A Review of Popular Reverse Engineering Tools from a Novice Perspective

A. Lee, A. Payne, T. Atkison
Computer Science Department, University of Alabama, Tuscaloosa, AL, USA

Abstract—Many tools are available to help reverse engineers get a deeper look at a software artifact. For a novice, these tools might be formidable and difficult to understand. Additionally, no survey has been done of reverse engineering tools for several years. In this paper, tools a newcomer might find themselves choosing between are discussed from a novice’s perspective, so as to assist future novices in choosing the proper reverse engineering tool for their task. In examining the tools, usability, functionality, availability, and durability of the tools were examined, among other aspects. Overall, IDA Pro was found to be the most user friendly and capable of the tools.

Keywords: Reverse Engineering Tools, Survey, Software Engineering.

1. Introduction

Software systems, particularly older systems, often require maintenance. However, the people maintaining the system are not always the ones who designed it. Furthermore, systems often lack full documentation, and sometimes the code may be the only ‘documentation’ that exists [1]. Missing documentation may be lost, pieces of the system may have never been documented, or the system may be behaving in unexpected ways which a simple debugging cannot explain [2].

In such cases, reverse engineering is often used. Reverse engineering is the process of analyzing software to identify its processes and their relationships with one another, and create a representation of these processes and relationships [3], [4]. As the size and complexity of programs continues to grow, reverse engineering tools have become more important than ever as a method of understanding source code and binary files that would be difficult to comprehend without assistance [5].

Experts in reverse engineering have their own tools, and know how to use them [6]. However, novices are faced with an array of unknown tools, each of which have unique abilities and disjoint properties even when they fall under the same tool category [7]. This study looks at understanding some of the most popular reverse engineering tools in use today from the perspective of a novice reverse engineer, in order to more easily understand what modern reverse engineering tools lack, and what they succeed at helping novice users comprehend.

To properly evaluate these tools, a set of guidelines was developed to compare dissimilar tools across a few standard points. These guidelines are not meant to be comprehensive, but are meant to capture the points a newcomer to the tool would notice immediately or upon their first usage of the tool as a measure of novice user satisfaction.

By reviewing the tools from a novice’s perspective, differences between the tools and how these relate to user satisfaction can be found, and suggestions can be made to help make new and current tools more user friendly to novice users.

The paper is organized as follows. Section 2 discusses the background. Section 3 holds the methodology. Section 4 contains results. Section 5 holds the discussion. Section 6 discusses the threats to validity. Section 7 concludes the paper.

2. Background

Reverse engineering has been assisted by tools almost since its inception, as tools greatly help a developer understand code, whether it be source code or binary [1]. Tools allow users to visualize code, examine items that would otherwise not be human-readable, and generate an understanding of the code; in some cases, the tools even attempt to reverse or alter the code themselves [2]–[4]. Reverse engineering saves companies money over engineering a new system from scratch; therefore, it is not surprising that it has long been supported by tools. If reverse engineering required many man-hours of manual work, then it would not be a money-saving technique [3]. Instead, companies would simply develop new tools whenever technology changed.

This is not to imply that reverse engineering, even with tools, is a simple task. Specialized knowledge is required to understand the output from most reverse engineering tools. Additionally, tools can often only semi-automate the process of reverse engineering, requiring the developer to do some, or even most, of the work manually [1], [2], [7].

Reverse engineering can help in many different scenarios. The best-known scenario is that in which reverse engineering is used to rediscover or translate legacy software, particularly in cases where the software is complex or large [1]. However, reverse engineering can be used in any phase of the development stage [3], [4]. For example, if a newcomer entered a poorly-documented software project under development, they could use reverse engineering tools to obtain a basic understanding of the project. Defect detection can also be accomplished through reverse engineering techniques; an engineer can use the tools or techniques to play the part of an attacker searching the
compiled code for defects, and hence detect them before the software hits the market [3], [4].

There was a boom in reverse engineering tools in the late 90’s due to the oncoming threat of Y2K. For a time, there was a real need for reverse engineering tools to fix the date issue in legacy software so that the software would keep running past Y2K [1], [4]. Many popular tools, such as Rigi, a visual reverse engineering tool, died out shortly after the event [7], [8]. However, new reverse engineering tools have risen up to fill the gap of the defunct older tools. Since Y2K, reverse engineering has helped adapt old code to new technology, including the Web, and has saved industry millions, if not billions, of dollars over creating new applications every time new technology appears [1], [4]. Modern tools include program analyzers, architecture and design recoverers, and visualizers, and have been used to reverse engineer UML diagrams [9], visualize web tools [10], and translate programs across languages, among other applications [4].

