FISSURE: The RF Framework for Everyone

Christopher Poore poorec@ainfosec.com
Assured Information Security, Inc., 153 Brooks Road, Rome, NY 13441 USA

Abstract

FISSURE (Frequency Independent SDR-based Signal Understanding and Reverse Engineering) is a newly released open-source RF and reverse engineering framework designed for all skill levels with hooks for signal detection and classification, protocol discovery, attack execution, IQ manipulation, vulnerability analysis, automation, and AI/ML. This paper introduces the principles behind FISSURE and provides a synopsis of its components. Additionally, the extent of GNU Radio integration within the framework is detailed along with a projection of future directions for the project.

1. Introduction

RF devices are everywhere and provide access to cyber physical systems, connecting the digital world to the physical environment in the form of: vehicles, unmanned aerial systems, communications networks, industrial control systems, medical devices, weapon systems, etc. In today's world the "smart" label means connected and these connections reveal additional attack vectors. This proliferation of new potential pathways and exposure to vulnerabilities from upgrades to various technologies is producing an ever-growing demand for combined RF and Cyber solutions across wide-ranging applications.

There is a common vulnerability analysis process for RF-enabled systems that cybersecurity experts repeat frequently when working with new devices and RF protocols. This includes detecting the presence of RF energy, understanding the characteristics of the signal, collecting and analyzing samples, developing transmit and/or injection techniques, crafting custom payloads or messages, and investigating the effects complex payloads can impose on targets. There are a series of challenges in this process which can provide a steep learning curve for most people – whether it is battling issues with software dependencies, software updates, hardware compatibility, configuration, or re-creating past work.

FISSURE is an open-source RF and reverse engineering framework designed to speed up the characterization of signals and the identification of vulnerabilities in RF protocols, waveforms, and devices. It is a solution to address many of the challenges that come from working with known and unknown signals/protocols, and also serves as a means to publicly consolidate existing RF solutions that get repeated many times over by researchers, developers, and hobbyists.

FISSURE has grown considerably from its beginnings as a prototype which demonstrated aspects of a meticulously planned out modular framework that enables rapid RF device assessment while simplifying development and integration of third-party tools and hardware. Then following years of investment by AIS through internal research and development, the prototype grew into an in-house laboratory tool that acted as a workflow enabler with a scripted installation, tool storage, library development, and additional hardware support. Now, as an open-source project, FISSURE consolidates all-things RF in the form of software modules, radios, protocol information, signal data, scripts, flow graphs, reference material, and third-party tools. It offers a means to prototype techniques and acts as a standardized interface to interact with other open-source tools and one-off solutions.

FISSURE is intended for both experts and beginners. It offers an out-of-the-box, pain-free software installer built with transparency. It is written mostly in Python and PyQt with support for legacy systems. Users can edit the pieces on their own to expand functionality and use it in everyday testing. Helpful guides exist to make it easier to interact with the framework and understand the different RF technologies contained therein. FISSURE seeks to draw its strength from the community through feedback and contributions and have a future role in education, research, and improving everyday work.

1.1. Principles

For an RF framework to be truly valuable it needs to encompass several technologies and capabilities and always be in a state of expansion. It must be flexible enough to support new features and the rapid integration of the latest tools and algorithms to keep up with the pace of innovation. FISSURE will be constantly evolving to assist as many people as possible, but it will also try to adhere to the

founding principles that make it technically sound and valuable to users.

The core technical principles of FISSURE are to help speed up signal characterization and help with the identification of vulnerabilities in protocols and devices. That concept is very wide-ranging and it allows for simultaneous maturation of other surrounding topics. For example, FISSURE can be a testbed for AI/ML and automation in several technical areas including signal detection, feature extraction, protocol/emitter classification, demodulation, pattern recognition, data analysis, batch processing, vulnerability analysis, and more. However, this comes with the understanding that the technical foundation needs to exist before automation can have a larger role.

