Embedded System Cybersecurity Spring 2021-CSAW-A Final Report

Connor Bushnell (Student)
Vertically Integrated Projects
Georgia Institute of Technology
cbushnell3@gatech.edu

Akshat Sistla (Student)
Vertically Integrated Projects
Georgia Institute of Technology
asistla6@gatech.edu

Cameron Newman (Student)
Vertically Integrated Projects
Georgia Institute of Technology
cnewman35@gatech.edu

Lisa Nute (Student)
Vertically Integrated Projects
Georgia Institute of Technology
lnute3@gatech.edu

Allen Stewart (Advisor)
Vertically Integrated Projects
Georgia Institute of Technology
allen.stewart@gtri.gatech.edu

***Abstract*—This paper provides the final report submission for team CSAW-A in the Vertically Integrated Projects: Cybersecurity of Embedded Systems course. The team demonstrates their ability to use open-source reverse engineering tools, as well as details the creation of their own tools, for the purposes of analyzing and solving challenges presented by past and future CSAW ESC binaries.**

I. Introduction

This report details the discoveries and accomplishments of the CSAW-A team in their efforts to understand and apply open source reverse engineering tools to the challenges from the CSAW 2019 and 2020 ESC. It also details the creation and application of each team member's individually created tool, which aim to overcome competition challenges not already remedied by preexisting open-source tools.

II. Open-Source Tools

A. pwntools

1) Overview: pwntools is a CTF framework written in Python that allows the user to quickly develop and deploy exploits [1]. It has many similar capabilities to other programs, including binary debugging, analysis, instruction patching, and I/O control, but enables the user to statically and dynamically automate these tasks. When combined with tools like Ghidra for binary decompliation, pwntools can be extremely useful for dynamic program analysis.

2) Installation: Details for installation can be found on its wiki page [2].

3) Usage: After installation, using pwntools is extremely simple. To create a script, simply create a `.py` file and head it with the following line:

```
from pwn import *
```

Pwntools is an 'everything but the kitchen sink' library, so importing everything at once is not discouraged. Having access to the entire library is necessary for combating whatever roadblocks or additional analysis a challenge may demand. Going forward, this section is meant to demonstrate a usage of the tool, but as it is not yet functional for RISC-V, it will likely look at a past challenge or a binary from a different competition altogether.

B. Unicorn Engine

1) Overview: The Unicorn Engine is a CPU emulator that can use multiple architectures [3]. This means that it can take unreadable code such as assembly code or binary and converts it to readable code such as Pharo, Crystal, Clojure, Visual Basic, Perl, Rust, Haskell, Ruby, Python, Java, Go, .NET, Delphi/Pascal and MSVC. In addition to this, it has many features such as integration with Arm, Arm64 (Armv8), M68K, Mips, Sparc, and X86. The best part of this is the fact that the setup is native for Windows and Linux.

2) Installation: This can easily be setup in the following ways:

- Mac
  - $ Brew install unicorn
  - To upgrade it just use the command
  - $ brew update
  - $ brew update unicorn
- Windows
  - The zip file for the download can be found online [4]
- Pip
  - $ pip install unicorn
  - To upgrade from an older version of Unicorn, do:
  - $ pip install unicorn –upgrade
  - Remember to stick "sudo" in front for root privilege if necessary

C. hal-fuzz

1) Overview: Hal-fuzz, short for HALucinator: Firmware Re-Hosting through Abstraction Layer Emulation, is a high-level emulator for blob firmware that manually replaces hardware-related library functions in the binary with high-level python. It used to use QEMU and Avatar, but has recently replaced the two with AFL-Unicorn [5].

2) Installation: Hal0fuzz can be downloaded by cloning their repository and running their docker environment:

- Mac/Window (need Docker)
  - $ git clone https://github.com/ucsb-seclab/hal-fuzz.git
  - $ cd hal-fuzz/
  - $ docker build .
  - $ docker run -it ¡imagehash¿ /bin/bash
- Ubuntu (need Python environment)
  - $ git clone https://github.com/ucsb-seclab/hal-fuzz.git
  - $ cd hal-fuzz/
  - $ mkvirtualenv -p /usr/bin/python3 halfuzz
  - $ ./setup.sh

D. Ghidra

1) Overview: Ghidra is an open source reverse engineering tool developed by the NSA [6]. It has a variety of tools to analyze malware and malicious code across many platforms. It is able to run executable formats and processor instruction sets either with automation or user-controlled environments. Capabilities include disassembling, assembling, graphing, and scripting, and creating plug-ins with the open API.

2) Installation: To install Ghidra, download the zip file (found here: https://ghidra-sre.org/) and extract using any unzip program.

