Function Identification and Recovery Signature Tool

Angel M. Villegas
[1]Malware Research Team, [2]Cisco - Talos

This paper was presented at Botconf 2016, Lyon, 30 November – 2 December 2016, www.botconf.eu
It is published in the Journal on Cybercrime & Digital Investigations by CECyF, https://journal.cecyf.fr

Abstract

Reverse Engineering benign or malicious samples can take a considerable amount of time and new samples are created at an alarming rate. Leveraging disassemblers, like IDA Pro, a reverse engineer can analyze the same routines across several samples over the lifetime of their career. Their knowledge is not easily transferred to similar samples or functions for themselves or others.

In particular we can consider the problem code reuse has on reversing efforts, whether it is via statically-linked libraries or integrating existing software. In this paper we want to provide a solution for transferring knowledge to similar functions by introducing a new reverse engineering tool, named FIRST (Function Identification and Recovery Signature Tool), to reduce analysis time and enable information sharing.

Keywords: FIRST, reverse engineering, disassembly analysis, code reuse

1 Introduction

To understand the capabilities of malware, discover vulnerabilities in software, or to make software interoperable engineers leverage reverse engineering and the tools of the trade.

Currently a reverse engineer will open a sample in an analysis tool/framework to look more in-depth at the sample. For instance, let's say we have an unknown Microsoft portable executable (PE) believed to be malicious. To learn more about the different code execution paths we would open the sample in IDA Pro. IDA will display the disassembly and we can start looking through the functions. IDA, via FLIRT signatures, will label some know library functions thereby reducing the number of functions to be analyzed. However, many samples contain tens/hundreds/thousands of functions not labelled by IDA. After analyzing that sample an analyst will understand what those functions do and have added helpful annotations to IDA (function name, prototype, argument names, and function comments). Once the analysis is complete the engineer will move on to another sample and another and another… Eventually functionality seen or previous analyzed will be seen again, either due to the generic nature of the disassembly, code reuse (copy/pasting) or evolution of the same malware family.

Whether it is time, money or intelligence; resources are wasted in this process. In this paper we will discuss FIRST, a cross tool/platform analysis framework to prevent wasting the resources in the future.

2 Problem and a solution

The main idea behind FIRST is to preserve an engineer's analysis of a function (name, prototype, comment) and reduce analysis time by restoring function analysis. A function's analysis can be used to provide helpful information for future reversing efforts. In addition, many research areas dive into ways to find similar functions that are not one to one mappings of functions. By using methods like opcode hashing, mnemonic hashing, locality sensitive hashing, etc., we can find variations from the function originally analyzed.

Starting from this observation, FIRST has four main goals. First, we want to obtain an analyst's function annotations. However, just storing annotations will not aid us in accomplishing our other goals, thus we will need to gather additional data unique to the function such as opcodes and used APIs. Second, we want to store and recover annotations saved. Third, we want to use the stored information to generate ways to find similar functions in which the metadata can be applied. The goal is to provide quick lookups for similar functions. Finally, we want to receive opcodes for a function, use methods for detecting the same or similar functions, and to return a list metadata that can be applied to the function.

Practical application would involve integrating into a reverse engineer's workflow, FIRST would need to accomplish four important goals: 1) obtaining function metadata and implementation details, 2) exposing an API for performing various operations (adding or updating functions), 3) applying various algorithms to normalize or transform the function's implementation for matching, and 4) provide extensible framework to allow new algorithms for detecting similar functions. From an architectural point of view FIRST consists of three components:

- An application programming interface (API) or application binary interface (ABI) for developers to integrate FIRST into preexisting products or workflows.
- Integrations into analysis workflows. For example incorporate FIRST into the industry standard disassembler using an API to create a plugin.
- A server framework that can provide the interfaces and integrations with a REST API along with a plugin scheme to include new modules.

