Challenge Description and Rules:

Secure Audio DRM

Table of Contents

1 Challenge Overview

1.1 Motivational Scenario

You are part of a design team tasked with building a secure audio digital rights management (DRM) module for the next generation of multimedia players. Your employer (a large entertainment company) has outlined a set of requirements for the system and has chosen the development board that they want you to use, but the rest of the design is up to you.

Your challenge is to design and implement the software, programmable hardware, and protocols for the audio DRM module in the next-generation multimedia player: The miPod.

Your employer wants to emphasize security in this new player. The artists and music studios partnering with your employer expect that their songs will be safe from piracy.

Security researcher cracks Google's Widevine DRM (L3 only)

Widevine hack is clever, but it won't spur any waves of Netflix piracy any time soon.

By Catalin Cimpanu for Zero Day | January 3, 2019 -- 21:37 GMT (13:37 PST) | Topic: Security

Figure 1: DRM hacking makes headlines! https://www.zdnet.com/article/security-researcher-cracks-googles-widevine-drm-l3-only/

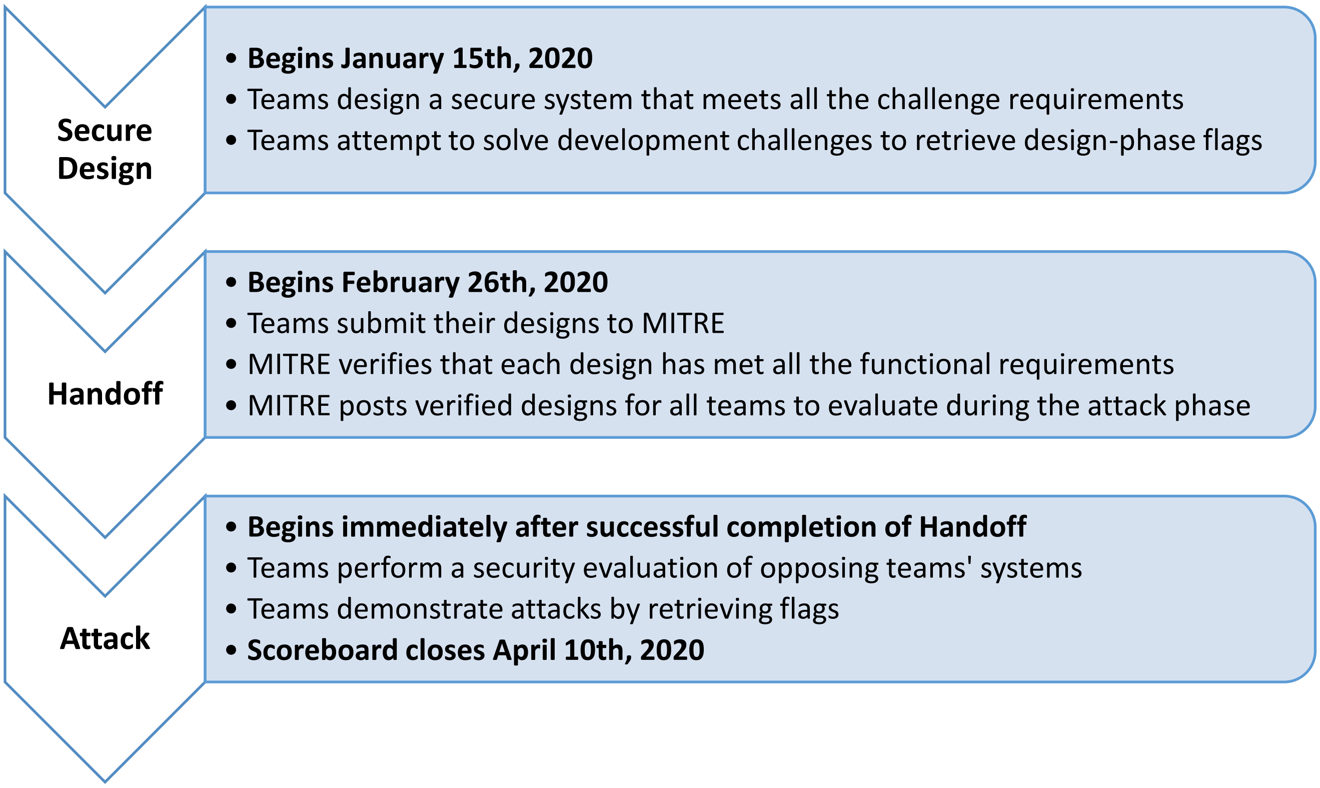
There are several threats that the entertainment company has identified – in particular, the DRM should prevent attackers from being able to:

- Play full versions of songs that they do not permission to play
- Create pirated, bootlegged, or modified songs that will play on the miPod
- Play songs that were provisioned for a different region than their miPod's region

The system that your team creates must meet the requirements specified in this document and defend against as many attacks as you (and your opponents) can think of. You must design and implement a functional multimedia player system which includes support tools to protect music as well as tools to provision players. Once your system is completed, it will be subjected to attacks from opposing teams, while you get a chance to attack the designs from the other teams.

1.2 Competition Phases

This is an attack-and-defend capture-the-flag, meaning there are phases of the competition for both attack AND defense, as summarized in the figure below.



- **Begins January 15th, 2020**
- Teams design a secure system that meets all the challenge requirements
- Teams attempt to solve development challenges to retrieve design-phase flags

- **Begins February 26th, 2020**
- Teams submit their designs to MITRE
- MITRE verifies that each design has met all the functional requirements
- MITRE posts verified designs for all teams to evaluate during the attack phase

- **Begins immediately after successful completion of Handoff**
- Teams perform a security evaluation of opposing teams' systems
- Teams demonstrate attacks by retrieving flags
- **Scoreboard closes April 10th, 2020**

Figure 2. eCTF Phases

The Handoff phase is completed when your team submits a system and the eCTF organizers verify that your submission meets all functional requirements. Therefore, the date and time of transition between phases may vary between teams. Each team is allowed into the Attack phase by the eCTF organizers as soon as that team completes Handoff. For example: if Team-A and Team-B both submit systems on the handoff date, but only Team-A's system passes the tests, then Team-A will move to the Attack phase, while Team-B remains in the Handoff phase until they resubmit their system with the necessary fixes to pass the tests.

Positive Tip!

There are significant scoring advantages to entering the Attack phase as soon as possible. We highly recommend setting a schedule for your design and implementation and sticking to it as closely as possible. If your schedule starts to slip, your team will need to make hard decisions about what security features can be excluded to submit a working design on time (or at least as soon as possible). Final testing always takes longer than expected and sometimes reveals tricky problems – so plan to start your final testing earlier than you think is necessary.

1.3 System Overview

Your team has been tasked with building the DRM module for your company's upcoming media player. Your employer is very excited to have partnered with several high-profile record labels who they have given guarantees to about the security of the player. For example, the labels have been promised that the music on the devices cannot be tampered with, ensuring that end users can listen to the audio as the artists intended. Furthermore, the record labels are concerned with people playing illegally downloaded music on the miPod. As such, your employer has promised the labels that only songs that properly provisioned for the miPod will play on it. Finally, your employer has promised record labels the ability to “region lock” certain songs to only allow them to be able to be played in certain parts of the world. The record labels plan to use this to comply with licensing agreements they have in different areas of the world.

Your employer also has a set of functional requirements that they are asking you to incorporate into the player. Specifically, users should be able to play, pause, resume, stop, and restart audio. Furthermore, your employer has plans to market the miPod as a family player, meaning that it must support multiple users. Users should only be able to listen to the full version of songs that they own, or songs that another member of their family has shared with them (assuming the song and player have been provisioned for the same region). However, all users should always be able to listen to a 30 second preview of any song on the player, regardless of if they own the song, and regardless of the region. Finally, your employer wants to ensure compatibility with various types of digital speakers. This means that your player must not only support playing audio from the headphone jack, but also writing audio to a file for digital playback.

