ISSN: 2582-7219

International Journal of Multidisciplinary Research in Science, Engineering and Technology

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

Impact Factor: 8.206

Volume 9, Issue 1, January 2026

© 2026 IJMRSET | Volume 9, Issue 1, January 2026| DOI:10.15680/IJMRSET.2026.0901041

ISSN: 2582-7219 | www.ijmrset.com | Impact Factor: 8.206| ESTD Year: 2018|

International Journal of Multidisciplinary Research in Science, Engineering and Technology (IJMRSET)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

A Comparative Analysis of Ghidra and IDA Free: Static Artifact Detection and Logic Reconstruction in Reverse Engineering

Stephanus A. Ananda, Agustinus Noertjahyana, Kartika Gunadi, Farrell C.M

Assistant Professor, Department of Informatics, Petra Christian University, Surabaya, Indonesia

Assistant Professor, Department of Informatics, Petra Christian University, Surabaya, Indonesia

Assistant Professor, Department of Informatics, Petra Christian University, Surabaya, Indonesia

Student, Department of Informatics, Petra Christian University, Surabaya, Indonesia

ABSTRACT: Static analysis is a fundamental approach in reverse engineering that allows researchers to inspect software artifacts without the risks associated with dynamic execution. While commercial tools like IDA Pro set the industry standard, the emergence of high-capability open-source alternatives has reshaped the landscape. This study evaluates the comparative effectiveness of two leading free reverse engineering frameworks: IDA Free and Ghidra. The assessment is conducted within an isolated FLARE-VM environment using a diverse dataset of modern binaries, including Python-packaged executables (PyInstaller), standard C applications, UPX-packed malware, and encrypted ELF payloads.

The findings reveal that the two tools possess distinct, complementary strengths. Ghidra demonstrates superior performance in initial artifact detection, explicitly identifying "MZ/PE" signatures and packer indicators (UPX) that IDA Free often obscures. Conversely, IDA Free excels in deep logic reconstruction, producing cleaner and more readable pseudocode for complex control flow structures, such as switch-case statements, where Ghidra frequently fails.

The study concludes that an optimal open-source workflow should leverage Ghidra for initial triage and identifying file characteristics, while utilizing IDA Free for precise algorithmic analysis, providing a cost-effective methodology for independent security researchers.

KEYWORDS: Reverse Engineering, Static Analysis, Ghidra, IDA Free, Decompiler, Malware Analysis, Software Forensics.

I. INTRODUCTION

The rapid evolution of malware and software complexity presents a persistent challenge to cybersecurity defense mechanisms. To effectively secure systems, researchers must possess the capability to dissect and understand malicious software artifacts without relying on source code availability. Reverse engineering (RE) serves as the cornerstone of this process, enabling analysts to reconstruct logic, identify vulnerabilities, and anticipate potential threats.

Within the RE domain, static analysis is often the first line of defense, allowing researchers to inspect binaries and code structures without the risks associated with executing potentially malicious code. While dynamic analysis provides behavioral insights, it requires controlled environments and can be evaded by modern malware detecting virtual machines. Therefore, robust static analysis tools are indispensable for deep code inspection.

Currently, the landscape of static analysis tools is bifurcated between expensive commercial standards and open-source alternatives. Interactive Disassembler (IDA) has long been regarded as the industry standard. However, its full capabilities are often locked behind prohibitive licensing costs, with the "Free" version historically lacking critical features such as a decompiler. In contrast, the National Security Agency (NSA) released Ghidra as an open-source

ISSN: 2582-7219 | www.ijmrset.com | Impact Factor: 8.206| ESTD Year: 2018|

International Journal of Multidisciplinary Research in Science, Engineering and Technology (IJMRSET)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

framework in 2019, disrupting the market by providing enterprise-grade features, including a built-in decompiler, at no cost.

Research Gap and Motivation: Despite the popularity of both tools, there is a lack of recent academic literature that specifically compares the free versions of these tool chains against modern obfuscation techniques. Most existing comparative studies focus on commercial versions (IDA Pro) or older software samples. This creates a knowledge gap for independent researchers, students, and open-source analysts who need to choose the most effective tool without financial barriers.

Contribution: This paper addresses this gap by conducting a comparative static analysis of Ghidra and IDA Free. Unlike previous studies, this research evaluates their effectiveness across a diverse set of modern challenges, including:

- High-level language extraction (Python/PyInstaller).
- Standard logic reconstruction (C/C++).
- Packed binary analysis (UPX).
- Encrypted payload handling (GPG/ELF).

By rigorously testing these tools in an isolated FLARE-VM environment, this study aims to provide a definitive guide on the strengths and limitations of each tool, ultimately assisting the open-source community in optimizing their reverse engineering workflows.

