# Know Your Tools:
# Qebek – Conceal the Monitoring

*The Honeynet Project*
http://www.honeynet.org

*Chengyu Song – The Honeynet Project*
*Brian Hay – The Honeynet Project*
*Jianwei Zhuge – The Honeynet Project*

Last Modified: *31 October 2010*

## INTRODUCTION

For the last few years, while low-interaction (LI) honeypot systems like Nepenthes and PHoneyC are getting more and more powerful, the progress of high-interaction (HI) honeypot technology has been somewhat slower. This is especially true for Sebek, the de-facto HI honeypot monitoring tool. In this KYT paper, we introduce Qebek, a QEMU based HI honeypot monitoring tool which aims at improving the invisibility of monitoring the attackers' activities in HI honeypots.

In the first part, we will first introduce the background and motivations. In the second part, we will show how to use Qebek. Since Qebek is still in its initial state, to help those who have interests in improving or extending this tool, in the third part, we will present the design and some important implementation details.

At this point Qebek only supports Windows guests, so in our examples we will deploy and monitor a Windows HI honeypot running on a Linux host OS.

## MOTIVATIONS

Sebek is the most widely used HI honeypot monitoring tool, and is a major component of the GenIII honeynet architecture[1]. However, at the time we launch Qebek project, the latest version of Sebek has several problems:

- It lacks **invisibility**, which means that it could be detected, subverted, and/or disabled by the attacker;
- The design lacks on the ability to **correlate** system activities with the network activities;
- The Win32 client version is **unstable**, which usually causes Blue Screen of Death (OS crash).

Among these problems, the visibility problem is probably the most difficult to address with the current HI honeypot monitoring approach, in which instrumentation is executed on the HI honeypot itself. Due to the fact that attackers can also execute their codes in the same security layer (i.e. kernel level) as Sebek, attackers are able to detect and subvert it. Sebek has already includes many anti-detection mechanisms designed to make the process of detection more challenging for an attacker, but an attacker who has fully compromised the HI honeypot

[1] Towards a Third Generation Data Capture Architecture for Honeynets, Edward Balas and Camilo Viecco, in Proc. 6th IEEE Information Assurance Workshop.

will at some point be able to detect the presence of the monitoring tool by importing new technologies. This is also the problem many HIDS/HIPS systems have to face.

To solve this arm-race problem from root, researchers introduced two important techniques: virtual machine introspection (VMI) and system view reconstruction (SVR). VMI enabled the IDS or other security system to monitor the system events from outside the virtual machine, while SVR allows the monitoring system to reconstruct meaningful high OS-level information from the raw hardware-level information generated by VMI. By leveraging these two techniques, the security monitor has been pulled from the OS layer to the virtual machine monitor (VMM) level. After that, researchers have invented many VMI-based security systems included one called VMScope, a virtualization-based honeypot monitoring system which allowed us to view the system internal events of VM-based honeypots from outside the honeypots.

Unfortunately, most of these systems are not open-sourced. This motivated us to develop Qebek, a QEMU-based HI honeypot monitoring tool to make these two great techniques public available.

## INSTALLATION

In this section we introduce how to install Qebek. The host OS we use is Ubuntu Linux.

### Prerequisites

To install and run Qebek, you need:

- A Linux host system. Qebek has been tested on Ubuntu 8.04, 9.10 and 10.04, but it should work on other distributions. We use Ubuntu 8.04 in following example. If the host supports hardware virtualization (e.g. Intel VT), we recommend you turn it on.
- A Windows XP installation disk. Qebek has only been tested on SP2, but it should work on other service levels.
- A basic knowledge of Linux system administration, especially network configuration.
- A working network. You need it to install software and get the Qebek source code.

### Install a vanilla QEMU and the build dependence of QEMU

```
sudo apt-get install qemu
sudo apt-get build-dep qemu
```

### Install GCC-3.4

Qebek is built upon QEMU 0.9.2 and GCC 3.4 is required to build QEMU version 0.9.x. Neither 3.3 nor 4.x will work.

```
sudo apt-get install gcc-3.4
```

Since Ubuntu 9.10, gcc-3.4 has been removed from the repository, but the required packages can still be found in the package archive of Jaunty (http://packages.ubuntu.com/jaunty/devel/gcc-3.4), you also need to download cpp-3.4 and gcc-3.4-base. To download the package, first click the link on your architecture (i386 for 32bit host, and amd64 for 64bit host), then choose a mirror. For example,

```
wget http://mirrors.kernel.org/ubuntu/pool/universe/g/gcc-3.4/gcc-3.4-base_3.4.6-8ubuntu2_i386.deb
wget http://mirrors.kernel.org/ubuntu/pool/universe/g/gcc-3.4/cpp-3.4_3.4.6-8ubuntu2_i386.deb
wget http://mirrors.kernel.org/ubuntu/pool/universe/g/gcc-3.4/gcc-3.4_3.4.6-8ubuntu2_i386.deb
```

Then install the downloaded packages:

```
sudo dpkg -i gcc-3.4-base_3.4.6-8ubuntu2_i386.deb
sudo dpkg -i cpp-3.4_3.4.6-8ubuntu2_i386.deb
sudo dpkg -i gcc-3.4_3.4.6-8ubuntu2_i386.deb
```

### Install Qebek

Check out the latest version of Qebek from subversion repository.

```
svn co https://projects.honeynet.org/svn/sebek/virtualization/qebek/trunk/ qebek
```

Configure and make.

```
cd qebek
chmod +x configure texi2pod.pl
./configure
make
```

If you'd like to install Qebek for other users on your system, you should also run `make install` as root at this time. If you only want to try Qebek without fully installing it the executable can be found in the *i386-softmmu* subdirectory after the build has completed.

```
sudo make install
```

## SETTING UP THE HONEYPOT

In this section we describe how to setup a typical HI honeypot with the support the Qebek.

### Network setup

One of the most frequently asked questions is the network setup, especially when setting up a virtual machine based honeynet system. In this KYT, we only introduce how to setup a honeypot with the support of Qebek. You can find more information on the topic of honeypot network setup by visiting our paper list (http://old.honeynet.org/papers/) or asking questions on our mailing list (https://public.honeynet.org/mailman/listinfo/honeywall).