In order to make your player more appealing to consumers, your employer has two optional goals that they are giving your team. The first is allowing users to skip forward and backward in a song. The second is allowing for higher quality audio playback. If implemented, these features will make the player more desirable, however, they also will make implementing a secure design more difficult. As such, your team should carefully consider if you want to implement these features.

2 Provided Materials (The Kit)

Building a secure audio DRM system from scratch is a daunting task; therefore, MITRE is providing a full example system (both hardware and software) that meets the functional requirements for the competition. You can use this system for testing and as a reference to help understand the requirements. You may also use it as a base to build on for your own system.

2.1 Hardware

MITRE will provide each team with a set of hardware and peripherals required for the competition. Teams may choose to obtain additional units of the hardware to increase testing and attack development capabilities for their team. However, your system must work with the hardware that was provided. Switching to a different platform or modifying the physical hardware during the design phase is not allowed. The following hardware is provided for each team:

- Cora Z7-07S Boards (2x)
- PMOD I2S2 Modules (2x)
- SD Card Readers (2x)
- Micro-USB Cables (2x)
- Micro SD Cards (2x)
- Headphones (1x)

2.1.1 The Chip — Zynq-7000 / ARM Cortex A9

The chip we will be using for this challenge is the Xilinx Zynq-7000 System on a Chip (SoC) which combines a single-core ARM Cortex-A9 based processing system (PS) with Xilinx programmable logic (PL). While the PL can provide teams with unique design opportunities, modifying the hardware design (i.e. the PL) is not required by the competition. Check out the Datasheet for the chip.

2.1.2 Restriction: Hardware AES/HMAC Engine

You will inevitably read through Section 32: Device Secure Boot in the Zynq-7000 Technical Reference Manual (TRM). This section details the hardened AES/HMAC Engine which resides in the PL. This engine is NOT available for use by your design as it is used by the MITRE-provided First Stage Boot Loader (FSBL) to decrypt and authenticate your designs during the boot process.

2.1.3 eFUSE Settings

The boards provided by the eCTF organizers will have appropriate eFUSE settings burned and will not be modifiable.

2.1.4 PMOD I2S2 Audio Module

We will also be using an audio codec chip to drive the audio signal. For this, we will be using the PMOD I2S2 Module from Digilent. This device will be connected to PMOD Header JA on the CORA Z7-07S board to output audio.

2.2 Code Repositories

Git repositories are provided that contain source code for an insecure example system that meets all the challenge requirements. Additionally, a repository is provided that contains instructions for setting up the required development environment and provisioning the example system. A summary of these is provided below:

- `2020-ectf-insecure-example/`
  - `miPod`
    - `Contains the source code for the Linux miPod CLI application to interact with the DRM module`
  - `vagrant`
    - `Contains code for vagrant provisioning scripts`
  - `mb`
    - `Contains code for a DRM module implementation in the MicroBlaze`
  - `pl`
    - `Contains source for the programmable logic to create the FPGA implementation`
  - `tools`
    - `Contains code for provisioning the reference design`
  - `vivado-boards`
    - `Submodule for Cora vivado board information`
  - `Vagrantfile`
    - `File responsible for creating the vagrant environment`

2.2.1 Insecure Example Implementation [updated v1.1]

The Insecure Example Implementation repo can be found at: https://github.com/mitre-cyber-academy/2020-ectf-insecure-example

The example implementation is comprised of three main components: the firmware for the Audio DRM module, the Vivado project (which contains the hardware design that is used to configure the programmable logic) and the Linux miPod application. The Vivado project included in the main repository does not need to be modified to meet the functional requirements. Should teams be interested in modifying or adding FPGA hardware, they may refer to the `pl` folder. Any updates to the PL must build using either Vivado version 2017.4 or 2019.2. You must notify the organizers by February 12th if you wish to use the latest version of Vivado (2019.2). If you do not tell the organizers you wish to use 2019.2, it will be assumed that your design is for Vivado 2017.4 (the version used by the reference design).

WARNING: This example system is insecure. The purpose of providing it is to give participants a reference to help in understanding the functional requirements. There was little or no effort placed on securing this system – that's your job! You may use the example system as a base and add your security features on top, or you may choose to start from scratch so that security is designed in from the start.

2.2.2 Development Environment

Along with the example system, we are also providing a Vagrantfile[1] with all the tools necessary to build and test the example. When you submit your design, you will need to include an updated `customizations.sh` and `config.rb` with all your dependencies so that the organizers can build and test your implementation. You may not modify any of the other vagrant files or change the folder structure (and in fact, the Vagrantfile itself will be replaced with the organizer's file during testing). Your customizations should build and install any tools necessary to run or test the design. Please refer to the documentation included in the git repository for more information.

[1] https://www.vagrantup.com/about.html

3 Secure Design Phase

The secure design phase encompasses the design of a secure DRM module and the creation of supporting tools. The following sections provides the requirements for your system and support tools.

3.1 Music Player Description

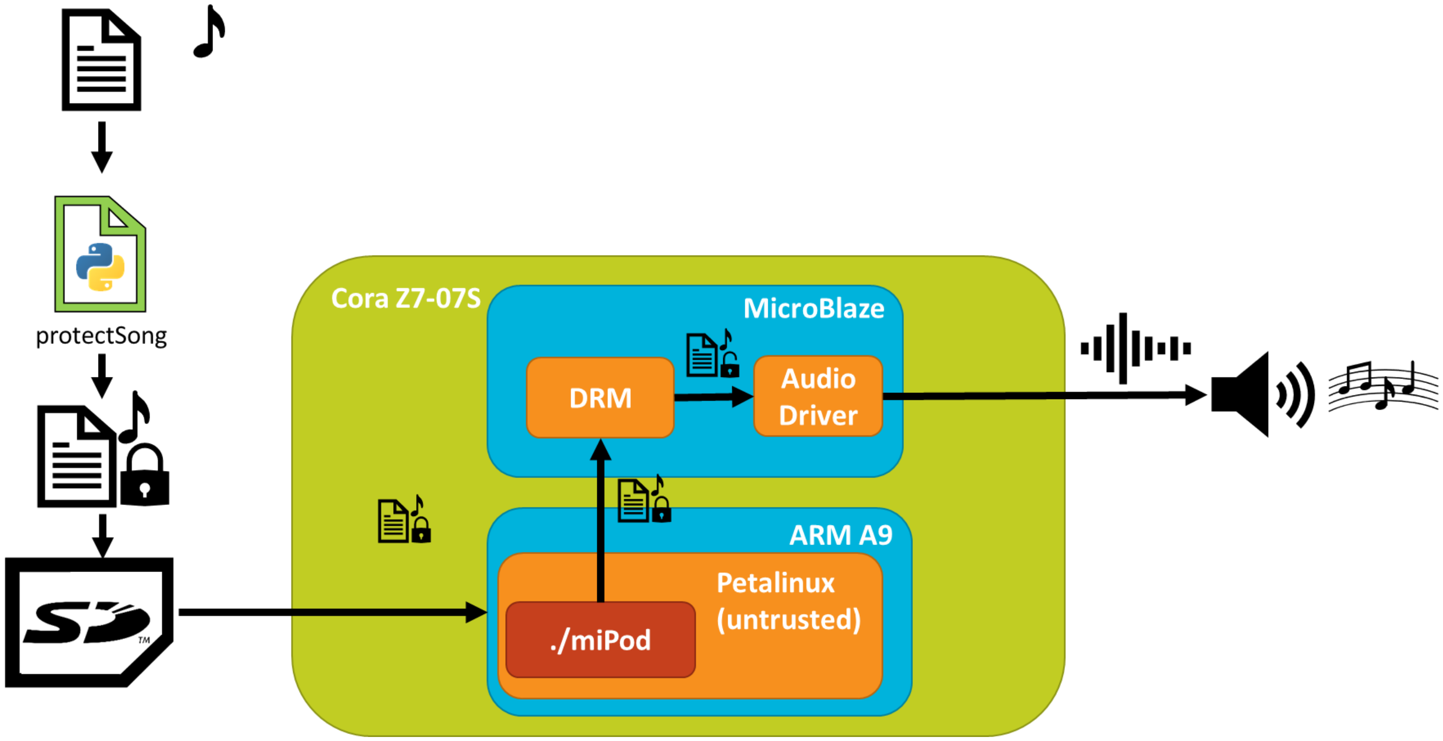


Figure 3: System Operation Diagram

The ARM Cortex-A9 PS on the Zynq will be running Petalinux (Xilinx's embedded Linux kernel) as the main interface that controls the system. However, since customers who purchase the miPod will have full control over the system, the ARM A9 processor must be considered compromised as a potential adversary will have full root access to the operating system. Thus, the audio DRM module must be isolated from the main processor in some way to prevent an attacker from stealing music or using it improperly. In the example design, a soft-core MicroBlaze Processor is implemented in the PL to act as a secure audio DRM module.