3 Implementation

To achieve the three components previously describe we need to create a client-server framework, see Figure 1. The server framework provides a REST API, an authentication model, database manager and engine manager. A database manager provides flexibility to integrate new data sources. The engine manager organizes and executes all modules used to derive function similarity, called an Engine. An engine can utilize any of the available data sources.

The client side provides end users with a method for registering and obtaining the integrations they elect to use. Once registered an

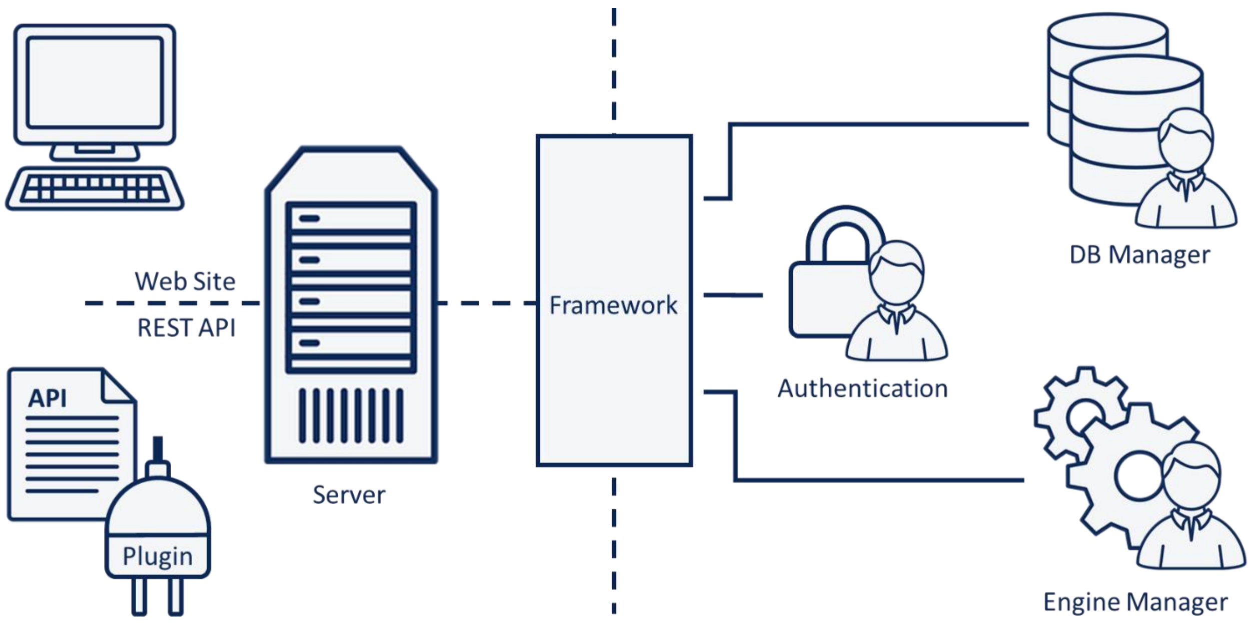


Figure 1: FIRST framework overview

end user receives an API key and can start using the API/ABI or integration. Each integration may bring different capabilities due to the nature and available features of the tool or analysis framework used. The basic functionality that should be exposed is for an end user to:

- Add function annotations (function name, prototype, comment)
- Check for function annotations
- Update function annotations applied
- Manage annotations created

3.1 Server framework

The framework incorporates a web service for registration and interacting APIs. Once registered, the user will receive a unique API key that is required by the interfaces and integrations. The API/ABI interacts with the server via an exposed REST API. Any data sent to the server will enter the modular framework. The framework includes several components: a web REST API (responsible for providing a RESTful API, validating input, and returning results to the client); an authentication module (validate logins via the web user); a database (DB) manager (responsible for providing a layer of abstraction for engines to interact with various databases without implementing specific functionality in the engine themselves); an Engine manager (responsible for interacting with the various installed engines and DBs to add new function metadata or retrieve it).