Generally, QEMU supports three kinds of networking, user mode, TAP and socket. Compared with the most widely used VMware products, QEMU's user mode networking is similar to NAT in my understanding and TAP is similar to bridge mode in VMware. Since we are setting up a honeypot, which has to be publicly accessible, we will use the TAP network interface.

To setup, first install the required tools:

```
sudo apt-get install bridge-utils uml-utilities
```

Then setup a bridge to the network the honeypot will work on (assume eth0 with 192.168.0.0/24). Run the follow script with root privileges:

```
#! /bin/sh
ifconfig eth0 down
brctl addbr br0 #add a bridge
brctl addif br0 eth0 #add an interface
ifconfig eth0 0.0.0.0 promisc up
ifconfig br0 192.168.0.2 up
route add default gw 192.168.0.1 dev br0
```

Alternatively you can setup a permanent bridge by editing the networking configuration file (*/etc/network/interfaces*):

```
auto br0
iface br0 inet static
    address 192.168.0.2
    netmask 255.255.255.0
    network 192.168.0.0
    gateway 192.168.0.1
    bridge_ports eth0
    bridge_fd 1
    bridge_stp off
    bridge_hello 1

auto eth0 inet static
    address 0.0.0.0
    netmask 255.255.255.0
```

Save the following content as */etc/qebek-ifup*, the default startup networking configuration file, to setup the TAP interface when the VM starts

```
sudo tunctl -b -u root -t $1 #create the tap interface
sudo brctl addif br0 $1 #add the tap interface to br0
sudo ifconfig $1 0.0.0.0 promisc up
```

And save the following content as */etc/qebek-ifdown*, the default after-shutdown networking configuration file, to delete the TAP interface after the VM is shutdown

```
sudo ifconfig $1 down
sudo brctl delif br0 $1 #remove the tap interface from br0
sudo tunctl -d $1 #delete the tap interface
```

You could also manually configure the TAP interface (as we do), which will be introduced below. This is useful when you already have some tap interfaces.

### Honeypot setup

After setting up the network, we will setup the honeypot.

Create the virtual honeypot disk image. The format used is the recommended qcow2 which has full support for snapshot, compression and encryption.

```
qemu-img create -f qcow2 winxp.img 6G
```

Then we install the operating system for the honeypot virtual machine. To speed up the process, we have two options: one is the old fashioned kqemu accelerator; and the other one is KVM, which requires the host platform having Intel VT or AMD-v support. Here we use KVM by executing the second line shown below, but if you prefer (or are required) to use the kqemu approach then uncomment the first line below and use it instead.

```
#sudo apt-get install kqemu-common
sudo apt-get install kvm
```

Use the following script to begin installation of the OS (**manually configure the TAP interface**):

```
#! /bin/sh
tunctl -b -u root -t tap0
brctl addif br0 tap0
ifconfig tap0 0.0.0.0 promisc up

kvm -localtime \ #use local time instead of utc
    -m 512 \ #512MB memory
    -had winxp.img \ #formal created disk image
    -cdrom winxp.iso \ #installation disk image
    -net nic,vlan=0,macaddr=00:11:22:33:44:55,model=pcnet \ #NIC info
    -net tap,vlan=0,ifname=tap0,script=no,downscript=no #use tap0 and no configuration script

ifconfig tap0 down
brctl delif br0 tap0
tunctl -d tap0
```

The default NIC model is ne2k, here we use pcnet. Since MAC address is visible to attackers, you should better copy a MAC address from a **real**, **unused** Ethernet adapter **with the same model** as the emulated NIC instead of using the one generated by QEMU.

**Begin monitoring**

Since everything is captured at the emulator layer, no additional component is required to install inside the guest OS, all you need is to specify the guest OS type. You could also specify the guest IP to improve the readability of captured network activities, because the listening address in most cases is 0.0.0.0, and Qebek does not perform an automatic lookup for this information. Here is a sample startup script (*qebek.sh*):

```
#! /bin/sh
tunctl -b -u root -t tap0
brctl addif br0 tap0
ifconfig tap0 0.0.0.0 promisc up

qebek -localtime \ #use local time instead of utc
    -snapshot \ #protect the disk image
    -m 512 \ #512MB memory
    -had winxp.img \ #formal created disk image
```

```
        -cdrom winxp.iso \ #installation disk image
        -net nic,vlan=0,macaddr=00:11:22:33:44:55,model=pcnet \ #NIC info
        -net tap,vlan=0,ifname=tap0,script=,downscript= \ #use tap0 and no configuration script
        -winxp \ #the guest OS is Windows XP
        -sbk_ip 127.0.0.1 #the honeypot ip address

    ifconfig tap0 down
    brctl delif br0 tap0
    tunctl -d tap0
```

Currently, Qebek accepts most options QEMU accepts, but does not support the kqemu accelerator for kernel code nor KVM. Qebek specific options are:

| Option | Description |
|---|---|
| -winxp | The guest OS is Windows XP |
| -sbk_magic | The magic number in Sebek header when output (optional) |
| -sbk_ip | The guest OS IP address (optional) |

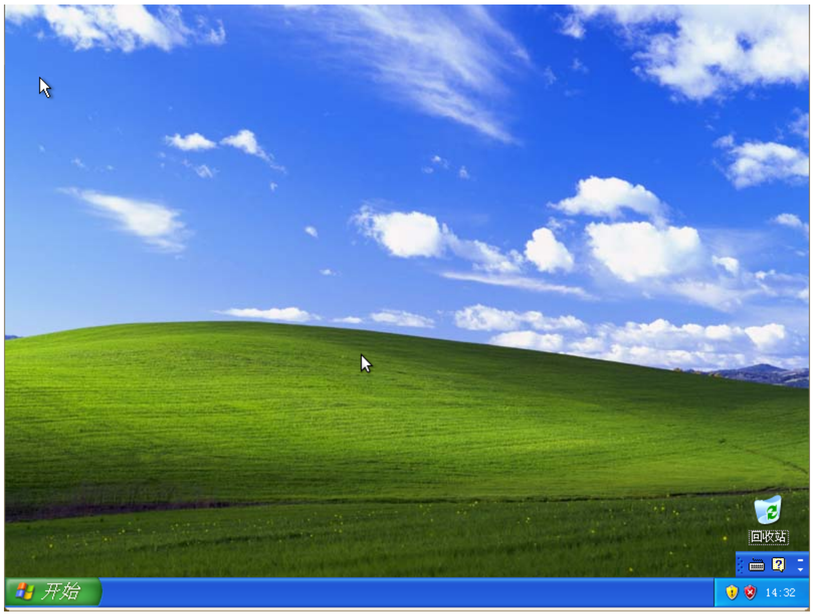
*Table 1 – Qebek specific options*

Windows XP is the only tested guest OS type, but we are working on adding supports for other OS, and any contributions are welcomed