II. LITERATURE REVIEW

Reverse engineering (RE) plays a pivotal role in cybersecurity, serving as the primary method for analysing malware behaviour, understanding legacy systems, and identifying vulnerabilities in closed-source software [1]. The discipline is fundamentally divided into two complementary approaches: static analysis and dynamic analysis. While dynamic analysis involves observing the code during execution in a controlled environment, static analysis examines the binary's structure and code without execution, making it a safer initial step for analysing potentially dangerous artifacts [2].

Recent studies emphasize the importance of static analysis in detecting malicious patterns. Suleman [3] highlights that static analysis allows researchers to identify hardcoded secrets, cryptographic constants, and logical flows that might be obfuscated during runtime. However, the effectiveness of static analysis is heavily dependent on the capabilities of the tools employed. Sugiarto et al. [4] argue that combining behavioural analysis with static signatures provides the most complete threat picture, yet for many researchers, the high cost of commercial tools remains a barrier to entry.

The landscape of RE tools has shifted significantly with the release of Ghidra by the National Security Agency (NSA). Prior to 2019, the Interactive Disassembler (IDA) was the undisputed industry standard [5]. Bhat et al. [6] conducted an early comparative study of RE tools, noting that while IDA Pro offered superior stability and a vast plugin ecosystem, its prohibitive cost limited accessibility for academic and independent research. The introduction of Ghidra provided a free, open-source alternative with enterprise-grade features, including a built-in decompiler that supports multiple architectures [7].

More recent comparisons have focused on the specific capabilities of these tools against modern challenges. Arias [8] evaluated Ghidra and IDA Pro in the context of malware analysis, finding that while IDA Pro remains more efficient for experienced professionals due to its refined graph views, Ghidra's collaboration features and scriptability make it superior for training and community-driven projects. Similarly, Alam [9] in his 2025 study on secure malware analysis tools, posits that Ghidra has elevated the standard for free tools, effectively challenging the monopoly of commercial disassemblers.

However, comparative literature specifically targeting the "Free" version of IDA versus Ghidra remains sparse. Most studies, such as Devine et al. [10], compare Ghidra against the full commercial version of IDA Pro or other paid suites like JEB Pro. In their evaluation of vulnerability detection rates, Devine et al. found that both toolchains performed comparably in identifying buffer overflows, though Ghidra often required less manual configuration for non-x86 architectures.

ISSN: 2582-7219 | www.ijmrset.com | Impact Factor: 8.206| ESTD Year: 2018|

International Journal of Multidisciplinary Research in Science, Engineering and Technology (IJMRSET)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

The challenge of code obfuscation further complicates this comparison. Kim et al. [11] discuss a framework for quantifying obfuscation quality, noting that static analysis tools often struggle with techniques like control flow flattening and packed binaries. Shankarapani et al. [12] demonstrated that opcode frequency analysis can aid in detecting metamorphic malware, but widely used tools like IDA Free often lack the advanced de-obfuscation scripts available in their paid counterparts. This limitation is critical when analysing packed executables, such as those compressed with UPX, where the ability to identify and unpack the payload is a prerequisite for successful analysis [13].

This paper addresses the gap in current literature by specifically comparing the *freely available* versions of these tools IDA Free and Ghidra across a diverse set of binaries (Python, C, Packed, and Encrypted) to determine their viability for open-source research and education.

III. METHODOLOGY OF PROPOSED SURVEY

This study employs a comparative experimental design to evaluate the efficacy of static analysis tools. The methodology is structured into four phases: environment setup, toolchain selection, dataset preparation, and the analytical workflow.

A. Experimental Environment Setup

To ensure reproducibility and safety, particularly when handling potentially malicious structures, all experiments were conducted within an isolated sandbox environment. The study utilized **FLARE VM**, a Windows-based distribution specifically tailored for malware analysis and reverse engineering.

The virtual environment was hosted on VMware Workstation Pro with the following hardware allocation to simulate a standard analyst workstation:

- **Operating System:** Microsoft Windows 10 (64-bit).
- **Processor:** 2 vCPUs.
- **Memory (RAM):** 4 GB.
- **Storage:** 80 GB virtual disk.
- **Network:** Configured in NAT mode to allow controlled update access while maintaining host isolation.

B. Software Toolchain

Two primary reverse engineering platforms were selected for comparison. Both tools were installed within the same virtual environment to maintain consistency.

1. **Ghidra:** The latest stable version was utilized, featuring its built-in decompiler capabilities.
2. **IDA Free:** The free version of the Interactive Disassembler was used, which relies on a cloud-based decompiler for high-level code reconstruction.

To preserve the validity of the comparison, both tools were operated using their default configurations without the addition of third-party plugins or scripts.

C. Experimental Dataset

Four distinct software samples were developed to represent common reverse engineering scenarios, ranging from high-level interpreted languages to packed and encrypted binaries.