The hardware considerations for FISSURE must allow accessibility for the average user while having the potential for expansion to improve performance. This means utilizing commercial software-defined radios (SDRs) and other commonplace hardware, but also maintaining the ability to support custom radios and standards such as VITA 49. The ability to pass data and commands over a network and between software components is essential to offload processing, operate out of more than one geographic location, and to support additional platforms and hardware.

FISSURE is meant to be a framework for everyone and that requires it to have footholds in both simplicity and complexity. The simple aspects are the reliable installation, instant access to commonly performed operations, easily modified code, helpful visualizations, support for the latest and legacy, examples and guides on how to do things, and the consolidation of tools and techniques. The complex aspects are the development of cutting-edge techniques and the integration of advanced solutions. The framework will never fully be complete as there will always be pieces that can be improved or added over time.

1.2. Comparisons to Similar Products

Signal characterization and vulnerability identification challenges have been around for a long time. Presently, there are a large number of concurrent solutions being developed by an unimaginable number of entities spread across the globe to address modern challenges. FISSURE may share several technical components found in such solutions but it also contains a unique combination of elements that set it apart.

The biggest selling point that distinguishes FISSURE from other very capable products is that FISSURE is free and open. It supports modifications to the source code and lets users work with affordable commercial off-the-shelf (COTS) hardware. It can also render an instant sense of familiarity by providing quick access to third-party tools and allow users to explore software for similar technologies of interest to which they may have previously been unawares. FISSURE gets a lot of its power by not reinventing the wheel and supports the use of existing applications where possible.

FISSURE is not a virtual machine (VM) or a dedicated operating system. It contains all the commands for installing a large amount of third-party tools tested against multiple Linux operating systems. While FISSURE avoids some of the hardware/processing limitations and other headaches associated with VMs, it does not create a sandbox during installation either. As a result, there can be compatibility risks with existing software already installed on a machine which may cause unexpected errors. However, the installer is a nearly pain-free method for staging computers and can be easily modified to add or subtract software items.

The extreme amount of flexibility and modularity that FISSURE contains allows it to encompass such a wide variety of applications. This variety along with the Python and GNU Radio code base make it a great option for introducing users to several programming and RF concepts. The user dashboard with its visualizations and lessons reduce the otherwise steep learning curve. The project also has the added bonus of being managed by a company that has developers working on the forefront of cybersecurity and in touch with a number of technical and professional communities.

2. Framework Components and Features

FISSURE is comprised of dedicated Python components communicating to each other over a central hub as shown in Figure 1 (the central hub is nicknamed “HIPRFISR” because it is not a real hypervisor in the traditional sense). The primary means for passing messages between components is performed via the open-source universal messaging library ZeroMQ (ZMQ). Messages can be assigned source identifiers and categories such as heartbeats, status, or commands. Each message has a YAML schema that assigns the number of expected parameters and the names of callback functions that get executed upon reception. Each component establishes a ZMQ DEALER-DEALER pair to the central hub for issuing/receiving commands. Additionally, each component has a ZMQ PUB socket and any number of SUB sockets for issuing/receiving one-to-many status messages.

The central hub receives commands from the user dashboard, coordinates actions to the other components, manages automation, and contains functions for editing the main library. Additional dedicated software components that perform new features can easily be added to the framework if 1) there are clear inputs and outputs that can be defined and 2) the possibility exists to create a simple wrapper that

manages the ZMQ connections.

