Project Progress Report

Firmware Module for Infer Static Analyzer

Yousef Elsendiony (YE2914) and Tony Butcher (ab4359)

Experiments and Direction of Project

In this experiment we will develop a module for Infer targeted at finding firmware related failures in C language. Some of these common failures include stack overflows, null dereferencing, memory leaks, unwanted compiler optimizations, and cast exceptions. The midterm research paper included more details regarding why these are common failures in firmware. Our goal will be to find more bugs which traditional static analyzers are unable to discover.

Setup

1- Get Infer

- **Build from Source:** Needed for adding modules and any editing of Infer source.
  1- Clone the Repo https://github.com/facebook/infer.git
  2- Run the build script with clang option (for C/C++ analyzer)
  3- Sudo make install (so it can be used system wide)
- **Use the latest release Binary of Infer:** For initial tests of release Infer.
  1- download + untar https://github.com/facebook/infer/releases/tag/v0.15.0
  2- Link/bind it in Linux (so it can be used system wide)

Tony: After some initial failures, I was able to build Infer with the clang compile option using a VM, Ubuntu 16.04LTS with 12GB RAM and 80GB disk. Using less RAM or disk space when compiling clang resulted in failure. Commands to install:

```
sudo add-apt-repository ppa:avsm/ppa
sudo apt-get update
sudo apt upgrade
sudo apt get git
sudo apt install git libgmp-dev libsqlite3-dev zlib1g-dev libmpfr-dev
libmpfr-doc cmake m4 automake autotools-dev libcap-dev opam pkg-config
gcc
cd Desktop
git clone https://github.com/facebook/infer.git
cd infer
sudo apt install clang
./build-infer.sh clang
sudo make install
```

2- Projects in scope and setup to run infer

- nvme-cli https://github.com/linux-nvme/nvme-cli

```
infer run -- make
```

- bmcWeb https://github.com/openbmc/bmcweb

```
mkdir build
cd build
cmake -DCMAKE_EXPORT_COMPILE_COMMANDS=1 ..
cd ..
```

```
        infer run --compilation-database build/compile_commands.json
```

3 - Challenges:
bmcWeb: The midterm paper project also explained that this project required Boost 1.66 which isn't easily upgradable. Documentation:
(Getting Boost to run) https://stackoverflow.com/questions/12578499/how-to-install-boost-on-ubuntu

```
        /usr/lib/x86_64-linux-gnu
        ./bootstrap.sh --prefix=/usr/local
```

(Checking boost version)

```
        cat /usr/include/boost/version.hpp - 1.65 {this needs to be 1.66}
        cat /usr/local/include/boost/version.hpp - 1.66
```

Evaluation

The criterion would be to introduce sample bugs similar to infer and demonstrate how the modules we build can detect these sample firmware failures. Another criterion for rating the effectiveness of our static analyzer would be to run against multiple firmware projects and examine the catch rate of errors. Lastly, we can compare the results from the above Infer runs against other static analyzers for the C language. A better or more effective analyzer should find a higher number of real bugs found and a lesser number of false bugs or noise.

Static analyzers examined (Tony):

Static analyzer	**Last update**	**Opportunities**	**Results / Comments**
BOON	Feb 2005	**Out of bounds array indexing**	Web pages reports obsolete; not surprising given last update in 2005
Clang Static Analyzer	Aug 2010	dead stores, memory leaks, null pointer deref, and more. Uses source annotations like "nonnull"	Not tested
Csur	Apr 2006	cryptographic protocol-related vulnerabilities	Not tested
Flawfinder	2005	uses of risky functions, buffer overflow (strcpy()), format string	Results for Nvme and bmcweb, seem tailored around the generic use of specific C functions

