Fall 2025

Master of Science in Cyber Security Project Report

Computer Science Department

California State University, **Dominguez Hills**

Comprehensive Feature Comparison of Software Reverse Engineering Frameworks: IDA Pro, GHIDRA, and Binary Ninja

---

A Project

Presented

To the faculty of

California State University, Dominguez Hills

In Partial Fulfillment

Of the Requirements for the Degree

Master of Science

in

Cyber Security

---

by

Jeffrey Han

Fall 2025

I am dedicating this paper to my mother, who originally pushed me to pursue a higher education, and my father, who was generous enough to help fund my education.

ACKNOWLEDGEMENTS

I would like to share my acknowledgements to the faculty of the Cyber Security program. Specifically I would like to thank Dr. Alireza Izaddost, for his generous patience and understanding, and Dr. Bhrigu Celly, for demonstrating a sliver of the professional world in his class.

AUTHOR'S DECLARATION

I hereby declare that this project consists of original work which I have authored. This is a true copy of the work, including any required final revisions as accepted by my committee or course instructor.

Name	Jeffrey Han
Signature	
Date	12/16/2025

TABLE OF CONTENTS

LIST OF FIGURES

PAGE

ABSTRACT

Cyber threats and malware continued to increase in complexity which further emphasizes the lack of well trained individuals with proper knowledge or tools for software reverse engineering (SRE). Despite the crucial role SRE plays in vulnerability discovery, malware analysis, and program repair, a large knowledge gap exists regarding the tools and concepts of decompilation. This paper aims to narrow this gap by providing a foundational understanding of decompilation and conducting a comparative review of the three most popular SRE frameworks: IDA Pro, Ghidra, and Binary Ninja. The frameworks are assessed on three core criteria: History/Maturity, Unique Features, and Cost. In review, it reveals that IDA Pro with its maturity and powerful add-ons makes the most comprehensive package despite its high financial barrier of entry. Ghidra in its fully featured, open source version provides a complete package with a wide range of community support and resources. Binary Ninja provides modern features with AI integration and embedded reverse engineering at a fractional cost in comparison to IDA Pro. A feature comparison chart was created to provide a clear reference for selecting the optimal SRE framework based on specific operational needs and budget constraints.

CHAPTER 1

INTRODUCTION

With the steady increase of elaborate cyber attacks and obfuscated malware, the importance of decompilation and software reverse engineering (SRE) only grows . As new threats emerge, cyber forensic efforts must keep pace to understand the behavior and purpose of malware and subsequently prevent future attacks from occurring. It can be applied to a wide range of applications in cyber security such as vulnerability discovery, malware classification, program repair, and so much more [1]. Decompilation is only as good as the tools used and many cyber forensic specialists use a range of tools from debuggers, disassemblers, decompilers, memory mappers, hex editors, and etc. These tools, crucial to the reverse engineering process, are obtainable but not particularly taught or readily accessible. According to a 2022 survey by the University of New Haven, they discovered that a majority of survey participants believe that there was a shortage in reverse engineering resources and that recent graduates are not leaving with proper reverse engineering knowledge from their education [2]. This paper aims to fill this gap by providing comprehensive knowledge on the foundations of decompilation along with background information on three software reverse engineering (SRE) frameworks: Ghidra, IDA Pro, and Binary Ninja. In doing so, further discussion comparing the features of each framework is possible along with a set of criteria which summarizes the strengths and weaknesses of each.

CHAPTER 2

BACKGROUND

WHAT IS DECOMPILATION?

Decompilation, being the opposite of compilation, is the process of taking machine assembly code and recovering functionally identical higher-level representation, often to human-readable code. And while compiling human readable code to assembly code is relatively trivial, the reverse is not as simple since the compilation process is not lossless. While the process differs depending on the structure of the original code, decompilation will generally align with the following pipeline:

