LVFS Host Security ID (HSI) and Silicon-Based Core Security

Linux Secure Launch - TrenchBoot Summit 2021

Piotr Król

3MDEB

whoami

Piotr Król
3mdeb Founder

- OSF and OSHW promoter interested in Trusted Computing
- Conference speaker and organizer
- TrenchBoot Steering Committee Core Member
- 13yrs in business
- 7yrs in Open Source Firmware and Trusted Computing integration
- C-level positions in

3MDEB LPN PLANT VITRO

About us



- coreboot licensed service providers since 2016 and leadership participants
- UEFI Adopters since 2018
- Yocto Participants and Embedded Linux experts since 2019
- Official consultants for Linux Foundation fwupd/LVFS project
- IBM OpenPOWER Foundation members

Kudos

- Michał Kopeć, Firmware Engineer @ 3mdeb, as main contributor this presentation

Agenda

- Introduction
- fwupd
- HSI
- Example HSI ID
- HSI plugin analysis
- Adding a new test to HSI
- Code structure
- Platform specific features
- D-RTM
- Q&A

Problem statement

- There is no easy way to determine level of platform security
- What we can do today
  - obtain access to NDA documentation
  - obtain access to source code or skill to perform RE tasks
  - get deep technical knowledge of hardware, firmware, bootloader, hypervisor, OS and system software
  - explore infinite space of new frameworks
- CHIPSEC was one of the first and most successful frameworks to detect most common platform security issues
  - of course not without own issues: UEFI Secure Boot had to be disabled, some tests required unsigned Linux kernel modules
  - Intel-centric
  - not all modules are OSS
- Now we are in era of fwupd/LVFS

fwupd/LVFS

- Linux Foundation project maintained by Richard Hughes
- Written in C with LGPL v2.1 license
- Initial release: 2015
- `fwupd` is a firmware update application
- LVFS is a update providing web service
- It grew over the years as biggest database of firmware updates
- Through LF project was able to establish relation with major IBV, OEMs and ODMs
- Already capable of enforcing some policies regarding firmware update quality as well as grading firmware based on detected features

fwupd/LVFS HSI

```
[hughsie@localhost ~]$ fwupdmgr security --force
Host Security ID: HSI:1 (v1.5.0)

HSI-1
✔ CSME manufacturing mode:       Locked
✔ CSME override:                 Locked
✔ CSME v0:11.8.77.3664:          Valid
✔ Intel DCI debugger:            Disabled
✔ SPI BIOS region:               Locked
✔ SPI lock:                      Enabled
✔ SPI write:                     Disabled
✔ TPM v2.0:                      Found

HSI-2
✔ Intel BootGuard:               Enabled
✔ Intel BootGuard ACM protected: Valid
✔ Intel BootGuard OTP fuse:      Valid
✔ Intel BootGuard verified boot: Valid
✔ Intel DCI debugger:            Locked
✔ TPM PCR0 reconstruction:       Valid
✘ IOMMU:                         Not found

HSI-3
✔ Intel BootGuard error policy:  Valid
✘ Intel CET Enabled:             Not supported
```

- since fwupd already possess vast information about firmware security features
  - Intel Boot Guard enablement status
  - SPI flash protection
  - TPM 2.0 presence
- In version v1.5.0 fwupd introduced **HSI** (Host Security ID), which gives ability to easily determine platform security.

https://blogs.gnome.org/hughsie/2020/10/26/new-fwupd-1-5-0-release/

HSI Specification

- HSI was turned into official specification authored by:
  - Richard Hughes
  - Mario Limonciello
  - Alex Bazhaniuk
  - Alex Matrosov
- With essential goal of providing easy-to-understand information to people buying hardware
- https://github.com/fwupd/fwupd/blob/main/docs/hsi.md