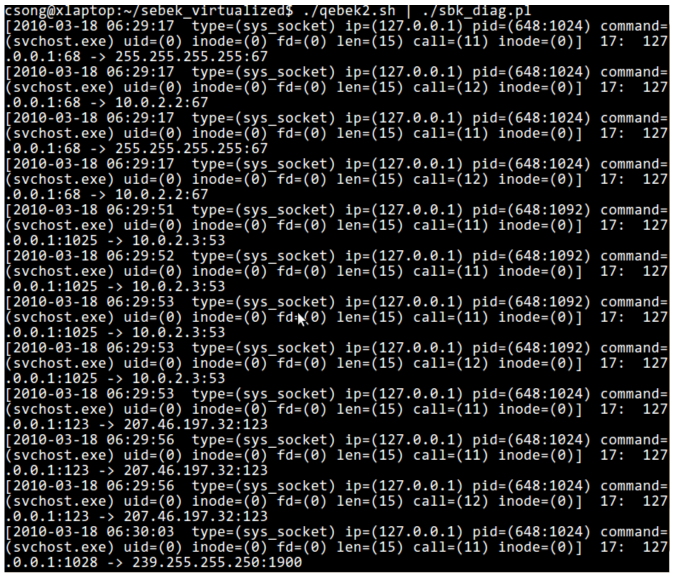
To get a more readable result we redirect the outputted information to *sbk_diag.pl* (the parser script for Sebek data packets), run the following command in a Linux console:

```
    sudo ./qebek.sh | ./sbk_diag.pl
```

Wait for the Windows guest OS to start



At this time, Qebek has captured following activities, which can be seen in the console window in the Linux host OS:

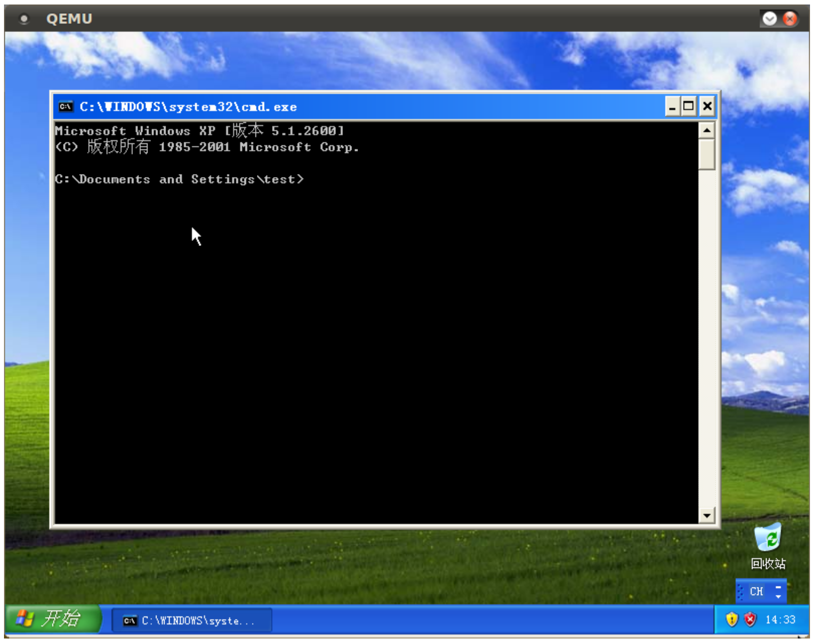


```
csong@xlaptop:~/sebek_virtualized$ ./qebek2.sh | ./sbk_diag.pl
[2010-03-18 06:29:17  type=(sys_socket) ip=(127.0.0.1) pid=(648:1024) command=
(svchost.exe) uid=(0) inode=(0) fd=(0) len=(15) call=(11) inode=(0)]  17:  127
.0.0.1:68 -> 255.255.255.255:67
[2010-03-18 06:29:17  type=(sys_socket) ip=(127.0.0.1) pid=(648:1024) command=
(svchost.exe) uid=(0) inode=(0) fd=(0) len=(15) call=(12) inode=(0)]  17:  127
.0.0.1:68 -> 10.0.2.2:67
[2010-03-18 06:29:17  type=(sys_socket) ip=(127.0.0.1) pid=(648:1024) command=
(svchost.exe) uid=(0) inode=(0) fd=(0) len=(15) call=(11) inode=(0)]  17:  127
.0.0.1:68 -> 255.255.255.255:67
[2010-03-18 06:29:17  type=(sys_socket) ip=(127.0.0.1) pid=(648:1024) command=
(svchost.exe) uid=(0) inode=(0) fd=(0) len=(15) call=(12) inode=(0)]  17:  127
.0.0.1:68 -> 10.0.2.2:67
[2010-03-18 06:29:51  type=(sys_socket) ip=(127.0.0.1) pid=(648:1092) command=
(svchost.exe) uid=(0) inode=(0) fd=(0) len=(15) call=(11) inode=(0)]  17:  127
.0.0.1:1025 -> 10.0.2.3:53
[2010-03-18 06:29:52  type=(sys_socket) ip=(127.0.0.1) pid=(648:1092) command=
(svchost.exe) uid=(0) inode=(0) fd=(0) len=(15) call=(11) inode=(0)]  17:  127
.0.0.1:1025 -> 10.0.2.3:53
[2010-03-18 06:29:53  type=(sys_socket) ip=(127.0.0.1) pid=(648:1092) command=
(svchost.exe) uid=(0) inode=(0) fd=(0) len=(15) call=(11) inode=(0)]  17:  127
.0.0.1:1025 -> 10.0.2.3:53
[2010-03-18 06:29:53  type=(sys_socket) ip=(127.0.0.1) pid=(648:1092) command=
(svchost.exe) uid=(0) inode=(0) fd=(0) len=(15) call=(12) inode=(0)]  17:  127
.0.0.1:1025 -> 10.0.2.3:53
[2010-03-18 06:29:53  type=(sys_socket) ip=(127.0.0.1) pid=(648:1024) command=
(svchost.exe) uid=(0) inode=(0) fd=(0) len=(15) call=(11) inode=(0)]  17:  127
.0.0.1:123 -> 207.46.197.32:123
[2010-03-18 06:29:56  type=(sys_socket) ip=(127.0.0.1) pid=(648:1024) command=
(svchost.exe) uid=(0) inode=(0) fd=(0) len=(15) call=(11) inode=(0)]  17:  127
.0.0.1:123 -> 207.46.197.32:123
[2010-03-18 06:29:56  type=(sys_socket) ip=(127.0.0.1) pid=(648:1024) command=
(svchost.exe) uid=(0) inode=(0) fd=(0) len=(15) call=(12) inode=(0)]  17:  127
.0.0.1:123 -> 207.46.197.32:123
[2010-03-18 06:30:03  type=(sys_socket) ip=(127.0.0.1) pid=(648:1024) command=
(svchost.exe) uid=(0) inode=(0) fd=(0) len=(15) call=(11) inode=(0)]  17:  127
.0.0.1:1028 -> 239.255.255.250:1900
```