1. **Sample A (Interpreted/Frozen Code):** A Python-based GUI calculator application utilizing the Tkinter library. This sample was compiled into a Windows executable using **PyInstaller** to test the tools' ability to handle bundled runtime environments.
2. **Sample B (Standard Logic):** A menu-driven console application written in C and compiled with **MSVC**. This sample includes standard control flow structures, such as if-else blocks, loops, and switch-case statements, to evaluate logic reconstruction accuracy.
3. **Sample C (Packed Binary):** A "Crackme" challenge application written in C/C++, protected by the **UPX (Ultimate Packer for eXecutables)** packer. This sample requires the user to input a password and validates specific character indices, testing the tools' packer detection and unpacking workflows.
4. **Sample D (Encrypted Payload):** An ELF (Executable and Linkable Format) binary compiled with **GCC** on Linux. The binary contains a hardcoded, obfuscated byte array (enc[]) that is decrypted at runtime using a bitwise XOR operation. This sample evaluates cross-platform analysis capabilities and data obfuscation handling.

D. Analysis Workflow

A systematic static analysis protocol was applied to each sample using both tools:

1. **File Format Identification:** Verification of file headers (PE/ELF) and architecture detection.
2. **Metadata & String Extraction:** Analysis of embedded strings to identify compilers, libraries, and packers.

ISSN: 2582-7219 | www.ijmrset.com | Impact Factor: 8.206| ESTD Year: 2018|

International Journal of Multidisciplinary Research in Science, Engineering and Technology (IJMRSET)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

3. **Disassembly:** Inspection of the assembly instructions to map control flow graphs (CFG).
4. **Decompilation & Logic Reconstruction:** Converting assembly code into pseudocode to interpret the underlying program logic and algorithms.

IV. CONCLUSION AND FUTURE WORK

This section details the findings from the static analysis of four distinct software samples. The evaluation focuses on file format identification, metadata extraction, and the quality of logic reconstruction provided by Ghidra and IDA Free.

A. Analysis of Interpreted Code (PyInstaller)

The first sample was a Python-based calculator packaged as a Windows PE file.

- **Format Detection:** Both tools successfully identified the binary as a valid Portable Executable (PE). However, Ghidra was more explicit in displaying the "MZ" and "PE" signatures in its listing compared to IDA Free.
- **Packaging Identification:** A critical step in analyzing Python malware is identifying the packaging method. Both tools successfully detected PyInstaller-specific artifacts. Ghidra's string view explicitly listed markers such as _PYI_ARCHIVE_FILE and PYINSTALLER_STRICT_UNPACK_MODE, confirming that the executable contained bundled Python bytecode.
- **Logic Recovery:** A significant limitation was observed for both tools. Neither IDA Free nor Ghidra could directly decompile the embedded Python bytecode. They correctly disassembled the PyInstaller bootloader and identified the Python 3.8+ runtime library (DLLs) and Tkinter API calls , but the actual calculator logic remained inaccessible without external extractors like *pyinstxtractor*.

B. Logic Reconstruction in Standard C Binaries

The second sample, a menu-driven C application, highlighted significant differences in decompiler performance.

- **Control Flow Analysis:** This sample utilized a switch-case structure for menu selection. IDA Free demonstrated superior performance in reconstructing this logic. Its decompiler produced clean, C-style pseudocode that correctly mapped the switch cases to their respective functions (e.g., if (v1 == 1) return sub_401420();).
- **Decompiler Errors:** Conversely, Ghidra struggled to parse the switch statement. The decompiler output contained warnings such as WARNING: Removing unreachable block, resulting in a broken control flow graph where valid code sections were omitted from the pseudocode view. Furthermore, Ghidra relied on generic function labels (e.g., FUN_004013a0), whereas IDA Free provided clearer variable typing and structure.

C. Detection of Packed Executables (UPX)

The third sample was a "Crackme" protected by the UPX packer.

- **Initial Triage:** Ghidra proved more effective for initial detection. Its "Program Trees" and "Defined Strings" views explicitly listed section names like UPX0, UPX1, and UPX2, immediately alerting the analyst to the presence of the packer.
- **Heuristics vs. Explicit Labels:** IDA Free was less direct; it did not list literal UPX strings in the segment view. Instead, it relied on heuristics, raising a warning that the "imports segment seems to be destroyed," which implicitly suggests packing but requires more interpreter experience to diagnose.
- **Post-Unpacking Analysis:** After the binary was manually unpacked, both tools performed comparably. They successfully recovered the password validation logic, identifying the requirement for a 10-character string where the 5th character (index 4) must be the '@' symbol.

D. Analysis of Encrypted ELF Binaries

The final sample was a GPG-encrypted ELF binary containing an obfuscated string.

