Security Venerabilities in Open Source EDA Software

Weston Braun

May 13, 2017

1 Introduction

With the rise of the new "Maker" movement the concept of open source hardware has received signifigant

2 Threat Model

Open Source implies that the design files and code of a open source hardware project are publicly hosted. A typical hardware project consists of a schematic with its associated printed circuit board (PCB) layout, code for a microcontroller, and possibly 3D CAD files for an enclosure. Due to the crossover between the open source hardware and open source software communities, git is commonly used for version control with hosting through github or similar services. Project discovery happens though aggregator websites such as hackaday.com or dangerousprototypes.com, mailing lists, publications such as Make Magazine, or through search engines that index the project page.

Once a user discovers a project they are often interested in viewing the schematic and PCB layout and to do so will download the project files. Unlike

a software project where a makefile can be viewed in a text editor before running any commands from it, or code reviewed before being compiled, most EDA file formats are not designed for direct user readability. The easiest way to view the hardware design files is through the same EDA software that the project developer used, cross compatibility between different EDA programs is uncommon.

The origins of these open source EDA programs date back to a time when public hosting of projects was less common and opening design files from the internet at large was not considered as part of the use case of the average user. However, with the current open source hardware community this is now expected behavior. This difference between the original design philosophy of the software and the current usage presents an appreciable attack surface against an end users computer.

As an additional threat, some PCB manufactures target towards the hobby and open source community have begun directly accepting EDA software files instead of the industry standard gerber files, which are a direct numerical representation of the PCB traces and holes. On the manufactures server a version of the EDA program will generate the gerber files directly to provide a preview for the end user and for the final manufacturing process. This presents an additional attack surface where an exploit in an EDA program can be used potential gain control of a server operated by a PCB manufacture.

3 Target Software

Popular EDA software in the open source hardware community includes KiCad, Eagle, Altium, and gEDA. These programs all feature relatively the same work flow. They consist of a schematic capture tool where the user creates an electrical schematic using a library of component symbols. There is then an association between the schematic symbols and physical part footprints. Once the schematic is finished the user uses a second interface to create the PCB by positioning the physical component footprints and then drawing traces between them.

Of the programs mentioned, both KiCad and gEDA are open source. KiCad is written in C++ and gEDA is written in C. In each of these programs the schematic capture tool and PCB layout tool are isolated as separate independent executables. KiCad is currently the more popular program with development funding being provided by CERN. However, both are available through the Debian repositories with gEDA using the package name 'pcb' and kicad using the package name 'kicad'. Which program to explore in depth was effected by the early results of the vulnerability search with gEDA becoming the focus of the project.

4 Vulnerability Search

Because of the previously mentioned threat model of server side gerber file generation from design files, the search for vulnerabilities focused on the pcb layout component of KiCad and gEDA. The search for vulnerabilities consisted of manual code review added by static analysis and then fuzzing. The biggest attack surface was expected to be the parsers, which parses the human readable PCB layout / netlist consisting of part names, locations, and pin connections into the PCB data structure for display and modification. A vulnerability in the parser would allow for an exploit to execute with the user opening the file without any additional action. An additional attack surface discovered in this process was the user interface itself. Malicious strings can make it through the parser and exploit display formatting functions.

4.1 Static Analysis / Code Review

KiCad and gEDA both have significant code bases, consisting of a parser for file input, functions to modify the schematic or PCB, and display formatting and GUI. According to the CLOC utility, the KiCad PCB utility, pcbnew, has a codebase of 109,386 lines of C++ code while the codebase for the gEDA pcb utility, pcb, has 102,379 lines of C code. This size of this codebase makes full manual review difficult. A static analysis tool, which searches through the code for potentially unsafe functions or pointer arithmetic was used to reduce this search. Flawfinder, a utility available in the Debian package

manager, is capable of parsing C and C++ code and proved suitable for this purpose.

Running on the KiCad pcbnew codebase Flawfinder yielded 104 potential security vulnerabilities. Upon manual review of these hits it was revealed that as a class of vulnerabilities, string based buffer overflows were in large prevented through the consistent use of the wxString library, which dynamically resizes strings, preventing direct buffer overflows. There were some potentially interesting issues relating to race conditions on file access. However, they are not applicable to the threat model of exploitation via a malicious file. Due to the more modern and better maintained codebase and use of C++ features that make buffer overflows more difficult it was decided at this point that gEDA would present a more promising target.

When run on the gEDA pcb codebase Flawfinder yielded 487 potential security vulnerabilities. Upon review of these hits two buffer overflows relating to the pin display names were discovered. Both of these are triggered by user action. One is a heap overflow triggered by the user moving the mouse over a pin which automatically pops up a dialog box showing the pin name. A stack overflow is achievable through a dialog box used by the change pin name tool. Unfortunately, neither of these exploits are capable of bypassing the stack or heap canaries.

4.2 Fuzzing

4.3 Implementation

Fuzzing was used with the hope of finding exploits in the gEDA pcb parser. The fuzzer of choice, AFL-fuzz, requires an instrumented binary compiled afl-gcc, a binary that will terminate on its own, and one or more minimal input files to fuzz. An example PCB layout was reduced down to the minimal feature set and the gEDA pcb source was modified to terminate after parsing the input file. Initial fuzzing attempts on the test VM were very slow (4 executions per second) due to CPU limitations and the fact that the GUI had to launch for every execution. Attempts to were made to isolate the parsing functions from the GUI to speed up execution but the parsing functions

directly use the GUI buffers. This would have required a rewrite of part of the GUI library which was deemed beyond the scope of the project. Using a dedicated machine with a fake x11 display improved performance and allowed for approximately 100 executions per second. The fuzzing was allowed to run for 24 hours to achieve acceptable path coverage.

5 Exploits

asdf

6 Future Work

Future work for this project would include writing replacement functions for the GUI library to bypass the launching of the GUI at every fuzzing execution, allowing for faster fuzzing and better path coverage.

7 Conclusion

asdf