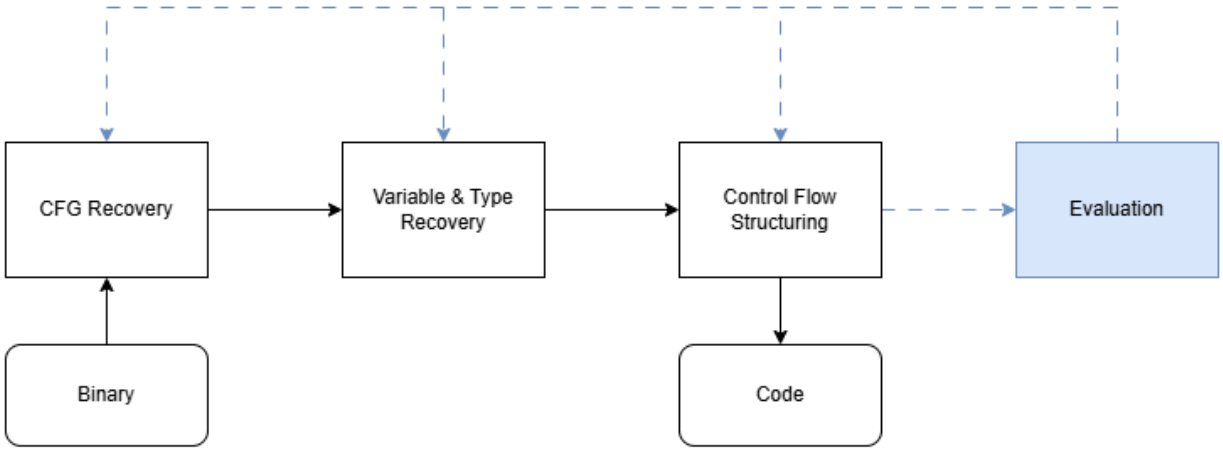

Fig 1: Generic Decompilation Pipeline [3]

The pipeline consists of three main areas: Control Flow Graph Recovery (CFG), Type Recovery, and Control Flow Structuring [4]. The control flow graph is a graph of directed nodes which represents the flow of code execution in assembly code. The first step in decompilation is recovering the CFG to understand how different parts of a program interact with one another. The second step involves type recovery where the types of variables and data within the program are properly identified. In compiling code, the data type which exists in higher level

representations is omitted as it is not needed for a computer to execute the code. However, in order for a cyber forensics specialist to analyze a program's behavior and function, these data types need to be recovered to make better sense of the original higher level code base. The very last step in the process is control flow structuring which takes the CFG and converts it back into a linear code-like output which is more human-readable.

IMPORTANT SRE CONCEPTS AND CONSIDERATIONS

1. Control Flow Reconstruction

Observe how the control flow graph is reconstructed and evaluate how closely it resembles the ground-truth.

2. Type Recovery

How easily does a decompiler automatically recognize and assign the proper types to variables, functions, and operands.

3. Data Structures & Symbol Reconstruction

How well does a decompiler maintain data structure and program symbols.

4. Readable High Level Output

How clear and understandable the decompiled code is.

5. Intermediate Interpretation

Understanding the intermediate representation of a decompiler, used to convert assembly code into logic control flow, is paramount in evaluating the accuracy and strengths of a decompiler.

6. Deobfuscation/Reversing Optimizations

Provide or allow functionality to help overcome obfuscation and optimization techniques which make it more difficult to trace or step through a program's operations.

7. Architecture and Format Support

How many system architectures and file formats does a decompiler support?

8. Interactivity and Extensibility

Allow a user or community to provide additional functionality to a SRE framework which can range from helpful to crucial.

9. Debugging Integration

Provides comprehensive logs and information which allows for a detailed analysis of a program's behavior.

10. Cost

The financial barrier of entry and what functionalities and conveniences would a user gain or lose in accordance to the amount of money paid.

CHAPTER 3

PROPOSED METHOD

To provide a comprehensive assessment of the current available SRE frameworks, we propose a detailed assessment of each framework's unique features. This comparative analysis will be grounded in three distinct criteria designed to highlight both the history and the unique offerings of each tool. First, we will provide background for each framework to understand its development trajectory, maturity, and community foothold. Second, we will analyze the features of a framework that differentiate each platform, focusing on specific capabilities such as decompilation accuracy, intermediate languages, and plugin architecture. Finally, we will review the Pricing models to determine the cost-effectiveness for various user demographics, ranging from individual hobbyists to large enterprise security teams.