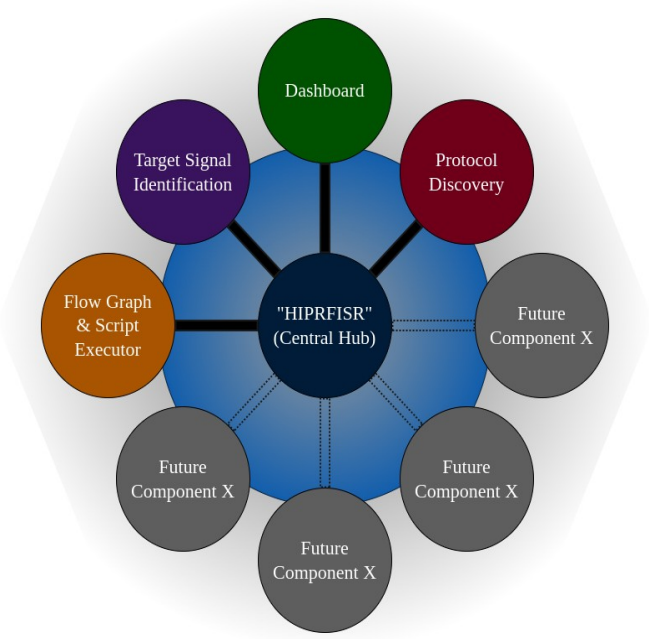

Figure 1: Dedicated FISSURE Components

The user dashboard consists of several tabs, menu items, and buttons to quickly assign hardware (SDRs, Wi-Fi adapters, IoT analyzers, etc.) to a particular functionality. Interaction with the user dashboard is the singular means to control the functionality within the tabs and to initiate commands to each component. Within the dashboard, there are menu items for the following: launching standalone GNU Radio flow graphs that are not tied to the rest of the software in the framework; quickly accessing third-party and online tools organized by protocol or application; lessons for learning more about relevant technologies; and help pages for operation, development, protocol reference material, calculators, and hardware instructions.

The dashboard has utilities for modifying the FISSURE library (YAML) – which contains protocol definitions, flow graph information, and signal archive metadata. There are utilities for browsing; searching; uploading images; and adding/removing modulation types, packet types, signals of interest, statistics, demodulation flow graphs, and attacks. The library is constructed to easily support the addition of individual plugins or proprietary add-ons to segregate data sensitive features from the public repository.

2.1. Target Signal Identification

The Target Signal Identification (TSI) component is intended to run four subcomponents: a detector, a signal conditioner, a feature extractor, and a classifier. The purpose of the TSI component is to detect signals of interest (SOIs), isolate and condition signals for detailed analysis, extract signal characteristics for protocol and/or emitter classification, and apply user-specified AI/ML classification techniques. The TSI component will result in additional knowledge of the surrounding RF environment and pass potential SOIs to the Protocol Discovery component.

As of Fall 2022, the TSI component only contains a slow-scanning detector which reports back power, frequency, and time values for signals above a power threshold (Figure 2).

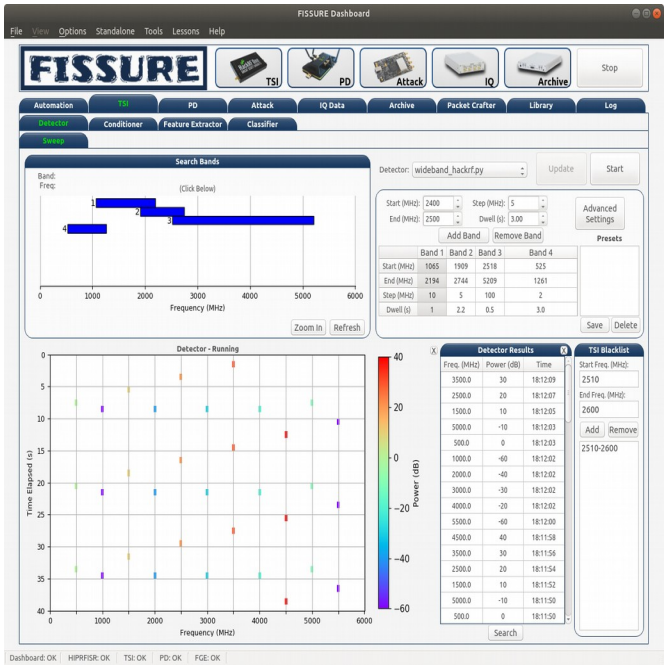

Figure 2: TSI Wideband Detector

2.2. Protocol Discovery

The Protocol Discovery component is responsible for identifying and reversing RF protocols to help extract meaningful data from unknown signals. It is designed to: accept SOI information, iterate flow graphs to perform recursive demodulation techniques, deduce protocol methods, assign confidence levels, analyze a bitstream, calculate cyclic redundancy check (CRC) polynomials, create custom Wireshark dissectors, and automatically add protocol information to the FISSURE library.