The framework supplies an abstract DB class

Figure 2: Engine manager

for developers to create their own DB modules that can be incorporated with the engines. The default installation of FIRST includes a DB module for getting data from the database FIRST uses to store all of its information. Developers do not need to use FIRST's database but can leverage completely different databases for storing data. This allows FIRST to be integrated into pre-existing workflows without engineering a completely new system.

The framework supplies an abstract Engine class for developers to create their own engines to expand and enrich FIRST's capabilities. The Engine Manager (see Figure 2) will dynamically load installed engines. Once installed, the engine will receive incoming function metadata and add to the engine's system for storing relevant information (whether it leverages a DB or another means of storage). Engines are given very specific input and expect specific output for the Engine Manager to handle requests from the client.

The abstract Engine Class defines what methods and class variables are required by the Engine Manager and includes many wrapper functions to ensure correct data is returned to the Engine Manager. The Engine Manager will initialize all installed engines by passing them the DB manager. If the DB manager contains the database connections required by the engine then the engine is operational and added to the list of engines the framework will use for processing requests from the client. However, if the engine does not have the required connections or dependencies, then the engine will be excluded from the framework's list of engines.

An Engine is a developer creation and treated as a black box by the Engine Manger. Each engine is required to implement two required methods (Add and Scan as labeled in the diagram) and can implement two optional methods (install and uninstall). Add and Scan methods are given predefined input and outputs standardized data. Engines are given function opcodes and perform some algorithm to create meaningful data for comparing with other function opcodes. However, engines do not have to generate data, for example an engine could look for certain fixed cryptographic constants and return metadata based on those values.

3.2 Client integrations

The APIs and ABIs provide developers with a way to incorporate FIRST into their current workflow. However, many research engineers will leverage industry standard tools. To meet our goal of integrating into a reverse engineers' workflow we are targeting industry standard tools, i.e. Hex Rays' IDA Pro. We will focus on FIRST integration into IDA Pro to show an example integration.

3.2.1 IDA Pro Plugin Requirements

Hex Rays actively develops IDA Pro and continues to build out its plugin APIs and environment. Due to this and the patches in various UI changes FIRST requires IDA Pro 6.9 or higher. FIRST is an IDA Python plugin and requires Python module requests. For easier installation, install Python 2.7 from python.org. In that case you can install

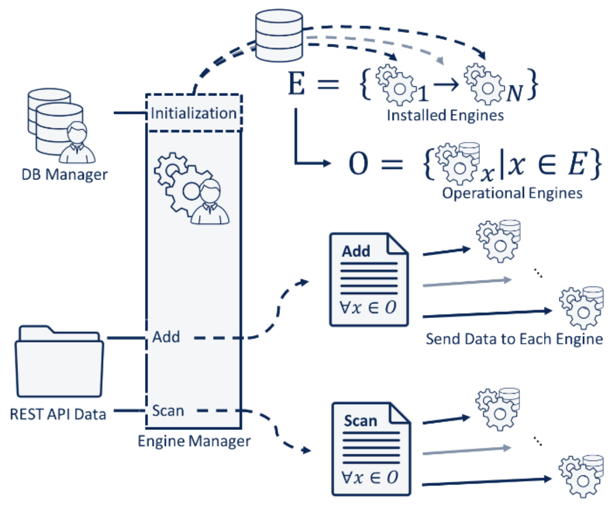
FIRST's requirements with `pip install requests`. If you use the Python installed with IDA's installer you will receive Python 2.7.6 where as the current version is Python 2.7.12.

Additionally, users must register with a FIRST server to get an API key.