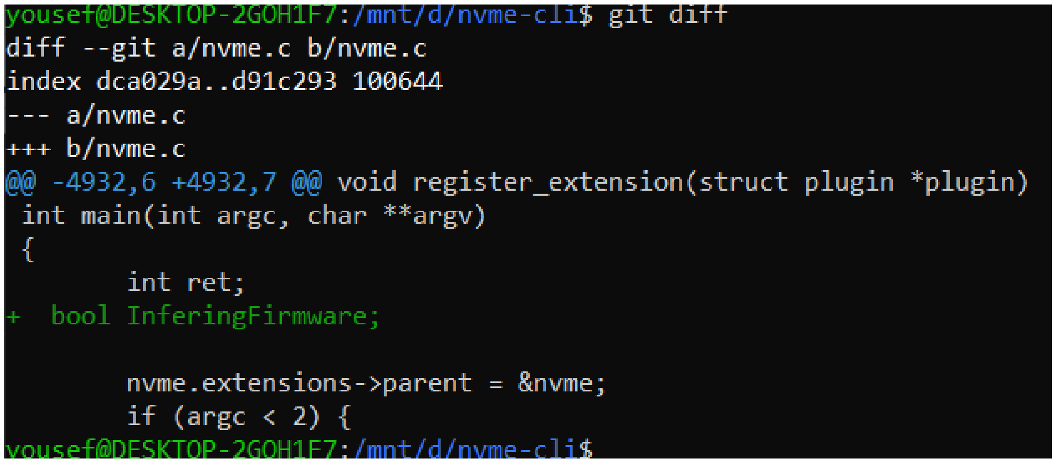
		([v][f]printf()), race conditions (access(), chown(), and mktemp()), shell metacharacters (exec()), and poor random numbers (random()).	such as sprintf and strncpy rather than deep analysis of how the code is constructed, and the use of staticly sized arratys
Cppcheck	Feb 2010	invalid STL usage, overlapping data in sprintf, division by zero, null pointer dereference, unused struct member, passing parameter by value	Tests in nvme and bmcweb yielded no results. Hypothesis is since there are relatively mature projects that the developers conducted static analysis and resolved any findings which would have been found by cppcheck
RATS	Sep 2013	Buffer overflows, race conditions	Nvme: buffer overflow; time of check/time of use findings Bmcweb: mostly static sized arrays similar to flawfinder; one finding for src/kvm_websocket_test.cpp:105: Medium: read "Check buffer boundaries if calling this function in a loop and make sure you are not in danger of writing past the allocated space."
UNO https://github.com/nimble-code/Uno	Mar 2019	Uninitialized variables, null-pointers, and **out-of-bounds array indexing**	Could not install with make, documentation does not discuss installation. Submitted issue to gitweb page and hope to receive feedback given the last update was recent

Splint http://splint.org/source.html	2010	Security vulnerabilities and coding mistakes	Failed to install; errors running make. Not surprising given this was last updated in 2010.

Deliverables

We plan on forking infer and trying to add the firmware modules to our forked repository. If we have success, initiating a pull request with the Infer folks would be a major win. Alternatively, if our bug findings do not generate notable results and our pull request is not accepted, we will provide a detailed analysis of what was done, what we found, any challenges we encountered, and why we did not succeed. Artifacts will be shared via github page provided to the instructors.

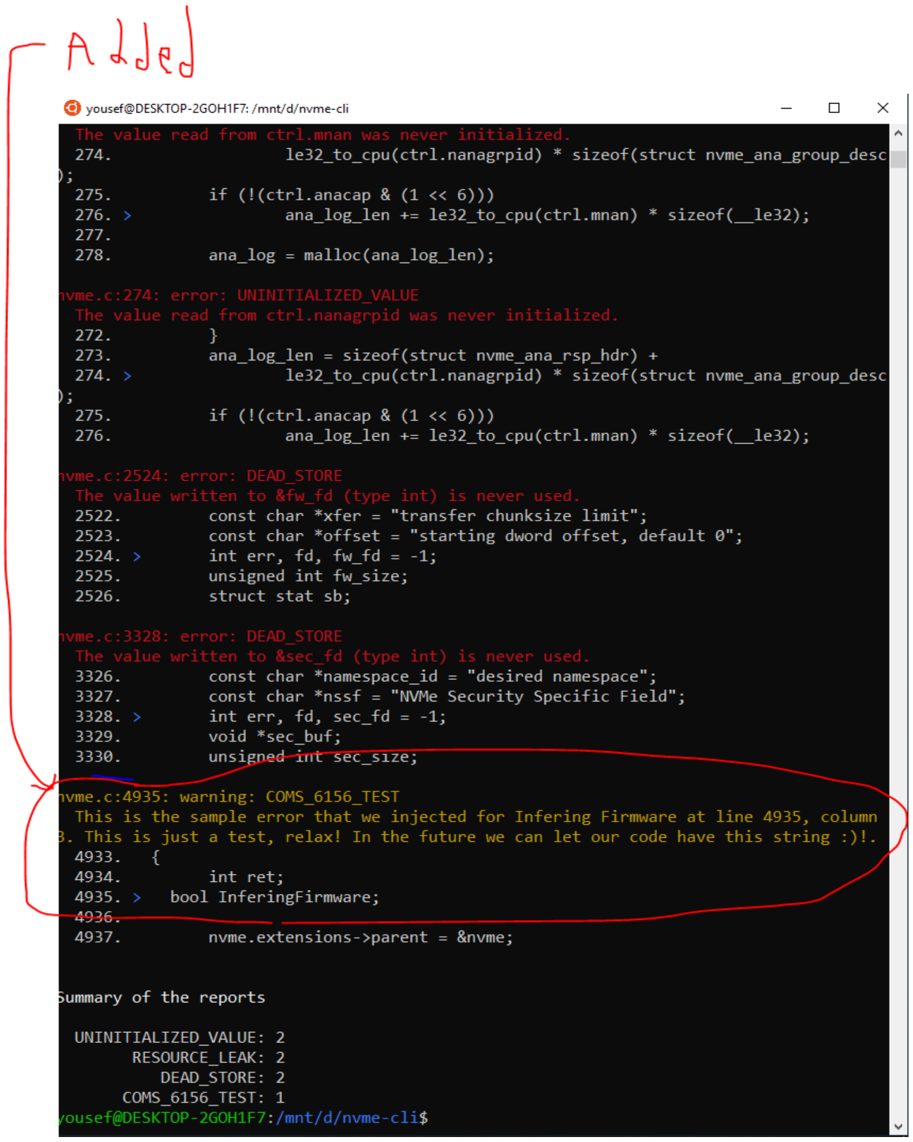
Based on the work done so far, we have decided to focus on using nvme-cli as our app to evaluate Infer against given the challenges in getting bmcWeb setup properly.
Yousef: Successfully added a test module to Infer linting portion, tested against nvme-cli. In the target code below, we can see the injected variable in nvme-cli which could be found by infer.



```
yousef@DESKTOP-2GOH1F7:/mnt/d/nvme-cli$ git diff
diff --git a/nvme.c b/nvme.c
index dca029a..d91c293 100644
--- a/nvme.c
+++ b/nvme.c
@@ -4932,6 +4932,7 @@ void register_extension(struct plugin *plugin)
 int main(int argc, char **argv)
 {
        int ret;
+  bool InferingFirmware;

        nvme.extensions->parent = &nvme;
        if (argc < 2) {
yousef@DESKTOP-2GOH1F7:/mnt/d/nvme-cli$
```