Ghidra is usually run in a GUI [7]:

1) Navigate to the directory where you installed Ghidra
2) Run *ghidraRun.bat* for Windows or *ghidraRun* for Linux/mac OS

III. Personal Tools

A. Automated Hardware-Flasher

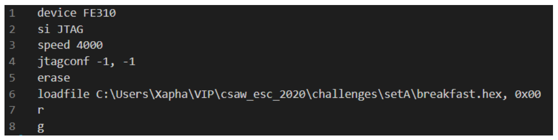
1) Overview: This tool enables the user to automatically flash, connect to, and control the I/O of an associated CSAW ESC device, with the additional functionality of being deployable to a server to allow completely remote access to the device [8]. Flashing each individual challenge between device crashes proves to be a tedious task, and the motivation behind the creation of this tool is to alleviate some of the monotony, as well as to enable teams to work from remote locations when they cannot all meet in person. At present, it is focused on flashing the CSAW ESC 2020 hardware, but it can be expanded to flash challenges to any firmware that can be accessed through SEGGER's JFlash tool [9].

The tool is primarily written in python for interfacing between various tools, such has JLink Commander and, eventually, Ncat and GDB. It is able to take user input, dynamically generate a `.jlink` file based on that input, flash the specified hardware to the device without user intervention, then initiate a JLink connection to the device. With the addition of python's nclib, the tool allows opportunity for the user to manually connect to the target device's WIFI network, then automatically polls the device until it is ready to accept a connection [10]. This greatly enhances the number of successful first-attempt connections and decreases overall time spent troubleshooting them even on an unsuccessful first attempt.

2) Use: The program is simple but effective for greatly decreasing the time and requirements to attempt a challenge. For instance, all it takes to flash the 'breakfast' challenge from CSAW ESC 2020 to the board is to enter:

```
$ python ./autoflasher.py -f ./firmware/
    breakfast.hex
```

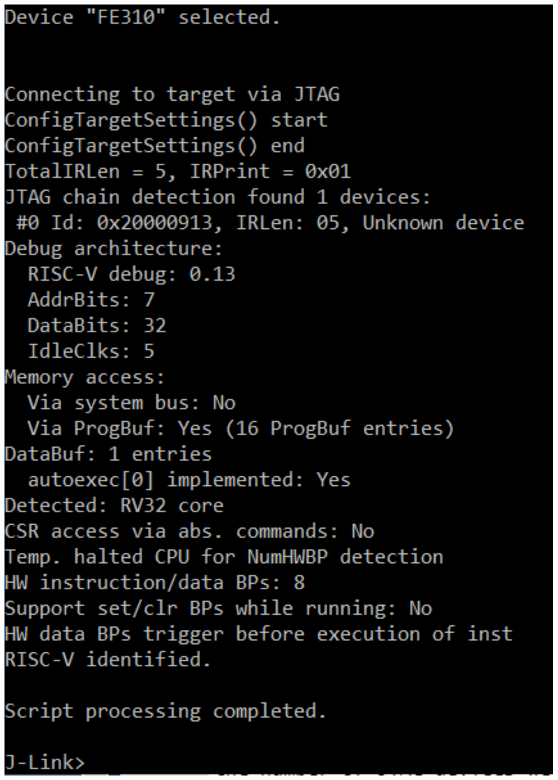
Running the program flashes the firmware automatically, allowing for the device to configure its on-board WIFI and allowing the user to connect to it before proceeding. An example of the `.jlink` file automatically generated and executed by the python can be seen in Fig. 1.



```
device FE310
si JTAG
speed 4000
jtagconf -1, -1
erase
loadfile C:\Users\Xapha\VIP\csaw_esc_2020\challenges\setA\breakfast.hex, 0x00
r
g
```

Fig. 1. Example of a JLink Commander script generated by the auto-flasher.

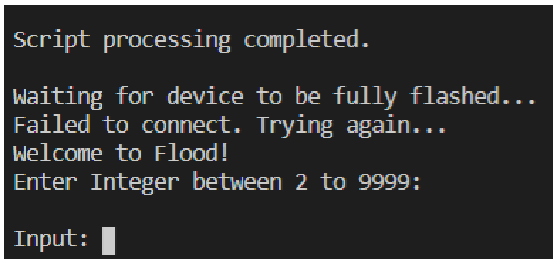
This script specifies, on a line by line basis: the target device, the target interface of said device (always JTAG), the baud rate (speed), and the number of JTAG devices. It then erases the currently flashed firmware, uploads the specified firmware, resets the CPU, then begins the CPU at the initial instruction location. The device is then ready to be accessed and the challenge attempted, as shown by Fig. 2.



```
Device "FE310" selected.


Connecting to target via JTAG
ConfigTargetSettings() start
ConfigTargetSettings() end
TotalIRLen = 5, IRPrint = 0x01
JTAG chain detection found 1 devices:
 #0 Id: 0x20000913, IRLen: 05, Unknown device
Debug architecture:
  RISC-V debug: 0.13
  AddrBits: 7
  DataBits: 32
  IdleClks: 5
Memory access:
  Via system bus: No
  Via ProgBuf: Yes (16 ProgBuf entries)
DataBuf: 1 entries
  autoexec[0] implemented: Yes
Detected: RV32 core
CSR access via abs. commands: No
Temp. halted CPU for NumHWBP detection
HW instruction/data BPs: 8
Support set/clr BPs while running: No
HW data BPs trigger before execution of inst
RISC-V identified.

Script processing completed.

J-Link>
```

Fig. 2. Output of script once the firmware has been flashed and the device connection completed.