Some of the most popular tools used in modern reverse engineering are dynamic debuggers and disassemblers such as IDA Pro [6], which allow a user to step through processes one line at a time as well as recreate the assembly code based on the bytecode. IDA Pro also allows a user to visualize a program with a graph. As visualization is a vital first step for system maintainers with no knowledge of the code, this is an important attribute [2]. Visualization can also help novice engineers to begin to grasp the program’s shape. Other tools, such as process explorers, allow the user to look inside running code and see how the code is linked together from a different level of abstraction. These can assist in the process of redocumenting and rediscovering the design of the systems, important steps in reverse engineering [3].

This paper focuses on tools that use binary files as input, rather than reverse engineering from source code. While some of the tools surveyed can be used on source code, the focus is on their capabilities with binary executable files.

Previous surveys on reverse engineering tools are outdated [7], [10], [11], as most of them take place around or before the turn of the century. Additionally, no surveys have looked at tool evaluation from the angle of a novice reverse engineer. By capturing a new angle, we hope to add new, rich data to the study of reverse engineering tools, as well as assist newcomers in making the right choice of tool to use.

### 3. Methodology

This paper examines five tools from a novice’s prospective. All tools are examined on a 32-bit Windows 7 virtual machine. The tools were chosen based on a few heuristics. Many novices may not wish to risk money on an unknown tool; therefore, for the tool to be considered for evaluation, it had to be free. Second, the tool had to be available. The list of tools from previous surveys included many academic tools which were either no longer available for download, or whose websites were dead and no longer functioning. Naturally, a tool that could not be located could not be evaluated. Third, the tool had to run on the system in question. Fourth, the tool had to be easily discoverable. All tools were discovered through a Google search for ‘reverse engineering tools.’ While this might seem like a strange step, it was included as a proxy measure to discover the ‘most popular’ tools. We cannot say with certainty that the tools surveyed are the most popular, but we can say with certainty that they are the most easily discovered, as they appear on the first page of the Google search either as a link to their websites or in lists of reverse engineering tools. In a study of novice reverse engineers, ‘most easily discoverable’ may be a truer representation of their choice of tool than ‘most popular,’ as they may not have colleagues to suggest tools to them and must rely on the output of search engines or other self-discovery methods.

The final tools used in the study are as follows:

- **IDA Pro 5.0** is a commercial dynamic debugger and disassembler. The version used in this study is the free online version. While the current version of IDA Pro contains functions not included in the free version, this study is comparing free software; therefore, the other feature are outside the scope of the paper [12].
  - **OllyDbg** is an x86 dynamic debugger. At the moment, it can only decompile 32-bit code, but a 64-bit version is under active development [13].
  - **Immunity Debugger** is a fork of OllyDbg from the 1.0 version developed by a commercial team but released for free online [14]. It is a dynamic debugger that primarily focuses on exploit creation and defect detection, but can also be used for reverse engineering.
  - **CFF Explorer** is a process explorer that is capable of more than just process exploring. Its primary purpose is to allow users to see how their software is connected with other libraries and functions, but it also includes many periphery features [15].
  - **SysInternals** is a suite of small tools that, when used together, can thoroughly examine a piece of code or a whole system. The tool suite was created by Microsoft developers to help test Microsoft software. It includes a rudimentary process explorer, a system crashing script, a background monitor, an autostart entry point manager, a file permissions checker, and many other utilities. Both 64 and 32-bit versions of the various tools are included in the package. Each tool offers its own insight into the system, though the tools do not work with one another or communicate [16].

Once the tools were chosen, they were investigated using a simple guideline. Previous guidelines were identified and examined to determine common metrics used in evaluating reverse engineering tools [7], [10], [11], [17]. The guidelines include questions on the tool’s content, usability, and interface, and can be seen in condensed form in Table 1. Durability of
the tool was also test by attempting to make the tool crash.

The purpose of this study was to compare the tools, but it was not expected that the study would find a best tool. Previous studies have failed to find a ‘best tool’ based on the heterogeneity of reverse engineering tools, as well as the vast variety of tasks that face reverse engineers, for which any one tool is unlikely to be all-encompassing [1], [7]. As such, the primary research questions for this paper were:

RQ1: How do some of the most commonly used reverse engineering tools compare to one another?

RQ2: Are the tools accessible to a novice user?