In the output shown above, each line represents a captured system activity during the Windows guest OS starts:

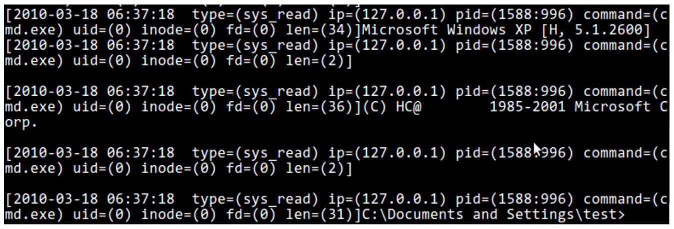
- The first field is the time stamp.
- The second field is the syscall type. Currently *sbk_dialog* supports three types of syscall: they are *sys_open*, *sys_read* and *sys_socket*. The last two types are most common, corresponding to console/process activities and network activities.
- *ip* field indicates the IP address of the honeypot. In most cases, this is the same address you provided to *sbk_ip* option.
- The fourth field contains the pid relation, the first number in the parenthesis is the PID of process that triggers the syscall and the second one is the parent process' PID.
- The command field contains the process name, and uid, inode, fd is not used.
- *call* field only appears when syscall type is *sys_socket*, it indicates the socket operation type:  2=bind, 3=connect, 4=listen, 5=accept, 11=sendto, 12=recfrom, 16=sendmsg and 17=recvmsg.
- The first number after the end bracket is the protocol type: 0=IP, 6=TCP and 17=UDP.
- And the last field is the IP:port pair.

Then we open a command line window in the Windows guest OS (Start->Run->cmd.exe):



```
Microsoft Windows XP [版本 5.1.2600]
(C) 版权所有 1985-2001 Microsoft Corp.

C:\Documents and Settings\test>
```

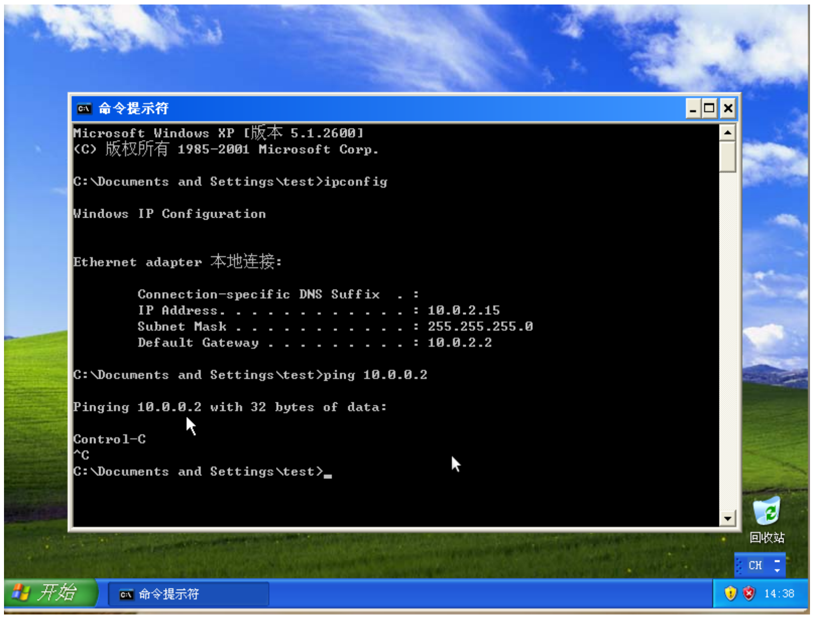
For this operation, Qebek captured following activities,



```
[2010-03-18 06:37:18  type=(sys_read) ip=(127.0.0.1) pid=(1588:996) command=(c
md.exe) uid=(0) inode=(0) fd=(0) len=(34)]Microsoft Windows XP [H, 5.1.2600]
[2010-03-18 06:37:18  type=(sys_read) ip=(127.0.0.1) pid=(1588:996) command=(c
md.exe) uid=(0) inode=(0) fd=(0) len=(2)]

[2010-03-18 06:37:18  type=(sys_read) ip=(127.0.0.1) pid=(1588:996) command=(c
md.exe) uid=(0) inode=(0) fd=(0) len=(36)](C) HC@        1985-2001 Microsoft C
orp.

[2010-03-18 06:37:18  type=(sys_read) ip=(127.0.0.1) pid=(1588:996) command=(c
md.exe) uid=(0) inode=(0) fd=(0) len=(2)]

[2010-03-18 06:37:18  type=(sys_read) ip=(127.0.0.1) pid=(1588:996) command=(c
md.exe) uid=(0) inode=(0) fd=(0) len=(31)]C:\Documents and Settings\test>
```