To access the device, the tool utilizes the nclib library to communicate over a TCP connection when it becomes available. The tool repeatedly polls the device for a connection for a specified number of failures (occasionally occur due to hardware/firmware error when flashing). As shown in Fig. 3, if a successful connection is made before the specified failure count is reached, the challenge can be attempted.



```
Script processing completed.

Waiting for device to be fully flashed...
Failed to connect. Trying again...
Welcome to Flood!
Enter Integer between 2 to 9999:

Input:
```

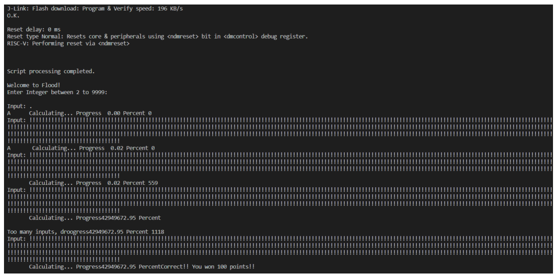
Fig. 3. An example of the first connection attempt being a failure, but the subsequent connection being successful and engaging the tool's I/O capabilities.

Upon failure to connect within the specified limit of attempts, the hardware can be reset with the following command:

```
$ python ./autoflasher.py -f ./firmware/
    breakfast.hex -r -nowifi
```

This can also be done to reset the hardware upon a failed challenge attempt.

Once fully connected to the device, challenges can be attempted in-tool through the pre-established TCP connection, as demonstrated in Fig. 4.



```
J-Link: Flash download: Program & Verify speed: 196 KB/s
O.K.

Reset delay: 0 ms
Reset type Normal: Resets core & peripherals using <ndmreset> bit in <dmcontrol> debug register.
RISC-V: Performing reset via <ndmreset>



Script processing completed.

Welcome to Flood!
Enter Integer between 2 to 9999:

Input: .
A       Calculating... Progress  0.00 Percent 0
Input: !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
A        Calculating... Progress  0.02 Percent 0
Input: !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
       Calculating... Progress  0.02 Percent 559
Input: !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
       Calculating... Progress42949672.95 Percent

Too many inputs, droogress42949672.95 Percent 1118
Input: !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
       Calculating... Progress42949672.95 PercentCorrect!! You won 100 points!!
```

Fig. 4. Completing the 'flood' challenge with the tool.

Additional functionality includes the ability to automate reading/writing from/to the challenge through the included customizable `challenge_script.py` script. This allows for users to control their own I/O, for challenges that require monotonous input or require quick reading from device output and subsequent input faster than human reaction.

3) Future Areas of Research:: Additional functionality is envisioned for the tool which has not been achieved over the course of this semester's development period. These expansions include a transition from JLink Commander scripts to the pylink library, which allows for remote debugging capabilities, such as those offered through SEGGER's GDB server [11]. Testing on an ssh server for remote access has also not been carried out, though should, in theory, be possible, but may require expansion of applicability to Unix systems (for purposes of connecting to WIFI and controlling JLink connections).

B. Solution Generator

1) Overview: This tool allows the user to input several different constraints into a designated area and processes all of them to create a password that satisfies these constraints. This is done through a Python back-end framework and then will eventually include an HTML front-end to make the user experience easier.

2) Setup: Currently, the setup is rather simple. All one needs to do is download the file from the Github and once this is done, one can run the code itself. The setup will be even easier once the code is run once since because a link to the website will be provided and the user experience will be easier.

3) Use: The way one would currently use this product is by taking cloning the Github repository. After this is done, one can simply open up the Terminal or Command Prompt (depending on the Operating System). Now the instructions to run flask can be found on their website [12]. The name of the file is "front(underscore)end". Once the file is run correctly, the website can be found at the provided link in the terminal. This will take you to a page where you can input all the constraints necessary to you. At the bottom of this page, the code will be provided and copied.

4) Future Areas of Research: The areas this project can improve over the next couple semesters is for students to add more functionality to the tool and improve the aesthetic. To improve the functionality, students could add more constraints that the tool can encompass or integrate the code with the Terminal/ Command Prompt. To improve the aesthetic of the tool, one could add a drop down menu to each constraint or add better aesthetics to the web page itself. Doing more research in these areas will help this tool grow.