HSI levels

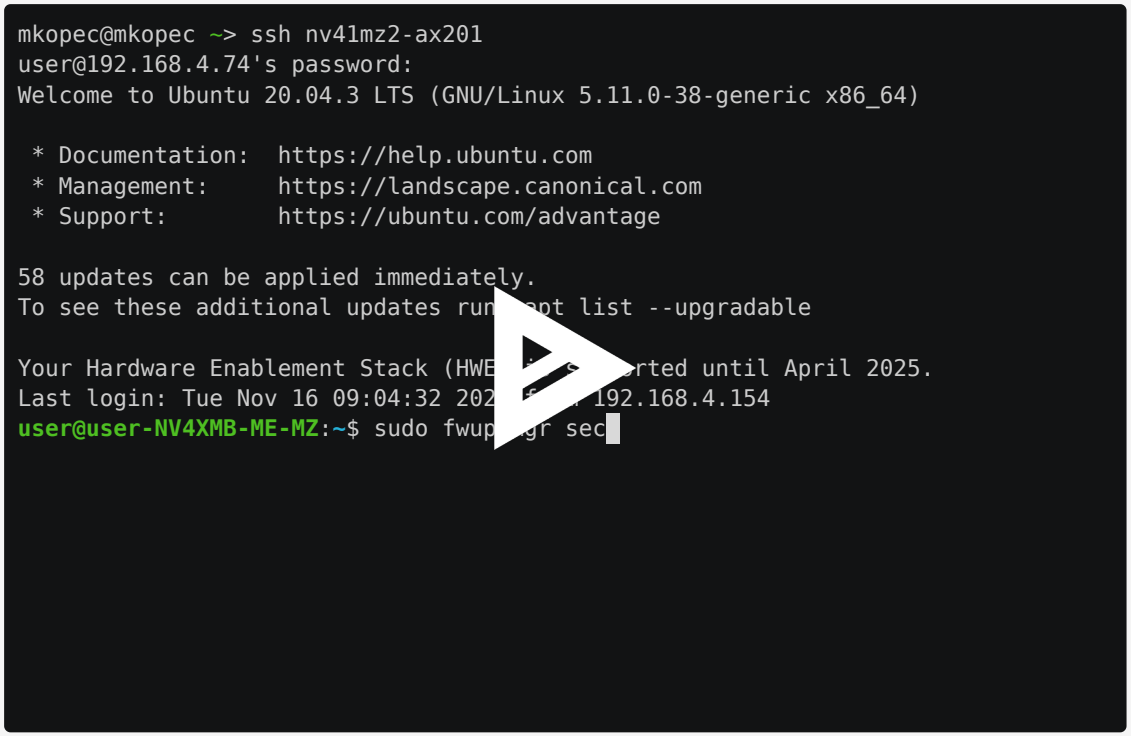
- **`HSI:0` - Insecure:** None or few firmware protections
  - The lowest security level with little or no detected firmware protections. This is the default security level if no tests can be run or some tests in the next security level have failed.
- **`HSI:1` - Critical:** Most basic protections, may or may not be sufficient to protect against remote attacks
  - This security level corresponds to the most basic of security protections considered essential by security professionals. Any failures at this level would have critical security impact and could likely be used to compromise the system firmware without physical access.
- **`HSI:2` - Risky:** Protects against more difficult attacks
  - This security level corresponds to firmware security issues that pose a theoretical concern or where any exploit would be difficult or impractical to use. At this level various technologies may be employed to protect the boot process from modification by an attacker with local access to the machine.

HSI levels

- **`HSI:3` - Protected:** Few issues, high security
  - This security level corresponds to out-of-band protection of the system firmware perhaps including recovery.
- **`HSI:4` - Secure:** No issues, several layers of protection for firmware
  - The system is corresponding several kind of encryption and execution protection for the system firmware.
- **`HSI:5` - Secure Proven:** Out-of-band attestation of firmware, no tests implemented at this moment.
  - This security level corresponds to out-of-band attestation of the system firmware. There are currently no tests implemented for HSI:5 and so this security level cannot yet be obtained.

Example HSI ID

- LVFS does not have to contain firmware update to be able to obtain HSI ID
- The HSI can be determined using `fwupdmgr`:



HSI plugin analysis

- Let's take a look at an example plugin that provides HSI security attributes
- `platform-integrity` is one example of such plugin
  - it checks if the SPI flash access from the OS is disabled, using a (proposed) `sysfs` entry `/sys/class/platform-integrity`
- Plugin integrates `fu_plugin_add_security_attrs` interface, which detects and registers security attributes

```
void
fu_plugin_init_vfuncs(FuPluginVfuncs *vfuncs)
{
    vfuncs->build_hash = FU_BUILD_HASH;
    vfuncs->init = fu_plugin_platform_integrity_init;
    vfuncs->destroy = fu_plugin_platform_integrity_destroy;
    vfuncs->backend_device_added = fu_plugin_platform_integrity_backend_device_added;
    vfuncs->add_security_attrs = fu_plugin_platform_integrity_add_security_attrs;
}
```

HSI plugin analysis

`fu_plugin_add_security_attrs` tests for:

- BIOS Write Enable
- BIOS Lock Enable
- SMM BIOS Write Protect

```
static void
fu_plugin_platform_integrity_add_security_attrs(FuPlugin *plugin, FuSecurityAttrs *attrs)
{
    FuPluginData *priv = fu_plugin_get_data(plugin);

    /* only when the kernel module is available */
    if (priv->sysfs_path == NULL)
        return;

    /* look for the three files in sysfs */
    fu_plugin_add_security_attr_bioswe(plugin, attrs);
    fu_plugin_add_security_attr_ble(plugin, attrs);
    fu_plugin_add_security_attr_smm_bwp(plugin, attrs);
}
```

Adding a new test to HSI

- PR to https://github.com/fwupd/fwupd
- Add a new criteria ID in `libfwupd/fwupd-security-attr-private.h`
- Add a new entry in `docs/hsi.md`
    - Specify HSI level and impact
- Implement or extend `fu_plugin_add_security_attrs` function in an existing or new plugin