WARNING: The Petalinux operating system is **not** part of your submission and thus will be insecure when your system is provisioned for the attack phase. One of the main challenges in this competition is creating a secure module in the presence of a compromised main processor.

3.1.1 Audio Digital Rights Management (DRM) System

The audio DRM module has two main roles: rights management/protection and audio driving. The module must take in protected song files and apply the appropriate actions based upon how it is provisioned. That is, this DRM system must check that a song can be played by the current user and must apply region locking. Finally, this system must also drive the audio for the song that has been fed to it via the PMOD I2S2 audio module.

3.1.2 User Interface / Operating System

The ARM Cortex A9 processor will run Petalinux and a root-accessible shell will be used for the user interface. This will allow users to mount the music partition of the microSD card and play protected songs from there. The provided Petalinux builds will include a variety of useful packages for debugging and reverse engineering:

- python3
- python3-crypt
- libpython3
- python2
- python2-crypt
- libpython2
- openssl
- libcrypto
- gdb
- vim

WARNING: Since the operating system has open root access, one must be careful about their memory usage. That is, one must make sure to not perform any sensitive memory operations in regions that the A9 processor can access.

3.1.2.1 ./miPod

The miPod application is responsible for taking a protected song as input and passing it to the audio DRM module. This module will take the appropriate actions and drive audio. The application will run in Petalinux.

NOTE: This file does not have a required extension. That is, it can be an .sh, .elf, .py etc, however, the application must be callable by running "./miPod" from the command line without any modifications. The example design implements this as an elf binary.

3.2 Security Goals

3.2.1 Confidentiality

Music should be protected to prevent songs from being played or accessed in full if the proper conditions have not been met. For example, if the song is not owned by the current user, has not been shared to the current user, or is locked out of a player's region, only a preview portion is allowed; the rest should be considered confidential. This is to prevent unauthorized listening to audio that has not been purchased. As the audio files will exist on an external SD card, it is important to note that this means that the audio samples should not be accessible from another player or external device if they could not be played on the miPod. The studios that your company has partnered with do not want end users to be able to access samples without it passing through the miPod DRM software first.

3.2.2 Integrity and Authentication

Your system should contain some mechanism(s) for validating that a given song is from a legitimate source and was not altered in any way. Music that fails integrity or authentication checks should not be played. Studios and artists that have specific requirements that their music be played as it was recorded; they don't want it to be changed or tampered with. In addition, custom music should be prevented from playing on the player as it may have been obtained through illegal means. Users must also be authenticated with a username and pin combination so that songs for a particular user cannot be listened to in full by someone who has not had the song shared with them.

3.3 Functional Requirements

3.3.1 Ownership

For the full song to be listened to, it must be owned by, or shared with, the current user and must match one of the regions that the player has been provisioned for. Otherwise, listeners can only listen to a preview portion of the song. Song previews are **30 seconds** in length starting from the first sample of the file. After that time has expired, if the song has not been purchased, playback must stop immediately. Note that a song that was not initially provisioned for a user could later be shared by one of the owners of the song with that user!

3.3.2 Region Locking

Your system must support region locking for up to 32 different regions. That is, each player will be provisioned for one or more (up to 32) regions. When a song whose region matches the player is queued (so long as it is owned or shared with the current user as well), the ***whole*** song ***must*** play. However, when the song's intended region does ***not*** match that of the player, **only** a ***preview portion*** of the song ***must*** play. The "previous portion" must be the 30 second portion of the song starting from the first sample of the file.

3.3.3 Login

As the player must support multiple users, there must be a way for a user to log into the device. Users must be uniquely identified by a username and protected by a pin. The player must support up to 64 different users.

Parameter	Length	Allowable ASCII Characters
Username	1-15	a-z, A-Z
PIN	8 -64	0-9

Table 1: Username and Pin Requirements

3.3.4 Region and User Information

On boot, the DRM module must output the regions that it has been provisioned for, as well as the users that exist on the device. This must match the following regular expression:

output `mP> Regions: [a-zA-Z]+[, [a-zA-Z]+]*\r\nmP> Authorized users: [a-zA-Z]+[, [a-zA-Z]+]*\r\n`

3.3.5 Song Sharing [updated v1.1]

Your employer wants to market the miPod as a family device. As such, they are requiring that a "family sharing" feature exist in all players. If a user owns a song, they can share the song with any other users on the player, allowing this user to listen to the full version of the song. However, a user can only share a song if they are the owner. A song can be shared with as many users as could be provisioned for a player. Sharing must be persistent across reboots.

For example, let's assume that user "A" owns song "foo" and shares it with user "B". User "B" now can listen to the full version of the song (assuming it matches a region with the player), however, user "B" cannot share the song with another user, since only user "A" owns the song. If the board is rebooted, User "B" can still listen to the full version of the song.

3.3.6 Digital Output Mode

Your employer is also requiring that you provide a Digital Output mode in all players. Another team within your company is handling the specifics of this mode, however, they have required that you output the raw audio to a file with a ".dout" extension. This file must match the original audio file provided during the provisioning process (i.e., with all protections removed). This requirement is to allow a device to play music through a speaker that accepts digital out in the form of a

WAV file. While that will eventually be hardwired to the DRM (and thus inaccessible to the end-user), that team designing that part of the miPod is behind schedule, so for a proof of concept, your team must dump the WAV song to a file instead.

3.3.7 Optional Functionality

These features are optional and do not need to be implemented for your team to enter the Attack Phase. If your team can successfully implement the following features, you will be awarded with points for each feature. See Section 6 for more information on scoring.

- *Seek*: Users may be able to jump back 5 seconds in the audio or jump forward 5 seconds in the audio. Note that both forwards and backwards must be implemented in order to receive credit for this feature.
- *Audio Quality*: Users may listen to audio at either a 22KHz or 48KHz sample rate. See section 11.1 for more information on audio quality.