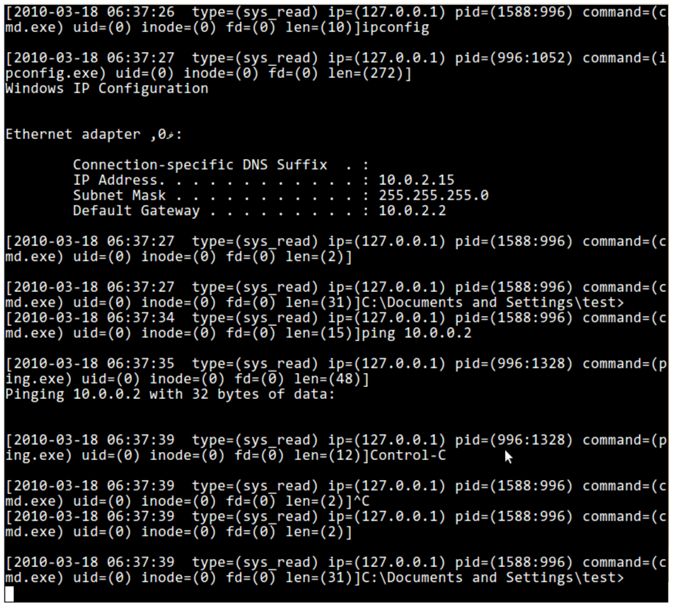
The content after the end bracket corresponds to the console input/output.

Then we try some common commands:



```
Microsoft Windows XP [版本 5.1.2600]
(C) 版权所有 1985-2001 Microsoft Corp.

C:\Documents and Settings\test>ipconfig

Windows IP Configuration


Ethernet adapter 本地连接:

        Connection-specific DNS Suffix  . :
        IP Address. . . . . . . . . . . . : 10.0.2.15
        Subnet Mask . . . . . . . . . . . : 255.255.255.0
        Default Gateway . . . . . . . . . : 10.0.2.2

C:\Documents and Settings\test>ping 10.0.0.2

Pinging 10.0.0.2 with 32 bytes of data:

Control-C
^C
C:\Documents and Settings\test>_
```

And the captured activities are:



```
[2010-03-18 06:37:26  type=(sys_read) ip=(127.0.0.1) pid=(1588:996) command=(c
md.exe) uid=(0) inode=(0) fd=(0) len=(10)]ipconfig

[2010-03-18 06:37:27  type=(sys_read) ip=(127.0.0.1) pid=(996:1052) command=(i
pconfig.exe) uid=(0) inode=(0) fd=(0) len=(272)]
Windows IP Configuration


Ethernet adapter ,0ؤ:

        Connection-specific DNS Suffix  . :
        IP Address. . . . . . . . . . . . : 10.0.2.15
        Subnet Mask . . . . . . . . . . . : 255.255.255.0
        Default Gateway . . . . . . . . . : 10.0.2.2

[2010-03-18 06:37:27  type=(sys_read) ip=(127.0.0.1) pid=(1588:996) command=(c
md.exe) uid=(0) inode=(0) fd=(0) len=(2)]

[2010-03-18 06:37:27  type=(sys_read) ip=(127.0.0.1) pid=(1588:996) command=(c
md.exe) uid=(0) inode=(0) fd=(0) len=(31)]C:\Documents and Settings\test>
[2010-03-18 06:37:34  type=(sys_read) ip=(127.0.0.1) pid=(1588:996) command=(c
md.exe) uid=(0) inode=(0) fd=(0) len=(15)]ping 10.0.0.2

[2010-03-18 06:37:35  type=(sys_read) ip=(127.0.0.1) pid=(996:1328) command=(p
ing.exe) uid=(0) inode=(0) fd=(0) len=(48)]
Pinging 10.0.0.2 with 32 bytes of data:


[2010-03-18 06:37:39  type=(sys_read) ip=(127.0.0.1) pid=(996:1328) command=(p
ing.exe) uid=(0) inode=(0) fd=(0) len=(12)]Control-C

[2010-03-18 06:37:39  type=(sys_read) ip=(127.0.0.1) pid=(1588:996) command=(c
md.exe) uid=(0) inode=(0) fd=(0) len=(2)]^C
[2010-03-18 06:37:39  type=(sys_read) ip=(127.0.0.1) pid=(1588:996) command=(c
md.exe) uid=(0) inode=(0) fd=(0) len=(2)]

[2010-03-18 06:37:39  type=(sys_read) ip=(127.0.0.1) pid=(1588:996) command=(c
md.exe) uid=(0) inode=(0) fd=(0) len=(31)]C:\Documents and Settings\test>
```

**Honeywall setup (*optional*)**

If preferred, you could also run Honeywall as a virtual machine on the same host, thus build a virtualized honeynet system. As hardware virtualization extension (AMD SVM or Intel VT) are common now, we use KVM to achieve better performance.

Firstly, we create a new virtual disk (at least 8GB):

```
qemu-img create -f qcow2 roo.img 8G
```

Then we install Roo using the latest cdrom image (https://projects.honeynet.org/honeywall/raw-attachment/wiki/WikiStart/roo-1.4.hw-20090425114542.iso). Since Roo requires three NIC, we use following script to launch Roo:

```
#! /bin/sh
tunctl -b -u root -t tap0
brctl addif br0 tap0
ifconfig tap0 0.0.0.0 promisc up

kvm -localtime \ #use local time instead of utc
    -m 512 \ #512MB memory
    -had roo.img \ #formal created disk image
    -cdrom roo-1.4.hw-20090425114542.iso \ #roo cdrom
    -net nic,vlan=0,model=pcnet \ #NIC1, connected to Internet
    -net tap,vlan=0,ifname=tap0,script=no,downscript=no \ #use tap0 to attach to br0
    -net nic,vlan=1,model=pcnet \ #NIC2, connected to honeypot
    -net socket,vlan=1,listen=127.0.0.1:8010 \ #use socket so another VM can connect to this NIC
    -net nic,vlan=2,model=pcnet \ #NIC3, management interface
    -net                                    user,vlan=2,net=10.0.0.0/24,hostfwd=tcp:127.0.0.1:30336-
10.0.0.15:3306,hostfwd=tcp:127.0.0.1:30022-10.0.0.15:22,hostfwd=tcp:127.0.0.1:30443-10.0.0.15:443
#forward to Honeywall's MySQL DB, SSHD and Walleye

ifconfig tap0 down
brctl delif br0 tap0
tunctl -d tap0
```