Adding a new test to HSI

- Example: Plugin that checks CPU vulnerability mitigations via `sysfs`
- We want to check if the the Meltdown vulnerability mitigations are enabled
- Start by creating a new plugin - in this case based on the `platform-integrity` plugin
- Add a new criteria ID in `libfwupd/fwupd-security-attr-private.h`

```
/**
 * FWUPD_SECURITY_ATTR_ID_MELTDOWN:
 *
 * Host Security ID attribute for the Meltdown vulnerability
 *
 * Since: 1.8.0
 **/
#define FWUPD_SECURITY_ATTR_ID_MELTDOWN "org.fwupd.hsi.vulns.Meltdown"
```

- Also add a name for the test in `src/fu-security-attr.c`

```
    fu_security_attr_get_name(FwupdSecurityAttr *attr)
    [...]
    if (g_strcmp0(appstream_id, FWUPD_SECURITY_ATTR_ID_MELTDOWN) == 0) {
        return g_strdup(_("Meltdown mitigations"));
    }
```

Adding a new test to HSI

- Implement `fu_plugin_add_security_attrs` - read the `sysfs` entry

```
void
fu_plugin_add_security_attrs(FuPlugin *plugin, FuSecurityAttrs *attrs)
{
  [...]
    /* create attr*/
    attr = fwupd_security_attr_new(FWUPD_SECURITY_ATTR_ID_MELTDOWN);
    fwupd_security_attr_set_plugin(attr, fu_plugin_get_name(plugin));
    fwupd_security_attr_set_level(attr, FWUPD_SECURITY_ATTR_LEVEL_CRITICAL);
    fu_security_attrs_append(attrs, attr);
    /* load file */
    if (!g_file_get_contents(fn, &buf, &bufsz, &error_local)) {
        g_warning("could not open %s: %s", fn, error_local->message);
        fwupd_security_attr_set_result(attr, FWUPD_SECURITY_ATTR_RESULT_NOT_VALID);
        return;
    }
    /* failure */
    if (g_strstr_len(buf, bufsz, "Vulnerable") != NULL) {
        fwupd_security_attr_set_result(attr, FWUPD_SECURITY_ATTR_RESULT_NOT_ENABLED);
        return;
    }
    /* success */
    fwupd_security_attr_add_flag(attr, FWUPD_SECURITY_ATTR_FLAG_SUCCESS);
    fwupd_security_attr_set_result(attr, FWUPD_SECURITY_ATTR_RESULT_ENABLED);
}
```

Adding a test to HSI

- The test now appears in the output of `fwupdtool --security`:

```
Host Security ID: HSI:0! (v1.7.0)

HSI-1
✔ Meltdown mitigations:          Enabled
✔ UEFI platform key:             Valid
✘ SPI BIOS region:               Unlocked
✘ SPI lock:                      Disabled
✘ SPI write:                     Enabled
✘ TPM v2.0:                      Not found
[...]
```

- Remember to document the changes - add a description to `docs/hsi.md`

See complete code at: https://github.com/3mdeb/fwupd/compare/3162c85...hsi_plugin_demo

How features are tested

Intel

- BootGuard detection - read from HFSTS registers
- Flash protection features - read from PCI config space
- Intel TME - read from CPUID
- Intel ME - read from the MEI interface, FW version, Flash Descriptor Override status, Manufacturing mode disable

AMD

- AMD TSME - Read from MSR

Other architectures

- Current specification is highly focused on x86 UEFI
- ARM, RISC-V specific features are planned

What about support for...

- Qubes OS: HSI detection via `qubes-fwupd` does not currently work
  - The wrapper script doesn't understand the `security` option at the moment
- coreboot: no protections commonly used with coreboot are detected
  - vboot, chip-based flash protection: these can provide high levels of security if implemented correctly
  - heads

D-RTM

- D-RTM belongs to higher HSI levels
- D-RTM has a number of hardware and firmware requirements:
  - Vendor-specific method of establishing trusted state (`SKINIT` / `GETSEC[SENTER]` etc)
  - DMA protection - IOMMU correctly configured
  - SMM protection
  - TPM 1.2 / 2.0
  - TPM event log

D-RTM and HSI

- We can extend the HSI specification to include D-RTM capabilities by adding the following tests:
  - CPUID detection
  - TPM 1.2 / 2.0 present and functional
  - Intel TXT / CBnT / AMD SKINIT / other vendor specific secure reset solution
  - Presence of TXT / CBnT Authenticated Code Module (for Intel) or SKL (for AMD)
- Then we can check for D-RTM enablement:
  - PCR 17-22 population

Discussion

- Authors of HSI Specification suggest: "To be trusted, this rating information should be distributed in a centralized agnostic database such as the LVFS."
  - maybe it is worth to discuss other models?
- fwupd/LVFS direction
  - it looks like fwupd/LFVS pursue BMC updates, which goes into direction of full-blown OS update
- HSI is limited to verification that can be done purely by software without additional hardware, software or configuration change
  - maybe this is good generic guideline for platform security features detectability
- What HSI level D-RTM checks should be added?
- What about modern OSF boot flow?

Q&A