3.3.8 miPod Usage

Running the "miPod" application must launch a prompt to allow the user to interact with the DRM module. This prompt must be launched by running "./miPod" from the directory containing the application, and must conform to the following regular expressions:

- A blank line:
  - \r\n
- Any MicroBlaze printout:
  - MB> [^\r\n]*\r\n
  - Example: `MB> Audio DRM Module has Booted\r\n`
- Any miPod print out:
  - mP> [^\r\n]*\r\n
  - Example: `mP> Unrecognizes command.\r\n`
- miPod user input prompt:
  - miPod [^\r\n]*# [^\r\n]*\r\n
- Song query:
  - mP> Regions: [a-zA-Z]+[, [a-zA-Z]+]*\r\nmP> Owner: mP> [a-zA-Z]+[, [a-zA-Z]+]*\r\nAuthorized users: [a-zA-Z]+[, [a-zA-Z]+]*\r\n
  - Example:
    - `mP> Regions: USA, Canada, Mexico\r\n`
    - `mP> Owner: alice\r\n`
    - `mP> Authorized users: bob, charlie, donna\r\n`
- Initial boot output (upon boot, your miPod application must output all of the regions and users provisioned for the board, according to this regular expression):
  - mP> Regions: [a-zA-Z]+[, [a-zA-Z]+]*\r\nmP> Authorized users: [a-zA-Z]+[, [a-zA-Z]+]*\r\n
    - `mP> Regions: USA, Canada, Mexico\r\n`
    - `mP> Authorized users: alice, bob, charlie, donna\r\n`

Users can issue the following commands:

Command	Argument(s)	Action if Logged in	Action if Not Logged In
`login`	`<Username> <Pin>`	Invalid command	Verify username and pin are valid; log in the specified user
`logout`		Log out the current user	Invalid Command
`play`	`<song_name>`	Verify that the current user owns the song (or has had the song shared with them), and that the song shares a region with the player. If both conditions are met, play the full song. Otherwise, play a 30 second preview.	Play a 30 second preview of the song
`pause`		If a song is playing, pause the song. Otherwise, invalid command.	If a song is playing, pause the song. Otherwise, invalid command.
`resume`		If a song was paused, resume playing. Otherwise, invalid command.	If a song was paused, resume playing. Otherwise, invalid command.
`stop`		If a song is playing, stop the song. Otherwise, invalid command.	If a song is playing, stop the song. Otherwise, invalid command.
`restart`		Restart the song from time = 0:00.	Restart the song from time = 0:00.

`share`	`<song_name>`
	`<username>`
		Verify that the current user owns the specified song. If so, share it with the specified user. Otherwise, invalid command.	Invalid command.

`digital_out`	`<song_name>`	Output the contents of a song to a digital file as opposed to playing over the audio jack. If the current user owns the song (or has had the song shared with them), and the song matches the board region, output the song in full. Otherwise, it should output 30 seconds of audio.	Output 30 seconds of samples of the audio file to a digital file as opposed to playing over the audio jack.
`query`	`<song_name>`	List the users and regions that a song has been provisioned for.	List the users and regions that a song has been provisioned for.

Table 2: ./miPod Command Information

Additional details and requirements for several command features are described below.

- *Real Time Audio:* As this is a real time system, any audio glitches will immediately be noticed by the user. There must not be any gaps, stutters, or other audio glitches during playback.
- *Resume*: Users must be able to *resume* paused audio. Note that this functionality does not need to persist through a device restart. That is, if audio is paused and the device restarted, it does not need to be possible to resume from the paused location. If another song is started, it does not need to be possible to resume the original song.
- *Stop*: Note that this functionality is different from *pausing/resuming*. If a song is *paused*, it may be *resumed* from the point where it was paused from. If a song is *stopped*, *playing* the song again should start from the beginning.
- *Song sharing:* Users who own a song can share it with any other user on the device, allowing that user to listen to the song, if the song shares a region with the player.
- *Digital Output Mode:* For songs that a user can only preview, only the 30 second preview must play. This does not need to be done in real-time.

3.3.9 Timing and Performance Requirements

The system must adhere to the timing/performance requirements outlined below. Remember that if this were a real music player, it would need to be useable and enjoyable to users, and nobody likes waiting for their system to boot up. If your system does not meet these Max Time requirements, it will not be accepted during Handoff.

Operation	Max Time for Completion
DRM Boot Up	5 seconds*
Starting the miPod application	1 second
Log in/log out	1 second
Play audio	1 second**
Pause audio	1 second
Resume audio	1 second
Stop audio	1 second
Restart audio	1 second
Query	2 seconds
Jump forward/backwards	2 seconds
Digital output mode	No longer than the length of the song
Share song	5 seconds

Table 3: Operating Time Requirements

** The system must be ready to receive commands 5 seconds from when FSBL completes its boot process*

***The system must start outputting the appropriate acoustic signal from the audio codec within 1 second after executing the play command.*

3.3.10 Non-destructive operation

The system must NOT attempt to prevent attacks by making the system unusable, as this is not a valid technique for systems that need to be deployed to customers. Any method used to try and destroy the hardware (e.g., thousands of writes in rapid succession to flash) is not permitted. Designs that are determined to be intentionally created to render the hardware inoperable will be disqualified and potentially banned from future competitions.

3.3.11 Invalid Audio Files

If you player detects that it is being asked to play an invalid audio file, this operation can fail. For example, if a song is loaded into memory, and your DRM module detects that this song was not provisioned for this player, the player may refuse to play the song (or in the case of Digital Output mode, is allowed to not output the song to a file).

3.4 Support Tool Descriptions and APIs

In addition to the DRM module and prompt, you must also design and implement multiple tools to provision, build, and deploy your system. The reference design implements these as python scripts. Upon design validation, each tool must exist in the "tools" directory after vagrant has completed booting. These tools must do the following:

1.) Provision any global region information **(createRegions)**. This script will be only run once.
2.) Provision any user information **(createUsers)**. This script will only be run once.
3.) Securely protect songs **(protectSong)**. This script will be run multiple times and does not make any assumptions about what device it is being run for.
4.) Provision any device specific information **(createDevice)**. This script will be run once for each device provisioned.
5.) Build a specific device **(buildDevice)**. This script will be run once for each device provisioned.
6.) Create an SD card image of your system, including any songs for the device **(deployDevice)**. This script will be run once for each device provisioned for your team.

Examples of these support tools are provided in the insecure example. The example tools may be modified by your team, however, the tools must conform to the requirements described in the following subsections, and must exist on $PATH. If you chose to implement them in a language other than python that requires compilation, they must be built during your vagrant provision process.

There is one additional script **(packageDevice)** that will be run once for each device provisioned for your team. This script is responsible for combining the DRM firmware into a miPod.BIN. It also is responsible for using hardware keys to encrypt and add authentication to the miPod.BIN during the attack phase. This script is provided by MITRE and may not be modified by your team. This script can be used in the design phase to generate a miPod.BIN for testing. During the attack phase, the eCTF organizers will use this script (along with a set of hardware keys) to create the official attack images which will be protected during the loading process.

3.4.1 The Provisioning Process

The provisioning process (shown below) begins with **createRegions**. This script will be run once to define the possible regions that players could be provisioned for, storing any output in the `region.secrets` file. For example, the reference implementation stores a JSON object in this file with a mapping between the region names and an integer corresponding to the region.

The **createUsers** script is then used to define all the possible users that could exist across devices. Usernames and pins are provided to this script and any output should be written to `user.secrets`. The reference implementation stores a JSON object in this file with a mapping between usernames, user IDs, and pins.

As soon as all the regions and users have been created, devices (players) can be provisioned and songs can be protected.

The **protectSong** script takes the `region.secrets` file, the `user.secrets` file, a list of regions to provision the song for, and the user the song is owned by. It will produce a “protected song” (DRM file) that can be loaded by any player. Only users who have been provisioned for the song can listen to the audio, and only on devices that share a region with the song.

Next, the `region.secrets` and `user.secrets` are provided to the **createDevice** script, along with a list of regions and users to provision for this device. This script may be run multiple times to provision different devices, and might be run before the device is build. This means that your script needs to consider both of the following workflows:

1. **createDevice** is run multiple times, then each device is built and packaged
2. **createDevice** is run one, then that device is built and packaged. That process is repeated multiple times.

The reference implementation writes necessary information into C header files so that they can easily be included in the DRM firmware source code.