As of Fall 2022, Protocol Discovery is an entirely manual process with no recursive demodulation techniques in place that work towards producing a bitstream. The bit slicing capabilities are best suited for fixed-length messages. The data viewer can perform bit-wise operations, convert between binary and hex, view ASCII text, and compare data against known packet types already present in the library. The custom Lua Wireshark dissectors are designed to monitor for individual packet types assigned to a protocol

that are inbound on designated UDP ports (the data is produced from demodulation flow graphs). The CRC calculator applies common CRC algorithms to data and can deduce certain polynomials from two messages with known CRC values.

2.3. Attacks

The Flow Graph/Script Executor component runs flow graphs or Python scripts to perform single-stage attacks, multi-stage attacks, fuzzing attacks, IQ recording and playback, live signal inspection/analysis, and transmit playlists of signal data constructed with files downloaded from an online archive. Attacks are organized by RF protocol, modulation type, hardware, and type (denial of service, jamming, sniffing/spoofing, probe attacks, installation of malware, misuse of resources, file). While FISSURE is intended for wireless applications, attacks can be performed against wired application or any network protocol in general.

Single-stage attacks can be in the form of Python2/Python3 scripts and GNU radio flow graphs with/without GUIs. The Python script attacks require a simple header to be added to the file that specifies the default values (Figure 3) for the attack variables. Python scripts and flow graphs with GUIs run as-is and do not accept updates from the dashboard. The flow graph attacks that do not utilize GUIs can change attack variable values before and during runtime (see GNU Radio Integration).

Multi-stage attacks string together a series of single-stage attacks that get run on repeat for a set duration. Fuzzing attacks can be in the form of data field fuzzing or flow graph variable fuzzing. Data field fuzzing allows the user to specify a particular packet type to fuzz and choose which fields to fuzz (sequentially or randomly) and the ranges for each field. Messages are generated at a configurable interval and the CRCs are automatically calculated for each new message. Flow graph variable fuzzing allows a user to load a flow graph and choose which variable to fuzz.

As of Fall 2022, attacks are not evenly distributed across all possible hardware types. Certain attacks are hardware-specific and require modification to GNU Radio blocks to replicate intended behavior. Not all protocols have fuzzing capabilities.

2.4. IQ Manipulation

The IQ Data tab contains several functions for working with IQ data (Figure 4). There are live inspection flow graphs for manual inspection; capabilities for record and playback; a data viewer with plot, zoom, pan, save, and measure capabilities; data modification capabilities to include: cropping, converting between data types, appending, applying timeslots, overlapping signals, resampling, normalizing; and some analysis capabilities in the form of magnitude plots, instantaneous frequency, spectrogram, FFT, moving average filters, Morse code deciphering, and polar plotting.