Here we set the management network use KVM's user mode network, with guest network `10.0.0.1/8` (you could choose any network address you like) and guest IP `10.0.0.15`. Therefore, when configuring the management network, the IP would be `10.0.0.15`, with mask `255.255.255.0`, gateway `10.0.0.2` and DNS server `10.0.0.3`. To access Roo from host, we also add three host port forwarding rules: `30336` to `3306` (mysqld), `30022` to `22` (sshd) and `30443` to `443` (httpd). After Honeywall is set up and configured, you should be able to connect to it through

```
ssh roo@127.0,0.1 -p 30022
```

and visit Walleye through

```
https://127.0.0.1:30443
```

The reason we need to connect to Roo's database is because Qebek does not use Sebek's convert UPD channel to submit captured system activities. However, Roo does not permit direct database connection by default. So we need to add permissions manually:

1. Add firewall permission to allow TCP connection to port `3306`;
2. Grant remote access privileges to `roo@10.0.0.2` to modify `hflow` database.

Above we use user mode for the management interface, if you have more than one physic NIC, you could also use bridge and TAP for the management interface.

```
#! /bin/sh
tunctl -b -u root -t tap0
brctl addif br0 tap0
ifconfig tap0 0.0.0.0 promisc up
tunctl -b -u root -t tap1
brctl addif br1 tap1
ifconfig tap1 0.0.0.0 promisc up

kvm -localtime \ #use local time instead of utc
    -m 512 \ #512MB memory
    -had roo.img \ #formal created disk image
    -cdrom roo-1.4.hw-20090425114542.iso \ #roo cdrom
    -net nic,vlan=0,model=pcnet \ #NIC1, connected to Internet
    -net tap,vlan=0,ifname=tap0,script=no,downscript=no \ #use tap0 to attach to br0
    -net nic,vlan=1,model=pcnet \ #NIC2, connected to honeypot
    -net socket,vlan=1,listen=127.0.0.1:8010 \ #use socket so another VM can connect to this NIC
    -net nic,vlan=2,model=pcnet \ #NIC3, management interface
    -net tap,vlan=2,ifname=tap1,script=no,downscript=no #use tap1 to attach to the second bridge

ifconfig tap1 down
brctl delif br1 tap1
tunctl -d tap1
ifconfig tap0 down
brctl delif br0 tap0
tunctl -d tap0
```

The installation and configuration process is similar to installing on bare host or VMware. Also, you need to make Roo's MySQL remotely accessible.

After Roo is set up, we need to change the networking configuration of the honeypot so it goes through Honeywall:

```
#! /bin/sh

qebek -localtime \ #use local time instead of utc
    -snapshot \ #protect the disk image
```

```
        -m 512 \ #512MB memory
        -had winxp.img \ #formal created disk image
        -net nic,vlan=0,macaddr=00:11:22:33:44:55,model=pcnet \ #NIC info
        -net socket,vlan=0,connect=127.0.0.1:8010 \ #use the socket of Roo
        -winxp \ #the guest OS is Windows XP
        -sbk_ip 127.0.0.1 #the honeypot ip address
```

Now you should test if the networking is working correctly. Then, you could modify the Sebekd to read from stdout of Qebek and insert information into Roo's database.

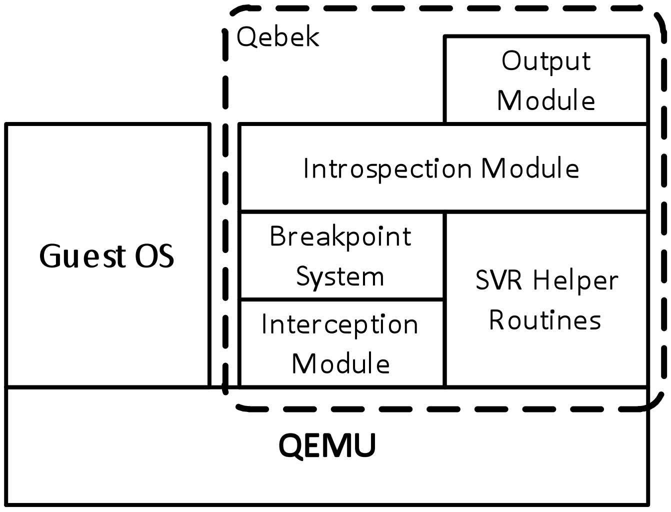
## QEMU

Qebek is built upon QEMU, so we will first give a brief introduction of QEMU before we describe the architecture of Qebek.

QEMU is a generic machine emulator which uses dynamic translation technology to execute binary instruction in the guest OS.  For the reason that it is open-sourced and is able to emulate a whole system and control the execution at instruction level, it is widely used in security oriented analysis systems. From version 0.10, it had switched to use tiny code generator (TCG) to perform the translation instead of the previously used *dyngen*. However, since most of those analysis systems are created in version 0.9 years, and are deeply hooked into the *dyngen* procedures. Therefore, they remain using this version and this is also the version Qebek is based on.

## SYSTEM ARCHITECTURE

Generally, Qebek can be divided into five parts: the interception module, the breakpoint system, the SVR helper routines, the introspection module and the output module.



### The Breakpoint System

The breakpoint system is the backbone of Qebek. Before diving into the details, we'd like to explain why Qebek uses breakpoints to implement the hook. At the time we designed Qebek, most existing virtualization-based honeypot monitoring tool intercept *sysenter* instruction to hook syscall made inside the guest machine, which is clear and simple. In some way, this mechanism is similar to the SSDT hooking used by Sebek Win32 client version.

The problem is, this is not the only way to make syscall. Once the attackers get the control of the OS kernel, they are able to:

- Install a kernel rootkit that directly calls the syscall implementation,
- Register a new syscall and make this handler a new syscall dispatcher,
- Register an interrupt handler to make syscall in user mode.