In the image below we can see infer flagging any variables with name “InferingFirmware” which I added to the nvme-cli target project. This is a simple example of adding a module to Infer to search for a string. We leveraged Facebook’s AI framework which allows us to highlight errors from a syntactical standpoint in the C/C++ target project.



```
 The value read from ctrl.mnan was never initialized.
 274.                le32_to_cpu(ctrl.nanagrpid) * sizeof(struct nvme_ana_group_desc
);
 275.        if (!(ctrl.anacap & (1 << 6)))
 276. >              ana_log_len += le32_to_cpu(ctrl.mnan) * sizeof(__le32);
 277.
 278.        ana_log = malloc(ana_log_len);

nvme.c:274: error: UNINITIALIZED_VALUE
 The value read from ctrl.nanagrpid was never initialized.
 272.        }
 273.        ana_log_len = sizeof(struct nvme_ana_rsp_hdr) +
 274. >              le32_to_cpu(ctrl.nanagrpid) * sizeof(struct nvme_ana_group_desc
);
 275.        if (!(ctrl.anacap & (1 << 6)))
 276.              ana_log_len += le32_to_cpu(ctrl.mnan) * sizeof(__le32);

nvme.c:2524: error: DEAD_STORE
 The value written to &fw_fd (type int) is never used.
 2522.        const char *xfer = "transfer chunksize limit";
 2523.        const char *offset = "starting dword offset, default 0";
 2524. >      int err, fd, fw_fd = -1;
 2525.        unsigned int fw_size;
 2526.        struct stat sb;

nvme.c:3328: error: DEAD_STORE
 The value written to &sec_fd (type int) is never used.
 3326.        const char *namespace_id = "desired namespace";
 3327.        const char *nssf = "NVMe Security Specific Field";
 3328. >      int err, fd, sec_fd = -1;
 3329.        void *sec_buf;
 3330.        unsigned int sec_size;

nvme.c:4935: warning: COMS_6156_TEST
 This is the sample error that we injected for Infering Firmware at line 4935, column
3. This is just a test, relax! In the future we can let our code have this string :)!.
 4933.   {
 4934.        int ret;
 4935. >   bool InferingFirmware;
 4936.
 4937.        nvme.extensions->parent = &nvme;


Summary of the reports

  UNINITIALIZED_VALUE: 2
        RESOURCE_LEAK: 2
           DEAD_STORE: 2
       COMS_6156_TEST: 1
yousef@DESKTOP-2GOH1F7:/mnt/d/nvme-cli$
```

Our forked repository can be found at: https://github.com/YousefElsendiony/infer/
We can see minimal changes were needed to catch this string.

Additional work to do:

- Additional research on constructing modules for infer not already covered types of (e.g. array out of bounds, cast exceptions)
- Understanding different types of nodes in the AST which is analyzed and how to work with them.

Additional References

Adding checkers to Infer:
https://fbinfer.com/docs/absint-framework.html (types of bugs to find)
https://fbinfer.com/docs/adding-models.html (tests)
https://fbinfer.com/downloads/pldi17-infer-ai-tutorial.pdf (not great so far)
https://code.fb.com/developer-tools/al-a-new-declarative-language-for-detecting-bugs-with-infer/ (how to add a checker)
https://github.com/facebook/infer/blob/master/CONTRIBUTING.md (how to develop / build quicker)