Following the detailed examination of these qualitative and quantitative factors, the findings will be synthesized to produce a holistic view of the market options. By consolidating the data gathered, we will construct a high-level feature comparison chart. This visual aid will serve as a solid reference point, allowing practitioners to quickly identify which framework best aligns with their operational needs and budget constraints, ultimately facilitating a more informed and data-driven decision-making process. The feature comparison chart will use the 10 SRE concepts and considerations presented during the Background section to outline the differences between the IDA Pro, GHIDRA, and Binary Ninja.

CHAPTER 4

IMPLEMENTATION

INTERACTIVE DISASSEMBLER (IDA)

Often attributed to being the industry standard for SRE, IDA has been commercially sold since 1996 by DataRescue with development eventually taken over by Hex-Rays in 2008. Hex-Rays advertises the following use cases for IDA: Malware Analysis and Digital Forensics, Penetration Testing, Dynamic Analysis and Debugging, Automotive Security, Interoperability, Software Assessment, and Education. It is available on MacOS, Linux, and Windows and had unique microcode as an intermediate representation of the decompiled code [5].

FEATURES

Being the oldest commercially available reverse engineering framework, IDA boasts a suite of unique features to their platform:

LUMINA

A metadata sharing service which hastens SRE operations by automatically recognizing common code patterns from various users and binaries and replacing them with known function names, prototypes, and operand types. The strength of this tool is allowing users to focus on decompiling unique and more difficult assembly code without working through the tedium of more common code structures. This feature is available with IDA Pro, Home, and Classroom licenses for free. However if the user or organization wishes to keep their metadata private, they can opt from a Private Lumina Add-on service which allows greater control over the user's metadata [6].

Fast Library Identification and Recognition Technology (FLIRT)

A tool closely aligned with Lumina, FLIRT is a signature matching tool which allows IDA to distinguish and label library code compared to unique assembly code. It can recognize code structures from common compilers like MSVC or GCC and label common functions like printf or strcpy. It saves valuable time like Lumina by reducing the time needed to decompile common code structures and allow the user to focus on unique program specific assembly code [7].

gooMBA

A specific plugin designed to simplify complex Mixed Boolean-arithmetic expressions often seen in obfuscated assembly code. This is incredibly useful in malware and threat analysis where malicious actors will purposefully obfuscate their code to thwart or delay forensic efforts [8].

TEAMS

An add-on allowing collaborative features using a git-like version control approach.

PRICING

IDA PRO runs on a subscription basis for three different use cases: commercial, individual/non-commercial, and education. Depending on the plan, features may be limited to a smaller set of architectures or have certain services omitted. Payment plans are annual only with the education plans just being commercial plans but discounted, excluding the Classroom plan [9].

Commercial Plans	Starting Price	Differing Features	Supported Decompilers
IDA PRO ESSENTIAL	$1099/yr	2 cloud-based decompilers of user choice from the same family	x86-32&64 ARM-32&64 MIPS 32+nM&64 PPC 32&64 RISC-V 32&64
IDA PRO EXPERT 2/4/6	$2999/yr $4999/yr $6899/yr	2/4/6 local decompilers of user choice + Optional Floating Licenses	Essential Decompilers + ARC
IDA PRO ULTIMATE	$8599/yr	All local decompilers included, Private Lumina, Teams + Optional Floating Licenses	All decompilers
IDA PRO OEM	Quote	OEM Licenses for integrated use of IDA	All decompilers

Figure 2: IDA Commercial Plans Breakdown

Education Plans	Starting Pricing	DifferingFeatures	Supported Architectures
IDA Classroom	Free	2 families supported for disassembly 4 cloud decompilers	x86-32&64 ARM-32&64
ESSENTIAL	$~~1099~~ $330/yr	“”	“”
EXPERT 2/4/6	$~~2999~~ $900/yr	“”	“”
ULTIMATE	$~~8599~~ $2500	“”	“”

Figure 3: IDA Education Plans Breakdown. “” represents the same features as Commercial Plan Equivalent.