Figure 3: Single-Stage Attack Example

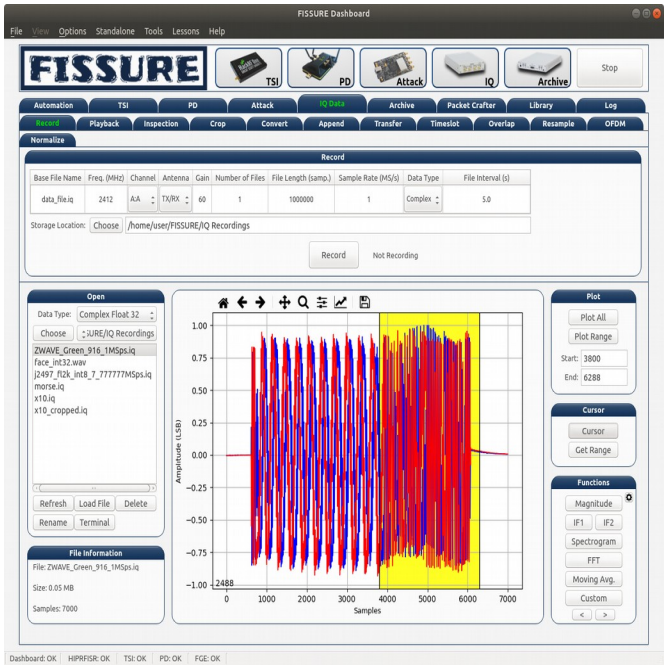

Figure 4: IQ Manipulation

2.5. Online Signal Archive

AIS has begun storing IQ signal data to an online signal archive located at fissure.ainfosec.com to reduce the size of the GitHub repository and to not force extraneous amounts of data upon users. The FISSURE library holds the metadata for each Archive file and the values can be viewed right from the user dashboard. The user has the ability to download specific files of interest from the Internet with a single click. The Archive tab contains replay capabilities which allows users to create playlists to simulate traffic and test systems. This is also a convenient way to test the front-end signal components (TSI, Protocol Discovery) for FISSURE.

2.6. Packet Crafting

Custom packet crafting capabilities exist for protocols with packet types entered into the FISSURE library. This allows users to browse sample messages, change field values at the bit-level, recalculate CRC values, construct sequences of messages, and save data to a file. The Packet Crafter tab also includes Scapy integration for transmitting different types of 802.11 packets while in monitor mode.

2.7. Third-Party Tools

The third-party tools installed as part of FISSURE are mostly self-contained as menu items in the form of standalone flow graphs, tools, and help items. There are some GNU Radio out-of-tree modules and a few other tools that can be found in Protocol Discovery and some attacks. The standalone flow graphs are favorites that can be quickly accessed and are separate from the rest of FISSURE. They will retain their last state upon reloading FISSURE. The tools and help menu items will launch programs; open a terminal with example commands; or open a browser to reference material such as maps, calculators, and databases.

2.8. Lessons

Lessons and tutorials for interacting with various RF technologies and tools within FISSURE are provided to users as Markdown/HTML pages. The goal is to teach new concepts and help refresh users on how the technology works by always having a set of steps available that can be utilized for quick reference. This is an area that is expected to grow as FISSURE expands its role in education and receives feedback from the community. As of Fall 2022, topics include: OpenBTS, Lua dissectors, Sound eXchange, ESP boards, radiosonde tracking, RFID, data types, custom GNU Radio blocks, TPMS, ham radio exams, and Wi-Fi tools.

3. GNU Radio Integration

GNU Radio flow graphs are found throughout FISSURE in the form of detection, inspection, protocol discovery, demodulation, sniffing, recording, replay, attacks, fuzzing, and third-party tools. They utilize data and interact with FISSURE components in many different ways as listed below:

- The wideband detector flow graphs launch and then accept commands to update parameters like frequency, sample rate, FFT size, threshold level, and gain. They return power, frequency, and time values periodically over the network via a ZMQ PUB socket.
- The inspection flow graphs contain GUIs with widgets for changing variable values within blocks that contain callbacks. As of Fall 2022, the variable values do not get modified before runtime so parameters like serial number are not utilized.
- The Protocol Discovery component currently only demodulates a limited set of protocols and packet types. The ones that do exist produce a bitstream that gets forwarded over the network to a circular buffer for further analysis. These demodulation flow graphs will act as one of the final stages in a recursive demodulation process.
- Sniffing flow graphs tap into the demodulation flow graphs (via streams, tagged streams, and messages/PDUs) and pipe the data into Wireshark for live viewing of messages and recording of traffic sessions.
- The IQ Data tab contains recording flow graphs for quickly saving data to a file and viewing the contents instantly. It also has the ability to record several files at a set interval. This is useful for hands-free recording operations. Playback flow graphs will replay a file a single time or on repeat.
- The archive replay flow graphs load data into a file source and are run individually as part of a playlist that has the option to repeat indefinitely.
- Attack flow graphs can be run with or without a GUI. Flow graphs without GUIs can have their variable values changed before and during runtime.
- Fuzzing flow graphs contain a special fuzzer block that accepts the parameters from the user dashboard and reads in the protocol information from the FISSURE library file to adjust fields and calculate CRC values. The output of the fuzzer block is a message that gets updated at a regular interval that can be fed into other transmit/modulation blocks.
- The ability to fuzz individual GNU Radio variables for blocks with callbacks is built into the Attacks tab.
- Third-party flow graphs/tools are typically in the