Any of these tricks will bypass the sysenter hooking used by those systems. Therefore, we decided to use breakpoint to directly hook the syscall implementation. Moreover, once having this system, developers are also able to hook any functions they'd like to, for example, user mode IPC functions or non-exported kernel functions.

The breakpoint system uses a hash table to manage the breakpoints. Each breakpoint has the following attribute:

```
typedef struct qebek_bp_slot {
    ...
    target_ulong breakpoint; //virtual address of the breakpoint
    target_ulong cr3; //!= 0, specify a particular process to break; == 0, break for all process
    target_ulong ebp; //!= 0, specify a particular thread stack to break; == 0, break for all thread
    qebek_cb_func cb_func; //callback function for this breakpoint
    void* user_data; //callback data for this breakpoint
    bool enable; //is the breakpoint enable?
}qebek_bp_slot, *pqbek_bp_slot;
```

The *breakpoint address*, *callback function*, *callback data* and *enable* are quite straightforward to understand. Here we explain other properties. The *cr3* property is used to make the breakpoint work only for particular process, and this is used when the callback data contains process dependent data, e.g. when we collect some data on precall hook and wanted to use these data on postcall hook. The *ebp* property is used to distinguish different thread within a process, i.e. when thread dependent data are stored in callback data.

Currently, the breakpoint system offers two functions to manipulate breakpoints:

```
bool qebek_bp_add(target_ulong address, target_ulong cr3, target_ulong ebp,
        qebek_cb_func cb_func, void* user_data);
bool qebek_bp_remove(target_ulong address, target_ulong cr3);
```

## The Interception Module

The interception module is one of the two attach points to QEMU. The most important duty for this module is to call the *qebek_bp_check* function when the instruction pointer (EIP in i386 or X86) of the guest system changes. In current implementation, since we only care about function calls, therefore, we only intercept instructions that cause control flow **jump** to a new place. This is done by hook the *OPPROTO op_jmp_T0* function in QEMU.

We separate the breakpoint system and the interception so the breakpoint system and upper introspection module can be reused when porting to a new VMM, e.g. Xen.

## The SVR Helper Routines

The System View Reconstruction helper routines are the second attach point to QEMU. Helper routines are provided to perform some common tasks like read data from given virtual address. And these routines also hide the details of address translation process, which is VMM-dependent. The real SVR procedures are implemented by introspection modules. In current Qebek version, the following routines are included:

```
bool qebek_read_ulong(CPUX86State *env, target_ulong address, uint32_t *value);
bool qebek_read_uword(CPUX86State *env, target_ulong address, uint16_t *value);
bool qebek_read_byte(CPUX86State *env, target_ulong address, uint8_t *value);
bool qebek_read_raw(CPUX86State *env, target_ulong address, uint8_t* buffer, int len);
```

### The Introspection Module

This module contains the re-implementation of most functionalities of Sebek. At current stage, the introspection module monitors the same events as Sebek does: the console keystrokes, process creation and network activities. The most difficult stuff here is to reconstruct the system view from the raw data. Fortunately, as many of the operating system functions Sebek hooks are undocumented, the original Sebek code already contains required data structures and access methods.

However, the network monitor is completely different from Sebek in current version. To capture network activities, Sebek uses TDI hook to intercept the I/O Request Packet (IRP) handler of the TCP/IP driver. Since Qebek does not have a driver lookup mechanism, we are not able to locate the TCP/IP driver object as Sebek does. As a result, the network monitor is implemented by intercept the NtDeviceIoControlFile syscall, through which the whole networking I/O commands are handled, except the creation and closing of the socket. This network monitor works pretty well, but has a limitation. That is, some of the listening sockets are open within the kernel, e.g. 445 for SMB over TCP/IP, 137,138,139 for NetBIOS over TCP/IP, meaning that we cannot monitor these ports from Qebek in the same manner used for other network ports. . We haven't figured out any solution for overcome this problem, ideas or suggestions are welcome.

### The Ouput Module

After capturing the activities inside the honeypot, the next step is to output the information to analysis systems. Another advantage of Qebek in invisibility compare to Sebek is that the information is captured at emulator level, which makes it possible to use more stealth mechanism to transfer the captured information. So, instead of creating a convert UDP channel as Sebek does, we could now directly log the information into database running on the host OS, or use the management network interface to transfer it to honeywall, making the communication completely invisible to attackers.

As Qebek is designed to be a potential successor of Sebek, it would be much better if we could make use of existing analysis tools in the GenIII honeynet architecture. For this purpose, the ouput module will format the captured information into Sebek datagram format and output to the standard ouput, which is the same as *sbk_extract* does.

## ADDING YOUR OWN HOOK

To help potential developers to add their own hooks, for monitoring other attackers' activities they are interested, here we generally describe how to hook a function in Qebek. Take *NtReadFile* as a sample.

The first step is to find the function address to add the breakpoint. This address can be hardcoded into the program, but in Qebek, this is done by using SVR to find the address inside the SSDT, giving the corresponding syscall number (see *qebek_hook_syscall* for detail):

3. Get the KTHREAD structure pointer from the well-known address(0xffdff124, i.e. fs:0x124);

```
pkthread = 0xffdff124;
qebek_read_ulong(env, pkthread, &kthread);
```

4. Get the SDT structure pointer from KTHREAD;

```
psdt = kthread + 0xe0;
qebek_read_ulong(env, psdt, &sdt);
```

5. Get the SSDT structure pointer from SDT;

```
pssdt = sdt;
qebek_read_ulong(env, pssdt, &ssdt);
```

6. Get the function address of NtReadFile.

```
index_NtReadFile = 0x0b7; //for Windows XP
qebek_read_ulong(env, ssdt + index_NtReadFile * 4, &NtReadFile);
```

The second step is to install a precall hook, which is generally process and thread independent.

```
qebek_bp_add(NtReadFile, //breakpoint address
        0, //cr3
        0, //ebp
        preNtReadFile, //callback function
        NULL); //callback data
```

The callback function need to accept two arguments, the first one points to the structure which contains the current CPU states (currently CPUX86State is used, which is not so VMM-independent) and the second one points to the callback data.

```
typedef void (*qebek_cb_func)(CPUX86State *env, void* user_data);
```

One tip for precall procedure is that the input arguments are not guaranteed to be preserved after the target function is executed, so these data should be collected in the precall procedure and passed to the postcall procedure through callback data.