Individual Plans	Starting Price	Features	Supported Architectures
IDA Free	Free	PC disassemblers only 2 cloud decompilers	x86-32&64
IDA HOME	$365/yr	1 disassembler of user choice 2 cloud decompilers from same family IDA Python, IDC, C++ Scripting Local and GDBServer Debugger	x86-32&64 ARM-32&64 MIPS 32+nM&64 PPC 32&64 RISC-V 32&64 ARC

Figure 4: IDA Individual/Non-commercial Plans Breakdown

GHIDRA

Released in March 2019 but maintained internally since the early 2000s, Ghidra is a fully featured free open source reverse engineering framework created by the National Security Agency (NSA) [10,11].

FEATURES

WIDE ARCHITECTURE SUPPORT

The list of supported processors for Ghidra is very extensive: x86 16/32/64, ARM/AARCH64, PowerPC 32/64/VLE, MIPS 16/32/64/micro, 68xxx, Java / DEX bytecode, PA-RISC, PIC 12/16/17/18/24, Sparc 32/64, CR16C, Z80, 6502, 8051, MSP430, AVR8, AVR32 [12].

GHIDRA SERVER

Allows for multiple users to contribute to a single reverse engineering project for easier collaboration and contributions [13].

EXTENSIONS

They are optional user-contributed components which extends Ghidra's functionality beyond its original function. The open source nature of the Ghidra project allows for a large number of user extensions [13].

BEHAVIORAL SIMILARITY (BSim)

A Ghidra plugin which helps find structurally similar functions in large collections of binaries. It creates a database of these pieces of code which behaviorally function the same even if the underlying code differs slightly [14]

BINARY NINJA

Developed by Vector 35 Inc. and released in July 2016,

FEATURES

SIDEKICK

An AI-powered plugin which uses large language models to automate complex tasks in malware analysis, scripting, interactive assistants, and smart suggestions. Requires a separate subscription fee from the base Binary Ninja subscription to function [15]

FIRMWARE NINJA

Additional functionality included in the Binary Ninja Ultimate Edition which helps in reverse engineering embedded systems. Some features include entropy analysis, memory insights, board descriptions, and firmware relationship analysis. Particularly useful in the reverse engineering of Internet of Thing embedded devices and systems [16].

PRICING

Allows users to purchase a perpetual named client license with one year of updates. A continued subscription is available for continued customer support and software updates [17].

Non-Commercial	Price	Unique Features
Free	$0	x86-32/64, ARMv7, Thumb2
Personal	$299 ($74 w/ student discount)	12+ architectures, Sidekick capable, BNIL Introspection, Plugins

Figure 5: Binary Ninja Non-commercial Plans Breakdown

Commercial Plans	Price	Unique Features
Commercial	$1499 ($374 w/ student discount)	Personal Plan Features + Headless, Local Project Management
Ultimate	$2999 (Introductory price)	Commercial Plan Features but 17+ architectures Firmware Ninja, Enterprise Features with separate Enterprise Server Purchase.
Enterprise	Quote	Ultimate Plan Features w/ included Enterprise Server Features Floating Licenses Offline Update Distribution

Figure 6: Binary Ninja Commercial Plans Breakdown

FEATURE COMPARISON

Feature	Ghidra	Hex-Rays (IDA)	Binary Ninja
1. Control Flow Reconstruction	Excellent: P-code IR enables strong structural recovery	Outstanding: gold standard for clean control structures	Very good: MLIL/HLIL provides clear structured flow
2. Type Recovery & Variable Renaming	Strong: inferred types and persistent renaming	Excellent: best-in-class type inference	Good and improving: MLIL aids strong type propagation
3. Data Structures & Symbol Reconstruction	Good: reconstructs structs, classes, and vtables	Excellent: rich reconstruction of complex data	Good: supports custom types, structs, and metadata
4. Readable High-Level Output	Very good: C-like, though sometimes verbose	Excellent: highly readable and idiomatic	Good: clear pseudo-C via HLIL; steadily improving readability
5. Architecture & Format Support	Excellent: x86, ARM, MIPS, PPC, RISC-V, etc.	Strong but licensed per architecture	Excellent: supports x86, ARM, MIPS, PPC, AArch64
6. Intermediate Representation (IR)	Yes: robust P-code IR	Yes: proprietary microcode IR	Yes: layered ILs (LLIL, MLIL, HLIL) — very flexible
7. Cross-Referencing & Navigation	Excellent integration with graphs and listings	Excellent IDA graph view and navigation	Excellent: IL cross-links, visual flow graph
8. Anti-Obfuscation / Optimization Handling	Good: tolerates compiler optimizations, some obfuscation	Excellent: best at handling optimized binaries	Moderate: improving with IL-level analysis
9. Interactivity & Extensibility	Excellent: full API, Python/Java scripting	Good: IDAPython and SDK available	Excellent: strong Python API and plugin ecosystem
10. Debugging & Integration	Yes: integrated debugger	Yes: integrated debugger	Yes: integrated debugger (Binjatron plugin, core support)
11. Cost	Free	Commercial, Individual, and Educatioin Plans	Personal -$299, Commercial $1499, Ultimate $2999
12. Official Support	Community-Driven and Open Source	Commercially Supported	Commercially Supported