4. Results

4.1 Tool Functionality

Of all the tools examined, IDA Pro offered the most functionality. Beyond its basic abilities, it offers multiple graph views of the code, including call flow graphs; allows users to view subgraphs of specific functions or calls; allows users to edit and print graphs; offers a notepad; allows users to comment on the code; has a calculator; allows users to edit the code, edit values used by IDA, and convert objects in the assembly code; and many other functions. Users can search for text, code, and many other types of items. Many of these functions were available both from the toolbars as well as the drop-down menus. In particular, the graphs included both a graph view that could be used actively in debugging, while an option on a toolbar allowed the user to view alternate graphs. Some of the functionality, such as the calculator, seemed unnecessary. Of all the tools, it requires the most memory on startup, and is the slowest to analyze the code. However, it has the most capabilities of all the tools presented.

While not as functional as IDA Pro, Immunity Debugger and OllyDbg both offer dynamic debugging and disassembly as well. Immunity Debugger includes code graphs and Python scripting ability and therefore is more flexible than OllyDbg; however, OllyDbg also has some features that Immunity Debugger does not have, such as the ability to launch some file formats that Immunity Debugger cannot read. While they are not designed for static analysis, both provide a disassembler and can disassemble files, though the files are not as well annotated as IDA Pro’s files.

Aside from the scripter and the minor differences mentioned above, OllyDbg and Immunity Debugger are nearly identical in terms of functionality. Both are relatively slim tools that complete what they are asked to do. However, their search feature is somewhat lacking. It can only find the single next occurrence of a command, which must be an exact match with the input to the search function. Interestingly, while Immunity Debugger advertises being lightweight, it actually uses more memory than OllyDbg. Both applications can analyze the code much faster than IDA, with Immunity Debugger being the faster of the two.

None of the debuggers could attach to themselves. OllyDbg and Immunity hid this by not listing themselves in the list of attachable processes; IDA Pro listed itself, but gave a no-permissions error if the user tried to attach to it, even with elevated privileges. Attaching the debuggers to one another was a surefire way to crash them all. Beyond that, Immunity Debugger and OllyDbg crash if the process they attach to is closed, or if they are closed without detaching from a process. OllyDbg did not offer a detach button and the user had to know the correct order of steps (close the window of the process being analyzed, then close the debugger window), which was not described anywhere, in order to not crash the tool. IDA Pro was more robust, recovering from the closing process and giving the user a list of menu boxes to choose what to do with the process if it was closed without detaching. While the menu boxes were confusing and often led to loops wherein IDA Pro would not close, they at least prevented the tool from crashing.

Immunity Debugger revealed the most information about the processes that were available to attach to when the attach window was opened. IDA only had the ID in decimal and the service name. OllyDbg reported on process ID in hex, the name, the window name, and the path. Immunity Debugger had the process ID in decimal, the name, a service column that reported the services the process offered, a listening column that reported any open ports, the window name, and the path. In this way, the information to support any method of attaching to the process was revealed. However, it might be overwhelming for newcomers, as the information is not necessary to use the tool.

OllyDbg did not provide a command-line interface or a scripting language, while both of the other debuggers did. IDA Pro provides an internal programming language, and Immunity Debugger provides built-in support for Python scripting. While a novice might not be interested in scripting or command-line interfaces, it could become an important feature as they move on to mastery of the tool. However, OllyDbg’s pure GUI interface simplifies the tool for novices by removing the unnecessary and potentially confusing scripting.

CFF Explorer is a powerful tool as well: it includes a process and windows viewer, a PE dump, a PE utility viewer, a PE rebuilder, and a dependency walker, among many other functionalities. Some of them go beyond the scope of a process explorer. For example, while it is not a dynamic debugger, it does have a “quick and dirty” disassembler on one of the tabs, as well as a memory dump, a hex editor, a dependency walker, and various other tools. Almost every value it displayed could be edited by double-clicking on the item, then typing the edits. As the tools are not linked to one another, it sometimes felt like a tool suite with an integrated GUI. CFF Explorer also includes a script-entering mode. Unlike the dynamic debuggers, it was capable of attaching to itself, and could even allow a user to edit its own values without crashing.
The way the functionalities are presented makes it feel as though CFF Explorer is bloated, though it's hard to say that it is. It offers a more complete static analysis than the dynamic debuggers do, and each functionality is clearly delineated from the others through the tabs setup. Additionally, it is much more lightweight than any of the debuggers. Of all the tools with an overarching GUI, it uses the least memory at startup. This is likely because it only displays what it has been asked to display. For example, if the item it is analyzing is a small C application without many dependencies or external objects, it only displays the disassembler tab and the overview tab, but does not include any tabs for dependencies the file doesn’t have. On the other hand, if it is analyzing a large application with many dependencies, it will launch all its tabs. It does not disassemble an artifact until the user presses a button to ask it to disassemble; most other tabs also work on this principle. Thus, it does not waste runtime in loading all the possible functionalities available until the user asks for those specific functionalities.