The most significant difference between hooking a function in Qebek and hooking a function inside the OS is the *postcall* procedure. While hooking functions inside the OS, most hooking mechanisms will insert the callback function into the original invocation chain. This means the original function is invoked by the callback function, therefore, when the original function returns, the control flow returns to the callback function, making the postcall procedure convenient to implement. But in Qebek, in callback function is executed at the emulator layer, hence we have to manually add hooks for postcall procedure:

```
... //perform the precall procedure
qebek_read_ulong(env, env->regs[R_ESP], &ret_addr); //get the return address from stack
qebek_bp_add(ret_addr, env->cr[3], env->regs[R_EBP],
        postNtReadFile, pReadData); //install the postcall hook
```

## SUMMARY & CONCLUSIONS

In this paper we present Qebek, a virtualization layer monitor for HI honeypot as a successor of Sebek. We show how to use this tool to build a virtual honeypot and leverage existing data analyzing back-end. To help people involve the development of this open sourced tool, we also introduce the architecture of Qebek and show how to add a new hook.

We recommend you to try this tool and if possible, give us some feedback by submitting bugs at our project website (https://projects.honeynet.org/sebek/, account registration is required) or write emails to us or to our mailing list (honeywall@public.honeynet.org). We also strongly recommend that you get involved in the development of Qebek if you are interested in the fantastic binary level of the OS and VMM.

## FUTURE WORK

Qebek is still in an early stage and many features are still to be implemented. First of all, the supported OS type needs to be extended: so far, the only tested OS is Microsoft Windows XP SP2. We need to improve the SVR mechanism to support more Windows family OS and to re-implement the Linux version of Sebek client. Secondly, we want to implement a module lister to locate the address of user mode module (DLL) and kernel mode module (driver object). Previous work uses an inside-VM module to extract this information and transfer it to the monitor, but with SVR, we believe this could be done completely outside the VM. Once we have this module, we are able to rewrite the network monitor to use TDI hook. Finally, as a honeypot, we want to improve the performance of Qebek to reduce the detection surface.

Current Qebek is built upon QEMU 0.9.2 which is pretty old (the latest version is 0.13.0). From version 0.10.0, QEMU began using a new translation technique called Tiny Code Generator (TCG) that does not require GCC 3.4. So another improvement would be porting Qebek to newer version of QEMU. Although TCG is quite different from older version, as Qebek has a very small footprint on QEMU, we anticipate this won't require too much work.

## ACKNOWLEDGEMENTS

We would like to thank the following people:

- Christian Seifert of  the Honeynet Project
- Angelo Dell'Aera of  the Honeynet Project
- Jeffery Stutzman of the Honeynet Project
- Jamie Riden of the Honeynet Project
- Hugo González of the Honeynet Project
- Diego Gonzalez of the Honeynet Project
- Adel Karimi of the Honeynet Project
- Camilo Viecco of the Honeynet Project
- Markus Koetter of the Honeynet Project

## FURTHER READING

For the interested reader, here are some related works on monitoring guest OS outside the virtual machine:

**Projects Open Sourced**:

- Argos: a QEMU based HIDS that captures zero-day attack by using dynamic taint analysis, open sourced under GPL earlier than 2006: http://www.few.vu.nl/argos/

- Ether: a malware analysis tool aimed at avoiding being detected by malware by leveraging hardware virtualization extensions, open sourced under GPL in April 2009: http://ether.gtisc.gatech.edu/
- TEMU: the dynamic analysis component of the BitBlaze platform, also QEMU based, open sourced under LGPL in about November 2009: http://bitblaze.cs.berkeley.edu/temu.html

**Projects with Publication Only**:

- T. Garfinkel, and M. Rosenblum. A Virtual Machine Introspection Based Architecture for Intrusion Detection. Proc. of the 2003 Network and Distributed System Security Symposium, Feb. 2003.
- Ulrich Bayer, Christopher Kruegel, Engin Kirda. TTAnalyze: A Tool for Analyzing Malware, EICAR, 2005.
- Nguyen Anh Quynh, and Yoshiyasu Takefuji, Towards an Invisible Honeypot Monitoring System, Lecture Notes in Computer Science, Vol 4058, pp 111-122, 2006.
- Xuxian Jiang, Xinyuan Wang, Dongyan Xu. Stealthy Malware Detection through VMM-Based “Out-of-the-Box” Semantic View Reconstruction. In Proceedings of CCS 2007.
- Xuxian Jiang, Xinyuan Wang. “Out-of-the-box” Monitoring of VM-based High-Interaction Honeypots, In Proceedings of RAID 2007.
- Bryan D. Payne, Martim Carbone, Monirul Sharif, Wenke Lee. Lares: An Architecture for Secure Active Monitoring Using Virtualization. In Proceedings of 2008 IEEE Symposium on Security and Privacy. 2008.

For those who are interested in Windows networking, Helmut Petritsch’s mater thesis is a very helpful reference:

- Helmut Petritsch. Understanding and Replaying Network Traffic in Windows XP for Dynamic Malware Analysis, Master thesis, 2007.

For more information about the networking configuration of QEMU, here is a good reference:

- QEMU Networking, http://people.gnome.org/~markmc/qemu-networking.html

## ABOUT THE AUTHORS

***Chengyu Song*** is a Masters student at Peking University. He has been a member of the Chinese chapter of the Honeynet Project since 2005 and has been a Full Member of the Honeynet Project since 2007. He is the current maintainer of Sebek Win32 client and worked on Qebek during Google Summer of Code 2009.

***Brian Hay*** is a faculty member and Lab Director at the Advanced System Security Education, Research, and Training Center at the University of Alaska Fairbanks.  He served as the mentor on Qebek during Google Summer of Code 2009.

***Jianwei Zhuge*** is the leader of the Honeynet Project Chinese chapter and worked as the co-mentor on Qebek during Google Summer of Code 2009.