Figure 7: Feature Comparison Chart for IDA, Ghidra, and Binary Ninja

CHAPTER 5

RESULTS AND DISCUSSION

Each framework dominates specific aspects of the reverse engineering landscape through distinct strengths. IDA Pro remains the industry standard largely due to its unparalleled maturity; its decompiler is bolstered by its strong add-ons like the Lumina metadata sharing service and FLIRT. In contrast, Ghidra disrupts the field by being free and open-source; its greatest strength lies in its accessibility and collaboration features, allowing teams to work simultaneously on the same binary without additional capital on IDA's Teams add-on or Binary Ninja's enterprise services. Binary Ninja distinguishes itself with modern design and automation; it offers a clean user interface and a powerful, developer-friendly API based on its Binary Ninja Intermediate Language (BNIL), making it the preferred choice for those building automated analysis tools or custom plugins.

However, each tool requires compromises regarding usability, performance, or cost. IDA Pro's significant barrier is its high price point and closed ecosystem; the cost prohibits many hobbyists from using it, and its user interface, while functional, feels dated compared to modern software standards. Ghidra, while free, often suffers from performance issues with very large binaries and has a user interface that many find cluttered or non-intuitive compared to the sleekness of its commercial rivals. Finally, Binary Ninja, despite its rapid growth, has a smaller community and plugin ecosystem compared to IDA's decades of accumulated knowledge, meaning users may find fewer pre-built scripts or third-party resources for obscure architectures.

CHAPTER 6

CONCLUSION & FUTURE CONSIDERATIONS

To provide a comprehensive assessment of the current landscape in software reverse engineering, a structured evaluation of three industry-leading frameworks is proposed for: IDA Pro, Ghidra, and Binary Ninja. This comparative analysis will be grounded in three distinct criteria designed to highlight both the legacy and the innovation of each tool. First, the history of each framework is examined to understand its development trajectory, maturity, and community foothold. Second, unique features which differentiate each platform were analyzed, focusing on specific capabilities such as decompilation accuracy, intermediate languages, and plugin architecture. Finally, pricing models were reviewed to determine cost-effectiveness for various user demographics, ranging from individual hobbyists to large enterprise security teams.

In recent years, more and more research has started exploring the use of large-language models and other AI technologies to help address the limitations in traditional reverse engineering methods and tools. The complexity of the tasks and the manual labor required to decompile code traditionally is largely resolved with the use of artificial intelligence and machine learning. AI has helped in the automation of disassembly, decompilation, malware detection, classification, obfuscation detection , and even code semantic inference. There is great potential in this line of research as automation serves to lessen the burden of the user. It is seen in existing add-ons which utilize AI in existing SRE frameworks like Binary Ninja's Sidekick and Ghidra's user contributed GhidrAssist [18]. However, even with the vast potential of automation, human intervention is still necessary given the low accuracy of decompilers without human review [19]. Well educated and practiced cyber forensics specialists are still needed in the field of malware and threat forensics.

REFERENCES

[1] D. W. Team, “The decompilation wiki¶,” The Decompilation Wiki - Decompilation Wiki, https://decompilation.wiki/ (accessed Dec. 3, 2025).