Positive Tip! ***It may not necessary to have all of the information in the secrets files be built into your drm_audio_fw.elf file. It might be useful to use appropriate #ifdef encapsulation to control this.***

Once a device has been created it can be built into a binary. This is done via the **buildDevice** script. This is responsible for building any programmable logic (using Vivado) and well as the DRM firmware and miPod Linux application (using the Xilinx SDK). The programmable logic bitstream and output `drm_audio_fw.elf` are then combined using the `updatemem` command into a file called `download.bit`. This is then passed to the organizer-provided **packageDevice** script that is responsible for packaging the components into a bootable image (`miPod.bin`) using the `bootgen` command. During the attack phase, this script is also responsible for adding encryption and authentication to the resulting binary to secure the load process. Once the songs have been generated, they can be provided along with the BOOT image to the **deployDevice** script that will create an SD card that can be provided to the board. This includes the ability to format a fresh SD card. The songs will be loaded onto the second partition of the SD card (the ext4 partition) and the BOOT image onto the first (the FAT32 partition). This directory structure should look as follows:

- BOOT partition:
  - `BOOT.bin`
  - `image.ub`
  - `miPod.bin`
- `root` partition
  - `miPod`
  - `audio1.drm`
  - `audio2.drm`

NOTE: The combined bitstream will be packaged into its own binary (miPod.bin).

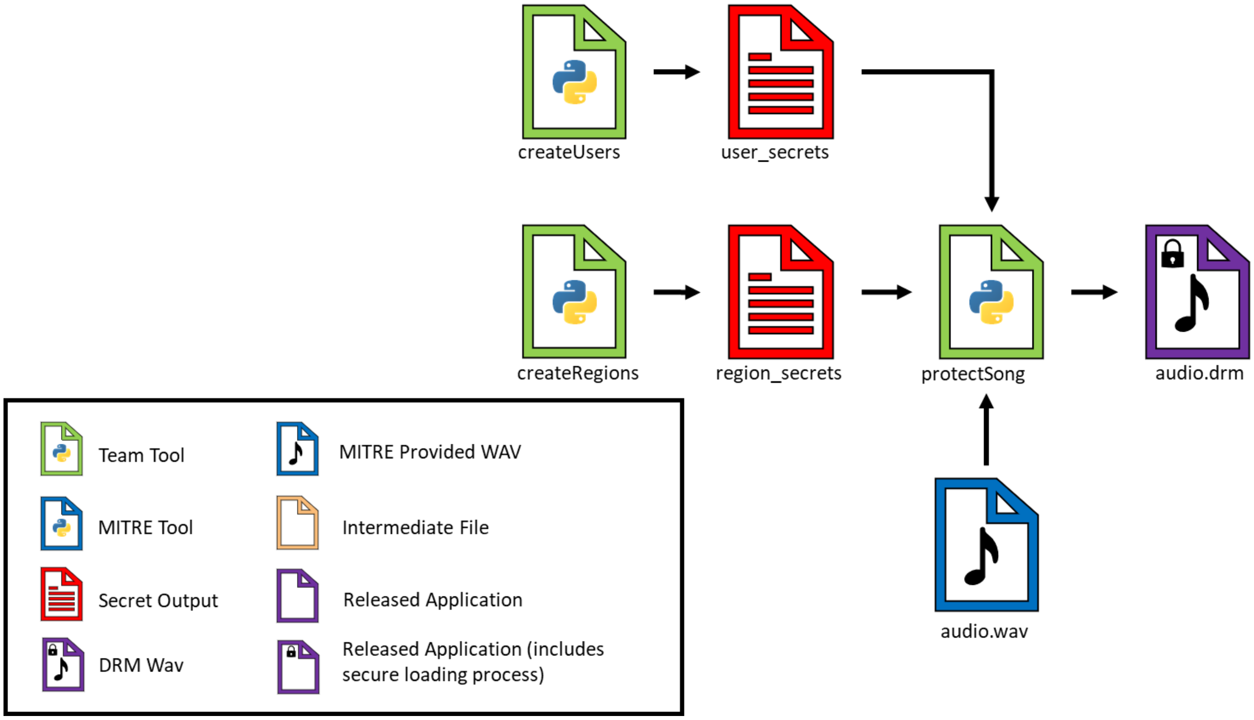


Figure 4: Song Protection Process

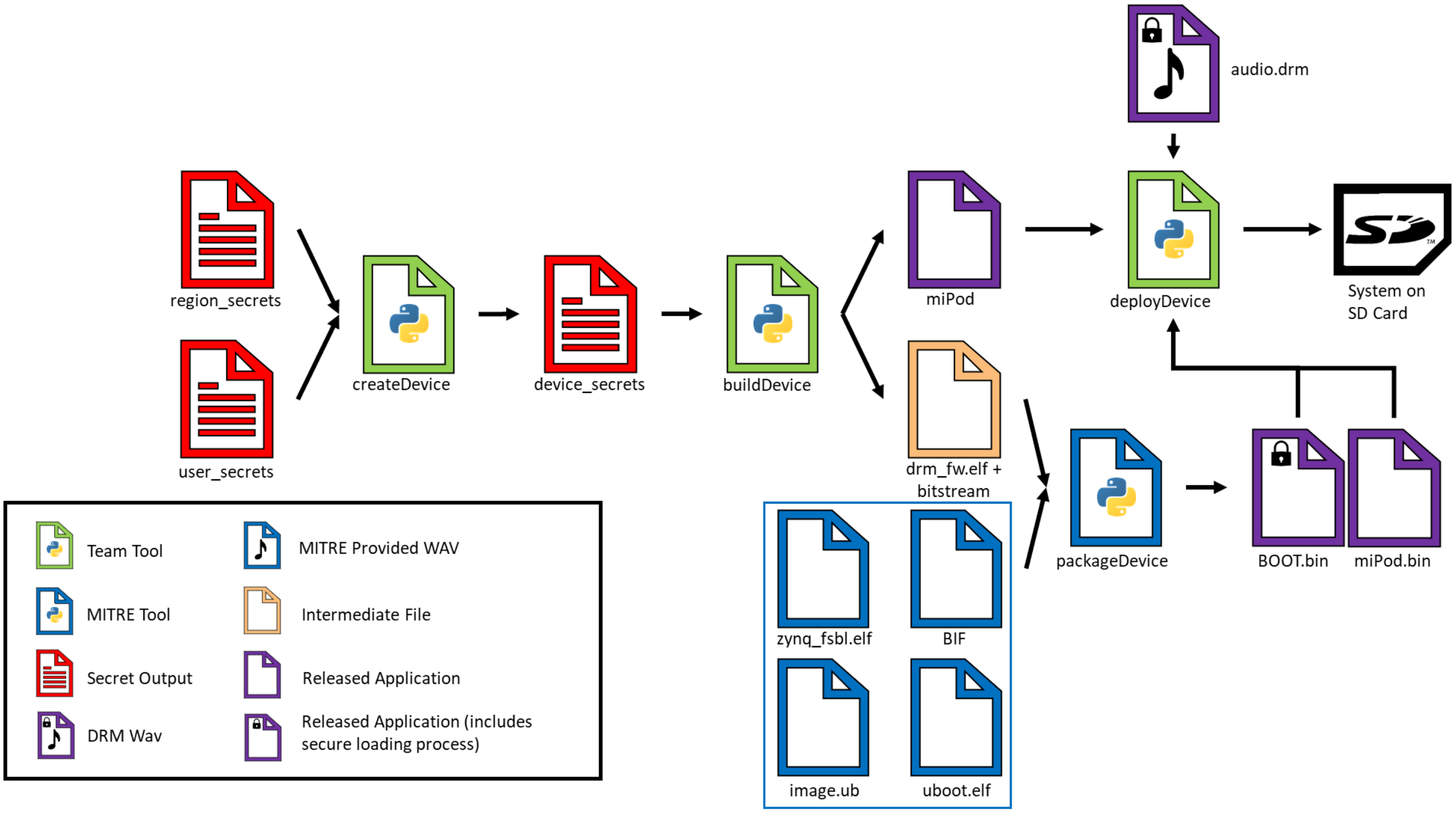


Figure 5: Device Provisioning Process

3.4.2 createRegions

Syntax: "`./createRegions –-region-list <REGION_LIST> --outfile <REGION_SECRETS>`"

Description: This script is used to provision the secret components for an ecosystem of devices belonging to various regions. It will only be run once to globally provision all regions.