3.2.2 Integration – IDA Pro

Since IDA provides a rich analytical environment the IDA Python plugin provide the following capabilities

- Add annotations (single or multiple functions)
- Check for annotations (single or all functions)
- Update applied annotations
- View applied annotations
- View annotation history
- Manage metadata
- Script FIRST with IDA Python



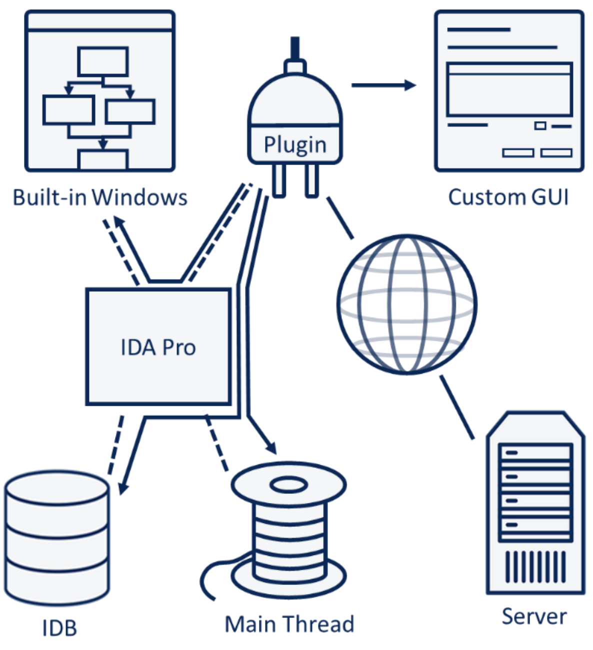


Figure 3: IDA Pro integration

Moreover it is also possible to insert tables inside csdfdsfsdfsdfsd

4 Related work

There have been several attempts to create collaborative frameworks for IDA Pro and match similar functions. IDA Sync [1], collabREate [2], BinCrowd [3], IDA Toolbag [4], SolIDArity [5] are just a few examples of collaborative frameworks build for reverse engineers in IDA Pro. Several of these projects focus on several analysts reversing the same sample or set of samples together. For this reason they focus on trying to keep everyone's view in IDA similar to each other and reflect user contributions to various levels. Unfortunately, some frameworks are based on the sample and the project (a collection of samples). The same function appearing in another sample will not match previous analytical efforts.

CrowdRE [6], FCatalog [7] and Kam1n0 [8]

Unfortunately, all projects (with the exception of Kam1n0 and FCatalog; SolIDArity has not been released yet) have not been maintained, no longer available, or are only compatible with older versions of IDA Pro and no other analysis framework

5 Future work

Currently a Python API exists for interacting with the server and we are expanding this to a full

ABI with a Radare2 integration in development. All code pertaining to the project is available on GitHub[1].

Acknowledgment: The author would like to thank the following people for their help Alain Zidouemba, Doug Goddard... This research was funded with the support of Talos' Malware Research Team (Cisco Systems, Inc.).

Author details

Angel M. Villegas

Talos – Cisco Systems, Inc.
8135 Maple Lawn Blvd. Suite 100
Fulton, MD 20759
anvilleg@cisco.com

References

[1] P. Amini, "IDA Sync," https://github.com/nihilus/ida-sync-plugin

[2] C. Eagle, "CollabREate," *The IDA Pro Book*, chapter 23, http://www.idabook.com/collabreate/.

[3] S. Porst, "ShaREing is Caring - Announcing the free BinCrowd community server," *Zynamics Blog*, https://blog.zynamics.com/2010/03/25/shareing-is-caring-announcing-the-free-bincrowd-community-server/

[4] B. Edwards and A Portnoy, "Toolbag" *Recon 2012*, https://recon.cx/2012/schedule/events/250.en.html

[5] M. Gaasedelen and N. Burnett, "Sol[IDA]rity," https://solidarity.re

[6] A. Meyers, "CrowdRE: Alpha++ Release," *CrowdStrike Blog*, https://www.crowdstrike.com/blog/crowdre-alpha-release/

[7] Xorpd, "FCatalog," http://www.xorpd.net/pages/fcatalog.html

[8] S. H. H. Ding, B. C. M. Fung, and P. Charland, "Kam1n0: MapReduce-based Assembly Clone Search for Reverse Engineering," In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16), p. 461-470.

[1] http://github.com/vrtadmin/FIRST