

Disclose Three Bugs in xhyve

Jul 21, 2022

Disclose Three Bugs in xhyve

Foreword

Earlier this year, I stumbled upon an interesting [advisory](#) from ZDI, which was yet to be published back then.

It said that a vulnerability exists in [xhyve](#), and was scheduled to be disclosed in a month.

So I decided to do a code review of xhyve, to see that if I could locate the bug before the disclosure.

In the end, I find 4 bugs, all of them are technically 1-Day, and have been already fixed in the upstream or downstream projects.

Unfortunately, ZDI did not publish this advisory until recently, so I was not sure if any one of these 4 bugs was the one mentioned in the advisory.

Turns out, I did find it, and it is a bug in `./src/pci_e82545.c:1410`, the e1000 virtual device code and has been found and fixed in xhyve's upstream project [bhyve](#).

Since ZDI still gives CVE-2022-35867 to this bug, I guess it makes sense to do a responsible disclosure of the other 3 bugs here.

All of the other 3 bugs were found by GitHub Security Lab in [HyperKit](#), a project based on xhyve, and disclosed in [GHSL-2021-054_057](#) advisory.

HyperKit is a toolkit for embedding hypervisor capabilities in your application. It includes a complete hypervisor, based on xhyve/bhyve, which is optimized for lightweight virtual machines and container deployment. It is designed to be interfaced with higher-level components such as the VPNKit and DataKit.

Since the security patches in HyperKit were not upstreamed, the latest version of xhyve still has these 4 bugs.

pci_virtio_rnd vc_cfgread NULL Pointer Dereference

Virtio devices use `struct virtio_consts` to register handler functions for device IO.

```
struct virtio_consts {
    /* name of driver (for diagnostics) */
    const char *vc_name;
    /* number of virtual queues */
    int vc_nvq;
    /* size of dev-specific config regs */
    size_t vc_cfgsize;
    /* called on virtual device reset */
    void (*vc_reset)(void *);
    /* called on QNOTIFY if no VQ notify */
    void (*vc_qnotify)(void *, struct vqueue_info *);
    /* called to read config regs */
    int (*vc_cfgread)(void *, int, int, uint32_t *);
    /* called to write config regs */
    int (*vc_cfgwrite)(void *, int, int, uint32_t);
    /* called to apply negotiated features */
    void (*vc_apply_features)(void *, uint64_t);
    /* hypervisor-provided capabilities */
    uint64_t vc_hv_caps;
};
```

In the case of `pci_virtio_rnd`, a `virtio entropy device emulation`, `vc_cfgread` is initialized to `NULL`.

```
static struct virtio_consts vtrnd_vi_consts = {
    "vtrnd", /* our name */
    1, /* we support 1 virtqueue */
    0, /* config reg size */
    pci_vtrnd_reset, /* reset */
    pci_vtrnd_notify, /* device-wide qnotify */
    NULL, /* read virtio config */
    NULL, /* write virtio config */
    NULL, /* apply negotiated features */
};
```

```
0, /* our capabilities */  
};
```

Yet, in `vi_pci_read()`, the existence of the `vc_cfgread` handler is not checked.

```
error = (*vc->vc_cfgread) (DEV_SOFTC(vs), ((int) newoff), size, &value);
```

So, when a malicious guest triggers the `vc_cfgread` IO, a null pointer dereference will crash the guest vm process.

pci_virtio_rnd vc_cfgwrite NULL Pointer Dereference

Similarly, in `vi_pci_write()`, the existence of the `vc_cfgwrite` handler is also not checked.

```
error = (*vc->vc_cfgwrite) (DEV_SOFTC(vs), ((int) newoff), size,  
    ((uint32_t) value));
```

So, when a malicious guest triggers the `vc_cfgwrite` IO, a null pointer dereference will crash the guest vm process.

pci_virtio_rnd Uninitialized Stack Buffer Usage

In the `pci_vtrnd_notify()` handler function, the return value of `vq_getchain()` is not checked, so if it returns due to some failed check,

`iov` stack variable will be left uninitialized, and `iov.iov_base` will be used as the buffer to store random data read from `sc->vrsc_fd`.

```
static void  
pci_vtrnd_notify(void *vsc, struct vqueue_info *vq)  
{  
    struct iovec iov;  
    ...  
    vq_getchain(vq, &idx, &iov, 1, NULL);  
  
    len = (int) read(sc->vrsc_fd, iov.iov_base, iov.iov_len);  
    ...  
}
```

Theoretically, this bug is exploitable. One possible way is to leave an address of the length of some buffer on the stack as `iov.iov_base`.

Since the data from `sc->vrsc_fd` is typically very large, we can overwrite the length with a larger value and achieve an OOB read/write primitive.

However, during my brief test, the stack position of `iov` seems to be too shallow to avoid being overwritten before `pci_vtrnd_notify()` is actually reached.

Afterword

At the time of writing, July 21, 2022, all of these 4 bugs have not been fixed by xhyve, and the project's last commit was back on Oct 2, 2021.

Besides, none of the [issues](#), open or closed, mentions any of these 4 bugs.

According to my understanding, [Homebrew](#) and [MacPorts](#) do not provide additional security patch themselves,

so both builds from their repository are also affected.

Sometimes, maintainers of some small open source projects may not be able to do a timely fix.



Until then, xhyve users may remove the `pci_virtio_rnd` and `e1000` device from their vm.

Reference

1. <https://github.com/machyve/xhyve>
2. <https://github.com/moby/hyperkit>
3. <https://www.zerodayinitiative.com/advisories/ZDI-22-949/>
4. https://securitylab.github.com/advisories/GHSL-2021-054_057-moby-hyperkit/
5. <https://www.freebsd.org/security/advisories/FreeBSD-SA-19:21.bhyve.asc>

awxylitol

awxylitol
your-email@example.com

 [awxylitol](#)
 [awxylitol](#)

hunt quality bugs for fun and profit.