form of standalone flow graphs and compiled Python files that get run from a terminal.

Running flow graphs with and without GUIs from Python is an important distinction due to the way in which the components control the flow graphs. Flow graphs without GUIs are first loaded in Python using the "*__import__()*" command. The text data for the variable default values is modified prior to issuing a "*compile()*" command and loading the altered result as a new module. The new flow graph is then loaded with the "*getattr()*" function and used with the conventional GNU Radio "*.start()*", "*.wait()*", and "*.stop()*" commands. Changing flow graph variables during runtime is done using the combination of "*getattr()*" and "*set_<variable>*" calls. These steps cannot be reproduced without error when flow graphs are compiled with GUIs enabled. The default values for these flow graphs are not modified and changes during runtime are done through the callbacks from the GUI elements.

As of Fall 2022, FISSURE is divided into three branches to reduce code redundancy and better support legacy versions of Python, GNU Radio, and PyQt. These branches are Python2_maint-3.7, Python3_maint-3.8, and Python3_maint-3.10. The out-of-tree modules installed with FISSURE are in the form of Git submodules that get cloned directly from online repositories. These modules will need to be monitored in case branch names change or updates change the functionality in unexpected ways.

6. Conclusion and Future Work

FISSURE is a new framework that has a good starting foundation which offers several features that can be utilized as-is by many types of users. With further development and support from the community, FISSURE will expand beyond a few examples of what can be done with an RF framework into a very extensive tool for working with RF and performing reverse engineering techniques.

As an open-source project, showing interest in FISSURE is vital to its success. By starring the project on GitHub, joining the Discord server, following on Twitter, it will make for an easier sell to internal/external customers. Contacting the developers, encouraging collaboration, and submitting contributions is sure way to speed up FISSURE development and aid your own projects at the same time.

Growing a strong user community will strengthen the software and expand the breadth of technologies it encompasses. Feedback is vital for steering the direction of the project and helping others who share similar experiences. The developers will always welcome suggestions for software tools, hardware suggestions, IQ analysis algorithms, attacks scripts, new operating systems, bug fixes, and any other improvements.

6.1. Moving Forward

The project will continue to encourage community collaboration, expand capabilities, and push for funding avenues to expedite development. The initial phase, which is still underway, intends to make the software as open and transparent as possible by supporting more types of hardware; finding better ways of utilizing GNU Radio; supporting more operating systems; producing a Docker alternative; integrating popular tools; and releasing documentation for user guides, instructional material, and APIs.

Short-term development will focus on improving the existing software (bugs, cleaning the code, testing more SDRs, etc.) and expanding base capabilities that will round out the framework and allow for task automation and the introduction of machine learning techniques. This also includes the creation of additional lesson material and trialing it in classroom environments such as labs, high schools, colleges, clubs, workshops, and RF/Cyber/Hacking events.

The biggest holes that need to be filled are related to the following topics: integrated fast-scanning signal detection, signal conditioning, feature extraction, protocol/emitter classification, recursive demodulation using flow graphs, protocol identification using digital data, vulnerability analysis against targets of interest, IQ measurement and filtering, and building up the signal archive using a standardized metadata format like SigMF.

Long-term goals include establishing more ties with education to promote the combined realms of RF and Cyber, exposing cutting-edge solutions from experts, expanding to more RF protocols and applications, improving visualization, comparing machine learning techniques, and advancing towards a generic sensor node deployment scheme to operate from multiple geographic locations.