Inputs:

- <REGION_LIST>: A space separated list of all possible regions. Note that regions may have spaces in them, but if they do, they will be enclosed in quotes.

Outputs:

- <REGION_SECRETS>: Any secret information will be stored in this file. This file will be kept secret and is not provided to other teams. The contents and structure of this file is up to your team.

3.4.3 createUsers

Syntax: "`./createUsers --user-list <USER_PIN_LIST> --outfile <USER_SECRETS>`"

Description: This script is used to provision all possible users that could exist on any device. It will only be run once to globally provision all provided users.

Inputs:

- <USER_PIN_LIST>: A space separated list of username and pin pairs. The usernames and pins will be separated by a colon. For example, "foo:1234567890" would be a username of "foo", and a pin of "1234567890".

Outputs:

- <USER_SECRETS>: Any user secret information will be stored in this file. This file will be kept secret and is not provided to other teams. The contents and structure of this file is up to your team.

3.4.4 protectSong

Syntax: "`./protectSong --region-list <REGION_LIST> --region-secrets-path <PATH_TO_REGION_SECRETS> --infile <PATH_TO_SONG> --outfile <PATH_TO_OUTPUT_SONG> --owner <USER> --user-secrets-path <USER_SECRETS>`"

Description: This script is used to securely package songs to be used with the DRM system. It takes in some a song as well as relevant metadata, and outputs the DRM protected song.

Arguments:

- `<REGION_LIST>:` List of country names to region-lock a song to. These names are simply separated by a space. Valid names include: USA, Canada, Mexico, Australia, and Japan.
- `<PATH_TO_REGION_SECRETS>`: The absolute or relative path to region secrets file.
- `<PATH_TO_SONG>`: The absolute or relative path to the input song to protect. Input song must conform to our audio requirements (see section11.1）.
- `<PATH_TO_OUTPUT_SONG>:` the absolute or relative path to save the output song to.
- `<USER>:` The username that the song is owned by.

- `<USER_SECRETS>`: The path to the user secrets file.

Outputs:

- The DRM protected version of the input song to be used in the miPod system, written to <PATH_TO_OUTPUT_SONG>. This file will be copied to the music partition of the microSD card for playback.

3.4.5 createDevice

Syntax: "`./createDevice --region-list <REGION_LIST> --region-secrets-path <REGION_SECRETS_PATH> --user-list <USER_LIST> --user-secrets-path <USER_SECRETS_PATH> --device-dir <OUTPUT_FOLDER>`"

Description: This script is used to create a device for the specified regions and users. It will be run once per device per team. For example, this script could be run once to create a device for the United States, once to create a device for Japan, and once to create a device for the United States and Japan.

Inputs:

- <REGION_LIST>: A space separated list of regions to provision this device for. Note that regions may have spaces in them, but if they do, they will be enclosed in quotes.
- <REGION_SECRETS_PATH>: The path (relative or absolute) to the `region.secrets` file created by the **createRegions** script.
- <USER_LIST>: A list of usernames to provision for this device. Usernames will be space separated.
- <USER_SECRETS_PATH>: The path to the `user.secrets` file created by the **createUsers** script.
- <OUTPUT_FOLDER>: The path (relative or absolute) to store any device related information required to build a device. This is to allow a unique location for any device related information to be stored separate from the build directory (to allow **createDevice** to be run multiple times without overwriting information).

3.4.6 buildDevice

Syntax: "`./buildDevice -p <DEV_PATH> -n <PROJ_NAME> --secrets_dir <PATH_TO_SECRETS_DIRECTORY>`"

Description: This script is responsible for building a device that can later be put onto an SD card. This script will be run once for each device created by the **createDevice** script. The reference design implements an optional argument to specify which phase of building you would like to run. This is not required to be implemented in your final design.

Inputs:

- <DEV_PATH>: The path to the root of your repository.
- <PROJ_NAME>: The name for your `Vivado` project.
- <PATH_TO_SECRETS_DIRECTORY>: The path to the secrets directory for the device to be built.

Outputs:

- The combined bitstream is written to `/ectf/mb/Cora-Z7-07S/download.bit`, and the `miPod` Linux application to `<PATH_TO_SECRETS_DIRECTORY>/miPod`. These files must exist in these locations upon a successful build.

3.4.7 packageDevice (Python 3.6)

Syntax: "`./packageDevice <SYSTEM.bif> <OUTPATH> <BITSTREAM_PATH>`"

Description: This script will use an organizer provided bif to create a `miPod.bin` that can be put onto an SD card to boot the system. This script will be provided by the organizers and cannot be modified by the participants.

Inputs:

- <SYSTEM.bif>: A path to the BIF to provide to `bootgen` to create a miPod.bin
- <OUTPATH>: The path to the output file (miPod.bin)
- <BITSTREAM_PATH>: The path to the bitstream file that will be used in the BIF.

3.4.8 deployDevice

Syntax: "`./ deployDevice <SD_DEVICE> <BOOT.bin> <miPod.bin> <AUDIO_FOLDER> <MIPOD_APPLICATION> <IMAGE.ub> [--no-format]`"

Description: This script will format the SD card and deploy the various components of the system to it. There is an optional flag to preserve the format of the card and only copy the provided files.

Inputs:

- <SD_DEVICE>: The path to the SD card device to deploy the system to.
- <BOOT.bin>: The path to the BOOT.bin
- <miPod.bin>: The path to the miPod.bin, containing your bitstream
- <AUDIO_FOLDER>: The path to the folder with the provisioned audio files.
- <MIPOD_APPLICATION>: The path to the miPod application to run in Linux.
- <IMAGE.ub>: The path to the Petalinux kernel.
- --no-format: an optional flag that if present will not format the SD card.

3.5 Development Environment Requirements

In addition to the functional requirements specified above, your submission must abide by the following submission requirements.

Your submission must contain a `customizations.sh` script (located at `vagrant/customizations.sh`) which contains any additional commands that are required to set up a Vagrant environment in order to build your system (install additional packages, etc), as well as a `config.rb` file (located at `vagrant/config.rg`) describing any configuration parameters. These files are the only two that are allowed to be modified. When a submission is tested, all other files will be replaced with default ones, so your team should take care not to modify any more than is required. Specifically, the default account username and password should not be changed from "vagrant", and SSH connections MUST be allowed for the "vagrant" user. As a reminder, attacking teams will not have access to the provisioned vagrant environments that the organizers use to build your design. However, they will have access to all the source code provided to the organizers, so attacking teams are able to build their own versions of your design.

WARNING: Please take care to test that your vagrant scripts work to provision your system. Broken scripts will prevent your team from entering the Attack phase.

4 Handoff Phase

After the completion of the Secure Design Phase, each team must submit their design to the organizers. Source code must be hosted on a public facing git repository (github, gitlab, etc). If you school provides a private git server, you are welcome to use it for development, however, you may be asked to move the source code to a public server if submission problems arise. When you are ready to submit, the organizers will provide an account that you must give access to your source code. The organizers will clone your repository and check out a commit tagged[2] “v1.0”.

NOTE: If using gitlab, the provided user account should be given at least “reporter” access, as any lower level will prevent cloning of your submissions.

Submissions should include all source code, documentation, and supporting files necessary to build your system. All of this should be made available through the modification of the provided `customizations.sh` and `config.rb`, which will allow the administrators to easily recreate your team’s development environment. Remember, these are the only two files that can be changed in the `vagrant` folder.

After receiving a team’s design, the eCTF organizers will provision a system and validate that the system meets the functional requirements described in this document. Any design that will not build or does not meet these requirements will not be able to progress to the attack phase of the competition. The organizers will contact each team with handoff status within two (2) **business** days after a team’s submission, whether it is accepted as functional or not. If a design is not accepted, teams will be allowed to address any problems identified and submit a revised version. Once a team’s design has been accepted, they may not submit further designs (e.g., to patch security flaws that are identified after submission).