[2] Barone, C. R., “A Reverse Engineering Education Needs Analysis Survey”, *arXiv e-prints*, Art. no. arXiv:2212.07531, 2022. doi:10.48550/arXiv.2212.07531.

[3] Brumley, David, et al. "Native x86 decompilation using Semantics-Preserving structural analysis and iterative Control-Flow structuring." 22nd USENIX Security Symposium (USENIX Security 13). 2013.

[4] Basque, Zion Leonahenahe, et al. "Ahoy SAILR! There is no need to DREAM of C: A compiler-aware structuring algorithm for binary decompilation." 33st USENIX Security Symposium (USENIX Security 24). 2024.

[5] “Rays: State-of-the-art binary code analysis tools,” Hex, https://hex-rays.com/ (accessed Dec. 9, 2025).

[6] “Private lumina,” Fast Function Recognition & Metadata Control, https://hex-rays.com/lumina (accessed Dec. 9, 2025).

[7] “Ida F.L.I.R.T. technology: In-depth: Hex-rays docs,” RSS, https://docs.hex-rays.com/user-guide/signatures/flirt/ida-f.l.i.r.t.-technology-in-depth (accessed Dec. 9, 2025).

[8] “Goomba: Hex-rays docs,” RSS, https://docs.hex-rays.com/user-guide/decompiler/goomba (accessed Dec. 9, 2025).

[9] “Ida pricing plans: Pro, Home & Free,” IDA Pricing Plans: Pro, Home & Free, http://hex-rays.com/pricing (accessed Dec. 9, 2025).

[10] D. Votipka, M. N. Punzalan, S. M. Rabin, Y. Tausczik, and M. L. Mazurek, “An investigation of online reverse engineering community discussions in the context of Ghidra,” in *Proceedings - 2021 IEEE European Symposium on Security and Privacy, Euro S and P 2021*, Institute of Electrical and Electronics Engineers Inc., Sep. 20

[11] B. Knighton and C. Delikat, “Ghidra - Journey from Classified NSA Tool to Open Source,” YouTube, https://www.youtube.com/watch?v=kx2xp7IQNSc (accessed Dec. 9, 2025).

[12] NationalSecurityAgency, “Frequently asked questions,” GitHub, https://github.com/NationalSecurityAgency/ghidra/wiki/Frequently-asked-questions (accessed Dec. 9, 2025).

[13] NationalSecurityAgency, “Ghidra/GhidraDocs/GettingStarted.md at master · NationalSecurityAgency/Ghidra,” GitHub, https://github.com/NationalSecurityAgency/ghidra/blob/master/GhidraDocs/GettingStarted.md#running-ghidra (accessed Dec. 9, 2025).

[14] NationalSecurityAgency, “Ghidra/GhidraDocs/GhidraClass/BSim/BSIMTUTORIAL_INTRO.MD at master · NationalSecurityAgency/Ghidra,” GitHub, https://github.com/NationalSecurityAgency/ghidra/blob/master/GhidraDocs/GhidraClass/BSim/BSimTutorial_Intro.md (accessed Dec. 9, 2025).

[15] “Your AI Reverse Engineering assistant,” Binary Ninja Sidekick, https://sidekick.binary.ninja/ (accessed Dec. 9, 2025).

[16] “Embedded reverse engineering with firmware ninja,” Binary Ninja, https://binary.ninja/2025/04/02/firmware-ninja.html (accessed Dec. 9, 2025).

[17] “Purchase,” Binary Ninja, https://binary.ninja/purchase/ (accessed Dec. 9, 2025).

[18]

A. Ghimire, S. R. Lingala, J. Zhang, F. Alsulami, and F. Amsaad, “A Survey on Application of AI on Reverse Engineering for Software Analysis and Security,” *IEEE Access*, vol. 13, pp. 152903–152913, 2025, doi: 10.1109/ACCESS.2025.3593456.

[19]

Y. Cao, R. Zhang, R. Liang, and K. Chen, “Evaluating the Effectiveness of Decompilers,” in *ISSTA 2024 - Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, Association for Computing Machinery, Inc, Sep. 2024, pp. 491–502. doi: 10.1145/3650212.3652144.