C. Device Simulator

This tool is a simulator for the CSAW final round devices. The simulation is expected to provide a virtual emulation of the device to interact with and use to solve challenges remotely without the restriction of having physical or simultaneous access. During CSAW 2019, the first place team, Shellphish, was able to simulate the RFID card reader using hal-fuzz, so this tool will focus on this tool as a starting point.

1) Creating the Emulation: To prepare to create the emulation, the following are needed : Docker, Hal-Fuzz, LibMatch, JTAG, and the physical hardware.

With the hardware, JTAG is used to extract the firmware and additional information, like memory layout, needed libraries, toolchain, etc. To use JTAG, the connection pins need to be recognized, firstly. After testing the connection, gather the needed information, and extract the firmware from flash memory [13].

Next using LibMatch, locate the libraries in the firmware and create a database of known functions. The found functions and their addresses should be returned along with any errors, which can be resolved manually if needed [14].

Finally, using Hal-Fuzz, input the information obtained from the previous tools : memory layout, list of functions, etc. Hal-fuzz should create the emulation.

2) Future Areas of Research: The hardware was unavailable, so the firmware could not be extracted for the emulation. So a future goal is to try getting the 2019 device to try this process on, or getting a different device.

D. Code Vulnerability Detector

1) Overview: This tool is a code analysis tool for identifying potential vulnerabilities in C code that can be exploited. In C, the main vulnerabilities are memory mismanagement, buffer overflows, and string manipulation. The tool is a Java program that scans through a C program (as a text file) line by line and highlights those where potential vulnerabilities lie. It then notifies the user of the potential issue and provides a suggestion for a more secure implementation.

2) Setup: To install the program, all you need to do is download the file from the Github into a directory. Then, make a copy of your C program and save it as a .txt file in the same directory and replace the string on line 8 of the program with the path to that file: *File text = new File("insert path here")* Lastly, just run the code.

3) Use: One would use this tool to go over their code to see if they use insecure functions or mismanage memory. They will receive notes on the command prompt saying which function or group of functions was used, why it is insecure, and offer an alternate solution to mitigate the vulnerability. The tool will also note if the number of memory allocations does not match the number of times the user frees that memory. With every run of the program, it will give a reminder not to have insecure imports and that when working with files, one must check if the file exists first, access it directly, and be sure not to overwrite the file.

4) Limitations: As of now, the tool is limited when it comes to checking the security of strings, files, and imports efficiently. One idea for partial implementation would be to cross-check against a known database of secure libraries. However, there is no efficient or successful way to add functionality to check strings or file paths, as they are unique to the programmer. Future work could also include extending the tool or creating a sibling tool to support other languages. Likewise, re-structuring the tool to dynamically analyze code as a user writes it instead of statically analyzing a text file is another thought for the future.

REFERENCES

[1] "pwntools," Available at https://github.com/Gallopsled/pwntools [Accessed 10 February 2021].

[2] "pwntools wiki," Available at https://github.gatech.edu/Embedded-System-Cyber-Security-VIP/ESCS-Hardware/wiki/Pwntools [Accessed 14 April 2021].

[3] "unic," Available at https://www.unicorn-engine.org/ [Accessed 12 April 2021].

[4] "unicin," Available at https://github.com/unicorn-engine/unicorn/releases/download/1.0.2/unicorn-1.0.2-win64.zip [Accessed 12 April 2021].

[5] "Halfuzz," Available at https://github.com/ucsb-seclab/hal-fuzz [Accessed 10 February 2021].

[6] "ghidra," Available at https://ghidra-sre.org/ [Accessed 10 April 2021].

[7] "ghidra installation guide," Available at https://ghidra-sre.org/InstallationGuide.html [Accessed 10 April 2021].

[8] "Auto firmware flasher," Available at https://github.gatech.edu/Embedded-System-Cyber-Security-VIP/ESCS-Hardware/tree/master/CSAW/tools/auto_flasher [Accessed 24 February 2021].

[9] "Jflash segger," Available at https://www.segger.com/products/debug-probes/j-link/tools/j-flash/about-j-flash/ [Accessed 10 February 2021].

[10] "nclib," Available at https://pypi.org/project/nclib/ [Accessed 14 April 2021].

[11] "nclib," Available at https://pylink.readthedocs.io/en/latest/ [Accessed 14 April 2021].

[12] "flask," Available at https://flask.palletsprojects.com/en/1.1.x/cli/ [Accessed 11 April 2021].

[13] "Jtag," Available at https://embeddedbits.org/2020-02-20-extracting-firmware-from-devices-using-jtag/ [Accessed 10 February 2021].

[14] "Libmatch," Available at https://github.com/subwire/libmatch.git [Accessed 10 February 2021].