Teams will be notified if the optional requirements fail the testing process. If the optional requirements are not met, teams are welcome to enter the Attack phase, but will forfeit the points associated with these requirements. If the optional requirements were expected to pass but did not, teams will be allowed to resubmit their design with any fixes, however, this will count as submitting a revised version.

All source code and documentation, as well as the build environment/Vagrantfile, will be provided to other teams during the attack phase to discourage security-by-obscurity, as well as to accelerate attack development. Exemplary documentation will be worth extra points at the discretion of the eCTF organizers – see Documentation Points for details.

5 Attack Phase

During the Attack Phase, each design that has been validated during the Handoff Phase will be available for attack.

5.1 Materials and Information Available

For each design, the files listed below will be made available to all attacking teams. These files will be created by the eCTF organizers by running the necessary build tools.

- Source code for the music system and provisioning tools
- Documentation
- miPod.BINs for various regions
- customizations.sh
- config.rb

[2] You can create a tag named “v1.0” by running “`git tag v1.0`” once you are ready to submit, and then can push the tag by running “`git push origin v1.0`”. See https://git-scm.com/book/en/v2/Git-Basics-Tagging for more info on git tags.

- Protected songs
- User pins (for accounts that are to be made public)

The final boot images (BOOT.BIN and miPod.bin) will be encrypted and signed by keys only accessible to the organizers.

5.2 Flag Descriptions

Any flag submitted to the score board will be of the form `ectf{<flag_name>_<16 hex characters>}` (except for pin extraction, where the hex characters are the characters in the pin). For example, the "Region Lock" flag would be of the form '`ectf{regionlock_0123456789abcdef}`'.

The following table lists the flags, as well as a description of each:

Flag Name	Capturing this flag proves that you can…	Requirement	To Submit this Flag…
Region Lock	play a song from a region that the player is not provisioned for.	Confidentiality	obtain the flag from the audio sample and submit to the scoreboard.
Unauthorized Play	play a song that your user does not have access to.	Confidentiality/Authentication	obtain the flag from the audio sample and submit to the scoreboard.
Pin Extraction	obtain the pin for a user that you do not have access to.	Authentication	obtain the pin and submit the pin to the score board.
Music Tamper	modify a protected song. Deliver a file of the modified, protected song.	Integrity	provide the organizers with the modified binary.
Custom Music	Protect a provided custom song. Deliver a file of the custom, protected song.	Authentication/ Confidentiality	provide the organizers with the modified binary, and optionally, a custom miPod application.

5.3 Vulnerabilities in 3rd-party Components

The security of each design will depend (at least partially) on the security of several 3rd-party components that are used within the overall system. For example, we expect that all systems will use some open source software libraries; and all systems will depend on some hardware security features within the Zynq chip. Therefore, the question often arises whether these components are in scope for the competition. In general, the answer is "it depends" and any 3rd-party vulnerabilities will be considered on a case-by-case basis.

The First Stage Bootloader (FSBL) provided by MITRE and security features provided by the hardware platform are off-limits and should not be attacked. If you have any question about whether a component is "in scope", please reach out to the organizers for clarification.

Attacks should focus on only the student-designed components of each system. However, students should not consider their design to be immune from vulnerabilities that are inherited by their system as a result of using insecure software libraries that were found on the Internet – as that would never be the case in the "real world".

If you think you've found a vulnerability in any components that were not student-built, please contact one of the event organizers privately. We strongly encourage responsible disclosure[3] if any zero-day[4] vulnerabilities are discovered. If desired, MITRE can help to coordinate the responsible disclosure of weaknesses to appropriate parties and ensure that the vulnerability is handled appropriately.

6 Scoring

Points are scored in one of the six ways listed in the following sections.

6.1 Design Phase Milestone Flag Points

To encourage teams to stay on schedule during the design phase and to give the organizers insight into each team's progress, points will be awarded for reaching certain milestones. Each Milestone flag has a deadline date at which point it can no longer be submitted for points. The milestone flags are:

Milestone	Description / How to obtain the flag	Suggested Submission Date	Deadline Date[5]
Read Rules	If you read all the rules, you'll know	01/17/20	01/29/20
Boot Reference	Provision and boot the reference design on the provided hardware	01/29/20	02/12/20
Play Demo Song	Play the provided (protected) example song	01/29/20	02/15/20
Design Document	Submit a design document containing descriptions of how each command will work on your system	02/05/20	02/26/20
unprotectSong	Provide an unprotectSong script to the organizers that "unwraps" the protections that are added by your protectSong script – outputting the original unprotected song file	02/12/20	02/26/20

Positive Tip!

Don't just do it for us! These milestone steps are helpful for you! For example:

- ***Booting the reference design and playing the demo song will help you learn how to use the board***
- ***Writing a Design Document will help you clarify and organize your thoughts***
- ***The unprotectSong script will be helpful for testing, since it provides a way to verify your protectSong script without the hardware.***

6.1.1 Design Document Details

The design document will serve as a high-level description of the security features of your media player. Specifically, the document must describe your system and song provisioning processes along with your miPod command protocols and how you plan to protect each of the flags. You will not be held to the system description that you submit. If you choose to modify your design after submitting the document, that is OK.

We do not expect incredibly detailed design documents; however, spending time on this will benefit you in two ways: (1) the document will serve as a way to communicate your design to your advisor and team and (2) we will be able to ensure

[3] https://en.wikipedia.org/wiki/Responsible_disclosure

[4] A "zero-day" vulnerability is a vulnerability that is unknown and/or unaddressed by the person/group responsible for maintaining and mitigating vulnerabilities in the target. See https://en.wikipedia.org/wiki/Zero-day_(computing)

[5] The exact cutoff will be at 11:59pm (eastern time zone) on the deadline date

that your design does not break any competition rules. If we feel that your team made little effort to document your intended design, we reserve the right to reject your design document submission.

Submitting this document early will help your team to come to agreement on a design and allow extra time to integrate any feedback we have.

Your design document must be submitted in PDF format.

6.2 Points for Optional Implementation

Each team can earn points for additional functionality implemented into their design. If your team is successfully able to implement each feature, you will receive the following point values:

- **500 points**: Seek functionality (i.e., jump forward/backwards while playing a song)
- **600 points**: Support 48kHz Sample Rate (if you support 48kHz you do not need to also support 22 kHz)

These features will be tested at the time of design submission. Once a team enters the Attack Phase, they cannot resubmit their design to earn additional “Optional Implementation” points.

6.3 Offensive Flag Points

Each system is required to hold and protect “flags” that should only be revealed if the system is compromised. By submitting flags, a team is demonstrating that they have compromised the target system. A brief description is required for each attack that results in a flag submission.

Since the vulnerabilities to be discovered in the attack phase come from other teams’ unintentional flaws in the design phase, the scoring system is designed to adjust points based on difficulty of capture as more information becomes available. The point value of any given flag will be adjusted dynamically and automatically based on multiple factors:

- If multiple teams capture the same flag, then the value of that flag will be divided among all the teams that capture it (distribution is not equal – it is weighted based on time of capture to provide more points to earlier captures). Naturally, more difficult attacks will be executed by fewer teams and therefore rewarded with more points.
  Note: Your total score will drop each time another team captures a flag that you had already captured. This is because the flag points that you are initially awarded need to be re-distributed as additional teams capture the same flag.

  Positive Tip!

  Although counter-intuitive, it may be a good strategy to seek out and spend time attacking the most difficult targets. The most challenging flags will, in theory, be worth the most points in the end.

- The number of points a flag is worth increases over time as it remains un-captured. This will make the difficult flags more and more appealing as the competition goes on.
- To discourage teams from “holding” a flag without submitting it, the first capture of each flag will earn the attacker 100 bonus points.