Each tool in SysInternals has its own role. The whole suite is made up of small, precise tools, which do not do any more than necessary. SysInternals separates each tool into its own item so that the user has no bloat whatsoever and no unnecessary functionality included when all they wanted was one particular tool. On the flipside, if the user wants a wide-angle or whole-picture experience, SysInternals is not going to present that without additional effort on the user’s part. All the tools could be used individually to get an idea of the system or program, but there is no overview available among the tool suite. Overall, SysInternals was similar to CFF Explorer in terms of functionality, but there was no integrated GUI.

### 4.2 Tool Accessibility to Novices

Overall, IDA Pro was the most intuitive tool in the study, which might be expected, as it was a commercially-developed tool. All the options were clearly labeled with descriptive names. The interface was set up so that even though there were many windows, clicking one item highlighted its location across several windows and made it easier to understand. When it comes time to close the analysis, IDA Pro auto-detaches when a user closes the window, which makes the whole process of detaching less confusing and prevents the generation of an unnecessary error. The graph view it provided also gave a wide overview of the whole system and made it easier to grasp a general understanding of how the whole program was wired more quickly. Its static disassembly feature revealed the most information of any disassembler in the study, including some auto-generated comments, and presented the information in the clearest format. The instructions were color-coded for clarity, with values tagged according to what names were associated with them.

IDA Pro’s search function was perhaps the most questionable part of the software from a usability standpoint; IDA Pro provides several different search functions instead of a single search, as is more natural. While it is understandable why these different searches are provided, as searching certain areas can take a very long time, it was confusing at first as to which search did what and how the searches were supposed to be used, potentially causing novices to waste time on
Immunity Debugger offers a few different ways to attach but do display alt-text in a border at the bottom of the screen. From that, they are nearly identical. Both tools have very simple toolbars with letters to identify each function. The letters were sometimes intuitive (i.e. B for Breakpoints), and sometimes not (i.e. K for Call stack). However, between the two, the letters all refer to the same items. The consistency made it easy to switch between the two; however, as they each offer more-or-less the same functionality, it is hard to imagine a situation where a user would have to swap between them. Neither show hovertext to help explain what the letters mean, but do display alt-text in a border at the bottom of the screen. As this is not a normal place to display alt-text, it may frustrate a novice confused by the letter-based toolbar options.

Immunity Debugger offers a few different ways to attach to a program, but unless the user knows beforehand which menu option equates to which method, it is confusing. It is not immediately obvious what differences there are between choosing the attach option and using the other methods. This may be because Immunity Debugger is intended for higher-level users who would be interested in different methods of attaching to use either its Python scripting capabilities or command-line navigation, giving the user finer-grained control than a GUI does. OllyDbg only has one method of attaching, which is easier for a newcomer to navigate. However, it does not offer Python scripting as Immunity Debugger does.

Both OllyDbg and Immunity Debugger had relatively limited annotations in the assembly code they generated. OllyDbg did not color code its assembly code, while Immunity Debugger did, a helpful move that allows novices to quickly grasp a system. In Immunity Debugger, a graph view was available, but not integrated. IDA Pro included annotations, color-coded the assembly code, and provided a graph view integrated into the initial view to provide the user an easy way to immediately recognize the code flow patterns.

Immunity Debugger had a detach button; IDA Pro offered users the option to detach or terminate the process. OllyDbg offers no detach options in its GUI. This was a serious issue that affected user satisfaction; it was not immediately clear how to properly detach from a process without either crashing the process, crashing OllyDbg, or both. Unlike IDA, closing the tool or database without detaching from the process generated an error message box which did not assist the process of detaching in any way, and could lead to the program crashing if handled wrong, and closing the OllyDbg window immediately crashed both OllyDbg and the process it was attached to. With OllyDbg, the only ‘proper’ way to detach without crashing anything is to close the process OllyDbg has attached to, though this sometimes generates errors as well. Immunity Debugger also gave some of these same errors, but the detach feature was available and the error easier to resolve. Overall, both OllyDbg and Immunity Debugger were easy to crash, while IDA was more difficult to crash.

