0x0804F000 -- 0x08053000
         0x40000000 -- 0x40013000 ld-2.2.5.so
         0x40013000 -- 0x40014000 ld-2.2.5.so
         0x40022000 -- 0x40023000
         0x42000000 -- 0x4212C000 libc-2.2.5.so
         0x4212C000 -- 0x42131000 libc-2.2.5.so
         0x42131000 -- 0x42135000
         0xBFFFD000 -- 0xC0000000
         .....
  958  CR3=0xC51A1000  foo
         0x08048000 -- 0x08049000 foo
         0x08049000 -- 0x0804A000 foo
         0x40000000 -- 0x40013000 ld-2.2.5.so
         0x40013000 -- 0x40014000 ld-2.2.5.so
         0x40014000 -- 0x40015000
         0x42000000 -- 0x4212C000 libc-2.2.5.so
         0x4212C000 -- 0x42131000 libc-2.2.5.so
         0x42131000 -- 0x42135000
```

```
0xBFFFE000 -- 0xC0000000
....
```

The PID, here 958, is shown on the header line for the named process. The other information isn't relevant for what we're doing, but if you're curious, the `CR3` value is a pointer to the kernel-space page table for each process, and the remaining lines show the virtual address ranges for the process's various segments (mappings), which are either text or data segments from executables or shared libraries, or anonymous heap or stack areas.

For a Windows image, you need to run the `guest_ps` command instead of `linux_ps`.

4. *Trace the process, and record the instructions it executes in a file:*

   The `trace` command takes the process id and the name of a trace file to write information into, as shown below.

```
(qemu) trace 958 "/tmp/foo.trace"
PID: 958 CR3: 0x06301000
PROTOS_IGNOREDNS: 0, TABLE_LOOKUP: 1 TAINTED_ONLY: 0
 TRACING_KERNEL_ALL: 0 TRACING_KERNEL_TAINTED: 0 TRACING_KERNEL_PARTIAL: 0
```

   As an alternative to steps 3-4 above, you can also tell TEMU to begin tracing a program before you've loaded it, with the `tracebyname` command. This command will monitor new processes and automatically trace the next instance of the target program. Example usage of this command is shown below.

```
(qemu) tracebyname foo "/tmp/foo.trace"
Waiting for process foo to start
$ ./foo
(qemu) PID: 472 CR3: 0x0a025000
Tracing foo
```

5. *Specify what input to taint, and give the input:*

   With the `taint_sendkey` command we can send input to the traced process, and also mark this input as tainted. The taint tracking engine will perform dynamic taint tracking, i.e. mark all data derived from tainted input as tainted. If any of the operands of an executed instruction are tainted, the result is also marked tainted. This command takes 2 arguments – the character (really, keyboard key) to give as input (`5` in the example below) and an identifier to identify this input in the trace (given by "1001" in the trace; it should not be zero). The trace of this process will log all data read and written at each instruction, the instruction itself, and the associated data taint in the trace file.

```
(qemu) taint_sendkey 5 1001
Tainting keystroke: 9 00000001
```

```
(qemu) taint_sendkey ret 1001
Tainting keystroke: 9 00000001
Time of first tainted data: 1197072993.761231
(qemu)
```

Note that TEMU is tracking taint throughout the whole simulated machine, but only tracing in the requested process. The `first tainted data` message refers to the traced program, and doesn't show up until a complete line has been typed, because the operating system is buffering the input line before that.

6. *Stop tracing and tainting:*

   We are done with tainting and tracing, so we use the following commands to turn off the components.

```
(qemu) trace_stop
Stop tracing process 958
Number of instructions decoded: 5979
Number of operands decoded: 13349
Number of instructions written to trace: 5890
Number of tainted instructions written to trace: 85
Processing time: 0.464029 U: 0.444028 S: 0.020001
Generating file: /tmp/foo.trace.functions
(qemu) unload_plugin
Emulation is now disabled
protos/protos.so is unloaded!
```

At the end, you should have a trace file generated at the file name you specified (`/tmp/foo.trace` in the example). The trace has a specific binary format which is not human-readable, but you can check that it contains some data (it should be between about 100k and 800k for this example). It contains instructions, concrete values of the operands seen in the execution of the program, and the associated taint value.

As an aside, if you want to generate traces with network input rather than keystrokes you can follow the same steps but with two changes. First, after the plugin is loaded, issue the command `taint_nic 1` to tell TEMU to mark all input received from the network card to as tainted. Second, instead of giving `taint_sendkey`, just simply direct the input to the IP address/port of the virtual machine. If the input causes the EIP to become tainted, TEMU will immediately write all trace data and quit. You can use this to launch network attacks on programs in the guest OS image.

6 Troubleshooting

This section describes some problems users have experienced when using TEMU, along with the most common causes of these problems.

1. *TEMU does not begin tracing program*

- Did you remember to `enable_emulation` before running the program?
- Did you enter to correct PID (`trace` command) or correct filename (`trace_by_name` command)?
- Are you using a supported operating system? TEMU has been preconfigured for Ubuntu Linux 9.04, and requires a driver to be installed on Windows systems.

2. *Generated trace file is empty*

   - Did you remember to run `trace_stop`?
   - Are you using the *tracecap* plugin? The *tracecap* can be configured to write only certain types of instructions (for instance, tainted instructions) to the trace file. Check the plugin settings in the main.ini file.
   - Did you load any HOOK files? Configuration settings in HOOKs may sometimes disable writing to the trace file until certain trigger conditions are met.

3. *No tainted instructions were written to the trace file*

   - Was any tainted data accessed by the traced program?
   - Are you loading tainted data from hard drive? Caching by the OS sometimes causes data from primary hard disk to be "missed" by TEMU. Try loading the tainted data from a secondary hard disk.

4. *Compile warnings about* `fastcall`

   - Are you trying to compile with GCC 3.3? It isn't supported.

5. *Missing symbols starting with* `_sch_`

   - These indicate a problem linking with the GNU Binutils. Make sure you have matching development and runtime versions of its libraries installed, and that `/usr/lib/libbfd.so` exists.

6. *TEMU can't find a BIOS image or keymap*

   - Either run `make install` to put these in the locations TEMU is expecting, or give their locations with the -L flag.

7. `linux_ps` *loops or prints garbage*

   - This can be caused by TEMU having incorrect information about your Linux kernel. Check that the version you are running is one of the already supported ones, or provide that information as described in Section 3

7 Acknowledgements

TEMU's Tracecap plugin links with OpenSSL (copyright 1998-2004 the OpenSSL Project), Sleuthkit (portions copyright 1997-1999 IBM and other authors), XED (copyright 2004-2009 Intel), and ll-conf (copyright 2004-2007 Oliver Kurth). However like TEMU itself our redistribution of that code is WITHOUT ANY WARRANTY.

8 Reporting Bugs

Though we cannot give any guarantee of support for TEMU, we are interested in hearing what you are using it for, and if you encounter any bugs or unclear points. Please send your questions, feature suggestions, bugs (and, if you have them, patches) to the bitblaze-users mailing list. Its web page is: http://groups.google.com/group/bitblaze-users.