6.4 Defensive Flag Points

Defensive points will be awarded every 24 hours for each flag that remains uncaptured. The 24-hour timer begins for each team once they successfully complete Handoff. Therefore, a team does not begin accruing defensive points until after submitting a working design.

6.5 Documentation Points

Good documentation will be rewarded to discourage security-by-obscurity. “Good documentation” is meant to describe clear and well-commented code, useful descriptions of modules/functions/classes, and other documents that clearly describe how to read or approach the entire code base.

Positive Tip!

We are not looking for lengthy documents that describe your implementation in excruciating detail. ***A concise and clear README.md, including a brief rationale for your security features, combined with well-structured and well-commented code will be enough for Max points. Quality will be valued over quantity.***

The maximum number of points that can be scored for documentation is equal to the value of an attack flag scored on the last day of the competition, with the actual amount being a percentage of that maximum:

- **Max** Exemplary documentation, comments, and code structure that is clear and easy to understand
- **75%** Good comments and high-level documentation
- **50%** Good comments, but lack of clear high-level documentation
- **25%** Confusing code and little or no actual documentation
- **0%** Confusing or deceptive comments and documentation

Points for documentation will not be awarded until near the end of the attack phase to allow for proper analysis. Honest feedback on documentation from other teams will be solicited and will be factored into the final point determination.

6.6 Write-ups

There will be an opportunity for the top teams to provide up to two write-ups for additional points (one for defense and one for offense):

- The defensive write-up may discuss security measures that worked well, those that could have been improved upon, or any that were planned but could not be developed in the time provided.
- The attack write-up is to award teams that develop interesting or novel attacks which do not directly capture an existing flag.

Further details on the number of teams that may submit write-ups and the content/format of the write-ups will be provided during the attack phase.

7 Award Ceremony

All teams (students and faculty advisors) who submit a working design are invited to an award ceremony at MITRE (Bedford, MA) on April 16, 2020. During the award ceremony, each team will be invited to give a presentation of their work during the design and attack phases of the competition. At this time, the final scoring will be revealed, and awards will be presented.

8 Important Dates

Kickoff --- January 15th, 2020

- Competition officially kicks off

Design Milestone Deadlines

- **See section 6.1**

System Hand-off --- February 26th, 2020

- System design and implementation is due
- Attack phase begins for each team once they successfully submit a working design
- Scoreboard opens for attack flag submission

Scoreboard Closes --- April 10th, 2020

- Flag submission is closed

Award Ceremony – April 16th, 2020

- All participants will be invited to an award ceremony, where teams will present their work and MITRE will present awards

9 Rules

Most rules are described and explained throughout the challenge description in the earlier sections – please read this entire document! This section is intended as a concise summary of the most important rules.

(1) In addition to the rules provided by MITRE, participants should also adhere to all the policies and procedures stipulated by their local organization/university.

(2) MITRE reserves the right to update, modify, or clarify the rules and requirements of the competition at any time, if deemed necessary by the eCTF organizers.

(3) When submitting your secure design, all source code and documentation must be shared.

- This is to discourage security-by-obscurity, as well as to accelerate attack development and encourage more sophisticated techniques for both sides.
- Creating any part of your submission in an obfuscated manner or using an esoteric programming language is considered security-by-obscurity and is not allowed. Please contact the organizers if you have any questions about this.

(4) During the attack phase, you may only attack the student-designed systems explicitly designated as targets.

- For example, the First Stage Bootloader (FSBL) provided by MITRE and security features provided by the hardware platform are off-limits and should not be attacked.
- If you have any question about whether a component is "in scope", please reach out to the organizers for clarification.

(5) All flags must be validated by submitting a brief description of the attack.

- Attack descriptions should be sufficiently detailed to allow the defender to correct their vulnerability.
- eCTF admins may invalidate points for flags that are not validated before the completion of the eCTF.

(6) Flag sharing across teams is not permitted and will result in immediate disqualification.

(7) No permanent lock-outs are allowed. See section Timing and Performance Requirements3.3.9 for timing/performance requirements.

(8) Team sizes are unlimited.

- Most teams will consist of members of varying degrees of experience and skill level. Our hope is that this creates an opportunity for mentoring, where the most knowledgeable team members will help teach and guide the other members of the team. Team advisors should help manage meeting times and organization of large teams.
- We want to encourage as many students to participate as possible, even if they are not willing to commit a significant amount of time to the competition.

(9) Teams may consist of recent alumni and students at any level: undergraduate, graduate, PhD, or a mix.

(10) Your system must work with the hardware that was provided. Switching to a different platform or modifying the physical hardware during the design phase is not allowed. ectf{readtherules_028ffa50133ffb58}

(11) All documents that are submitted (e.g., design document and write-ups) must be in PDF format.

If you have any questions, ask! Our Slack workspace (https://ectf2020.slack.com/) is the preferred method of communication and you will receive the most timely responses there. You may also contact us via email (ectf@mitre.org).

9.1 Eligibility

The eCTF is open to individuals who have registered for the competition with the MITRE Corporation, and are hereby known as Contestants. A Contestant is eligible for Prizes if they are a current high school or college student within the United States and are a United States Citizen. Void where prohibited. All federal, state and local laws and regulations apply. MITRE reserves the right to verify eligibility and to adjudicate on any dispute at any time.

10 Frequently Asked Questions [updated v1.1]

Is it OK to obfuscate our source code to make it more challenging to understand and attack?

No. Obfuscations performed at compile-time (e.g., to make binary reversing more challenging) are OK, but your source code needs to be written in a clear and maintainable fashion. It should be well commented and/or otherwise documented clearly.

Can we add intentional delays during boot or playing music to make it more difficult for an attacker to collect large numbers of observations?

There should not be any intentional delays that may be noticeable to the user because these may negatively impact the user experience and hurt sales for your product. Also, see section Timing and Performance Requirements3.3.9 for timing/performance requirements.

If your system detects that it is under attack, additional delays are OK, but must be limited to no more than 5 seconds. Permanent lock-out or self-destruction is not allowed (see next question).

Is it OK to brick the board when an attack is detected?

No! This is a music player that the customer bought! We can't have it be so easy for an attacker to disable the entire system! Can you imagine the cost of the recalls?!

Can we physically modify the board with countermeasures during the design phase?

No. The boards are provided to teams at the beginning of the competition, so you won't have an opportunity to modify the hardware during provisioning. You may, however, modify the boards for attacks.

How many boards can we get during the attack phase? (e.g., if we keep bricking them, can we keep getting new ones?)

Each team will be given two boards at beginning of the competition. Teams are welcome to purchase additional hardware for parallelizing development efforts; however, only the provided boards will run systems provisioned in the attack phase. If you want to try an attack that may brick the hardware, please contact the organizers to discuss options in the unfortunate event that an attack renders one of your boards useless.

Can we attack another teams' development environment?

No! Everything other than the provisioned boards, the host tools, and the firmware images are considered out-of-bounds. In other words, there is **nothing** that you can attack until your team enters the attack phase.

Is social engineering in-scope for this competition? Can we send phishing communications to other teams to trick them into revealing their secrets?

No, please don't do this. Keep your attacks technical. We love creative ideas, but this one can easily violate university, state, and federal regulations.

11 Appendix A: Audio Notes

11.1 Audio File Format [updated v1.1]

The audio files used in the competition will be mono channel, 16-bit depth Pulse Coded Modulation (PCM) WAV files sampled at 48kHz for the bonus implementation, or 22 kHz for the standard. Songs will be at least 30 seconds in length and will not be larger than 128 MB.

12 CHANGELOG

12.1 v1.1(1/29/20)

- Indicate that the latest version of Vivado may be used if the organizers are notified by 2/12/20.
- Update section on song sharing to indicate that sharing is persistent across reboots.
- Update FAQ on additional delays to indicate that delays must not be longer than 5 seconds per delay, as opposed to per boot.
- Update Appendix to indicate that audio is mono channel, songs will be at least 30 seconds in length, and will be at most 128 MB in size.