Immunity Debugger has an available user guide and an active forum manned by developers as well as users. OllyDbg has a quick-start guide with a short table of often-used functions, as well as a list of available plugins. Unfortunately, both OllyDbg and Immunity Debugger’s in-application Help functions are outdated and erroneously reports that there is no help, which may leave a novice user in a dead end. On the other hand, IDA Pro has a user guide and a working in-application help function, as well as available paid service packages.

Overall, the dynamic debuggers had the same type of interface: a large window with many smaller windows within it, one for each type of interesting output, that could be positioned around the larger window or layered over one another with toolbars and menus above. The interface felt cluttered, and individual windows sometimes got lost in the clutter. However, given the amount of information needed to debug from assembly, it was actually helpful to have all the information on the screen despite the clutter. It allows for several different items to be paneled on the screen beside one another at the same time, so the user can walk through how one step makes multiple alterations throughout the stack, names, logs, and other areas. IDA Pro’s large background window goes transparent when it attaches to a process in order to allow the user to see and manipulate the process it is attached to, which was a useful function that the other two debuggers did not offer. With Immunity and OllyDbg, the large window stayed prominent in the screen, and the user had to juggle between the application’s window and the debugger’s window.

CFF Explorer was very intuitive to navigate. There was only one method to open a file, and once a file was opened, CFF explorer read out all the relevant data. The initial screen, before any data was loaded, was very simple, with a button to load a file and an empty pane where the file would be loaded; unlike the debuggers, a user did not have to go hunting to find out how to load a file. Menus were simple, and only a few were available. The main window held a tabbed interface with multi-leveled file-viewers and other data on the tabs; sometimes, navigating through the folders displayed in the file-viewers could become annoying, if they didn’t hold anything interesting. Additionally, flipping away from a tab reset that tab; if a user was six levels into a file map and moved to a new tab to look at a different kind of data, they would come back to find themselves back on the first level again.

However, CFF Explorer also offers a multi-windowed interface where each window is persistent, so if a user anticipates swapping between tabs, they can open this view and see the
items side-by-side without having to worry about resetting each page. Navigating to this view was simple; it took the press of a single prominently-displayed button. There was not a user guide for the tool, but there were user guides for the scripting language the tool uses, the extension repository, and for writing an extension. Overall, CFF Explorer was very intuitive as a tool, though the information it displayed was sometimes opaque without specialized knowledge to understand it. Additionally, the author was not capable of crashing CFF Explorer, suggesting that CFF Explorer is very robust.

SysInternals was inconsistent in terms of usability. Some tools had easy-to-use GUIs; other tools ran invisibly. The tools were not intuitively named, nor were they labeled in any way based on the interface they used, be it command-line, GUI, or other. In terms of overall intuitive comprehension, SysInternals ranked the lowest. The tools in the suite that did offer GUIs were fairly intuitive to use, however, they all had different styles of interfaces with different designs and standards; knowing one tool’s interface would not help you navigate another tool’s interface. Some tools executed silently and were meant to be caught by a program, or popped up a command-line output. There was no how-to or README file to help explain the tools, and no user guide available. A book is available, but it is outdated and additionally hidden behind a paywall, which made it beyond the scope of this study. The toolset does have some associated white papers and videos, but none of those are a user guide. The user would have to read all the papers and watch all the videos just in hopes of gleaning relevant data in terms of using the tool. The author could not crash these tools, either, but these tools did succeed at crashing the author’s virtual machine.

5. Discussion

Of the dynamic debuggers, IDA stands out as the one with the most functionality and the best usability. It is the most intuitive for a novice reverse engineer. While it does take longer to load the code in and is relatively heavyweight when compared to either of the other two, it makes up for this in offering more readable and better annotated assembly code and a clearer layout. However, the other two tools are slimmer tools and faster at loading and analyzing code. If a user wants a full analysis of a tool, including both static and dynamic analysis, IDA Pro is the better choice, but if they are worried about memory usage and load times, then Immunity or OllyDbg may be the better tool. Because of its more limited functionality (one way to attach, no scripting) OllyDbg is the better of OllyDbg or Immunity for newcomers; however, a novice who wants to eventually expand into more detailed analysis may prefer to start with Immunity Debugger instead.

Additionally, while all three debuggers allow users to write new extensions, Immunity Debugger also allows users to execute Python scripts, which makes it more flexible than the other debuggers. However, the Python scripts may not be immediately transparent to a novice, so in terms of novice developers, OllyDbg may be the better choice, as it is simpler and relies only on its GUI for functionality. IDA Pro has its own internal programming language, giving it some of the same flexibility as Immunity Debugger. As a user will have to learn that programming language, Immunity Debugger would be a better choice for novices who want to write scripts for the tools. If the user already knows Python, they can jump immediately into scripting without having to learn a new language. Immunity Debugger also includes some prewritten methods for Python scripting, making the process more accessible overall.

CFF Explorer is the most lightweight of all the comprehensive tools; if the user does not need a debugger, it provides functionalities the other tools do not provide and gives a clear view of the target process’s dependencies, among other insights. Additionally, it is simple and intuitive to use, easily navigated by a novice, and robust.

SysInternals, while lightweight, is best used by an experienced user who knows what they’re looking for in a system. Its uses cannot be easily discerned through a point-and-click method, and as the user manual is behind a paywall, it is also difficult to look up how to use it properly. The majority of the SysInternals suite is almost impossible to comprehend without foreknowledge or extensive research.

Novice reverse engineers could be assisted by providing tools with intuitive user interfaces, or reworking existing tools to have more intuitive interfaces. A simple change would be to improve OllyDbg and Immunity Debugger by changing the letters in the toolbar to descriptive titles or images. In OllyDbg’s case, it could further improve by providing color-coding as Immunity Debugger does. Immunity Debugger could present one method of attaching in its menus, and abstract the other methods to a second menu level or to the scripting language. CFF Explorer is largely intuitive; however, it would be preferable if it did not reset the tabs when the user clicked to a new tab, or warned users about this quirk and directed them to the multi-window interface. IDA Pro presents many toolbars and windows at startup, which can give the user an impression of clutter. If it presented fewer toolbars on startup, perhaps only the ones that included the most vital information, but included the option to enable more toolbars later on, it would give less of an impression of clutter, and would still allow users to have quick access to all the functionality they wanted to have access to. The SysInternals tools that did have GUIs were reasonably comprehensible. If all the tools for which it is reasonable to give a GUI had GUIs, and all the GUIs were based on the same format, then SysInternals would be more accessible to a novice.

The tools do not have to be simple to be accessible to a novice. Although IDA offered the most complexity of any dynamic debugger and disassembly tool, it was also the most usable of those tools thanks to its intuitive user interface. By presenting a user interface that was human-readable and well-annotated,
IDA stood out as an easy tool to use, regardless of its advanced functionality. Clearly marking advanced functionality or making the simple functionality more reachable than the advanced, such as placing the simple functionality on a toolbar and the advanced functionality in menus, could assist a novice user in immediately determining which functions were of use to them. By providing better user interfaces overall, the tools will become more usable for both novices and experts. As several of the tools also offer command-line scripting, adding a more intuitive GUI should not prevent the expert user from using the tool the way they are familiar with; experts can continue to use the command line and other scripting interfaces, while the GUI can be used by the novices.

6. Threats to Validity

Internal Validity. Only one novice evaluated the tools. Therefore, the results may differ with a different novice. The guidelines were developed to give an overview of the system from a novice perspective; however, there are doubtlessly functionalities that were left out. To the best of the author’s ability, the guidelines capture the core functionalities of the tools.

The selection of tools may not truly reflect the most popular or the most easily discovered tools. However, it seems reasonable that a novice would search for tools on a search engine, so it is likely the tools are at least somewhat discoverable for novices.

External Validity. As the tools were chosen and evaluated by a single investigator who also developed the guidelines, the results are not generalizable. However, this paper presents the baseline for an investigation into modern tools, which other authors can use going forward to conduct their own investigations. Additionally, most of the chosen measures, such as what platform the tool runs on and whether or not it has a search feature, are essentially quantitative and would not change regardless of investigator. Only the qualitative elements of the guidelines would change with the investigator.

7. Conclusions

Overall, IDA Pro’s free version is the best freeware available for novice reverse engineers. CFF Explorer, while not a dynamic debugger, offers an intuitive UI and a simple interface suitable for newcomers as well. Immunity Debugger and OllyDbg are slim tools that might be confusing to a novice, but nonetheless offer functionality and user interfaces for the user to discover their features through. SysInternals is best used by an expert, and may be overwhelming for a newcomer. Each tool has its role in reverse engineering; novices, however, are best off sticking to tools with a clear, intuitive GUI. Future tools that want to be accessible to novices should focus on simple and intuitive GUIs with descriptive labels, while not removing functionality from the tools.

References