- **Encryption Handling:** As expected, neither tool could analyze the file while it was encapsulated in the GPG container. Analysis proceeded only after decryption using the correct passphrase.
- **Data Representation:** Upon analyzing the decrypted ELF, both tools identified the decrypt() routine involving an XOR operation. However, a subtle difference in data representation was noted. IDA Free displayed the obfuscated byte array (enc[]) and the key (0x9B) as unsigned values, matching the source code. Ghidra, in contrast, represented the XOR operand as a sign-extended value (0xffffff9b). While mathematically equivalent, Ghidra's representation adds visual noise that can confuse less experienced analysts.
- **Contextual Visibility:** Despite the sign-extension issue, Ghidra provided better contextual placement of the data, displaying the enc[] array directly within the function's disassembly view, whereas IDA Free mapped it as a separate "Pure data" segment.

E. Summary of Comparative Findings

The effectiveness of each tool across the tested scenarios is summarized in Table 1.

© 2026 IJMRSET | Volume 9, Issue 1, January 2026| DOI:10.15680/IJMRSET.2026.0901041

ISSN: 2582-7219 | www.ijmrset.com | Impact Factor: 8.206| ESTD Year: 2018|

International Journal of Multidisciplinary Research in Science, Engineering and Technology (IJMRSET)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

Table 1. Comparative Analysis of Effectiveness Between IDA Free and Ghidra.

Analysis Category	Specific Feature	IDA Free	Ghidra
Architecture & Privacy	Decompiler Model	Cloud-based: poses potential security risks when analyzing sensitive or confidential malware.	Built-in (Offline): allows for secure, isolated analysis without external data transmission.
File Format Detection	PE/MZ Signatures	Implicit: rarely displays explicit "MZ" or "PE" markers in the listing.	Explicit: clearly identifies and displays "MZ" and "PE" signatures in the string view.
Python Analysis	Packaging Detection	Effective: identifies PyInstaller markers and Python 3.8+ library references.	Effective: identifies PyInstaller markers and Python 3.8+ library references.
	Code Recovery	Limited: recovers bootloader only; cannot directly decompile Python bytecode.	Limited: recovers bootloader only; cannot directly decompile Python bytecode.
C Logic Reconstruction	Control Flow	Superior: reconstructs complex control flows (e.g., switch-cases) into clean, readable pseudocode.	Suboptimal: struggles with switch statements, often marking valid code as "unreachable blocks".
	Readability	High: uses specific data types and structures, enhancing clarity.	Moderate: relies on generic labels (e.g., FUN_00401000), requiring manual annotation.
Malware Analysis	Packer Detection (UPX)	Implicit: indicates packing via "destroyed imports" warnings; UPX names appear only in section headers.	Explicit: clearly lists UPX section names (e.g., UPX0, UPX1) in Program Trees, facilitating immediate detection.
Data Representation	Numeric Precision	Accurate: displays bytes as unsigned values (e.g., 0x9B), matching the original source.	Inconsistent: often defaults to sign-extended representation (e.g., 0xffffff9b), potentially confusing interpretation.

V. CONCLUSION AND FUTURE WORK

This study presented a comparative static analysis of two leading reverse engineering frameworks, Ghidra and IDA Free, evaluating their effectiveness across interpreted, compiled, packed, and encrypted binaries. The results demonstrate that while both tools can handle modern software artifacts, they exhibit distinct strengths that cater to different stages of the reverse engineering lifecycle.

Ghidra emerges as the superior tool for initial triage and artifact detection. Its ability to explicitly flag critical signatures such as "MZ/PE" headers and UPX section names (UPX0, UPX1) provides analysts with immediate insight into the file's structure and packing status. Furthermore, Ghidra's offline, built-in decompiler offers a significant advantage in terms of operational security, making it the preferred choice for analyzing sensitive or confidential malware samples where data leakage is a concern.

IDA Free, conversely, excels in deep logic reconstruction. The experimental data indicates that IDA Free's decompiler produces significantly cleaner and more readable pseudocode, particularly when handling complex control flow structures like switch-case statements. In the C application test, IDA Free correctly reconstructed the menu logic where Ghidra failed, marking valid code blocks as "unreachable". However, its reliance on a cloud-based decompiler introduces privacy risks that researchers must weigh against its superior code clarity.

ISSN: 2582-7219 | www.ijmrset.com | Impact Factor: 8.206| ESTD Year: 2018|

International Journal of Multidisciplinary Research in Science, Engineering and Technology (IJMRSET)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

In specific scenarios such as PyInstaller executables and encrypted ELF binaries, both tools showed similar limitations regarding direct code recovery, highlighting that static analysis often requires supplementation with specialized extractors or dynamic methods.

Ultimately, this research concludes that IDA Free and Ghidra should not be viewed as mutually exclusive competitors but as complementary instruments. An optimal open-source workflow involves utilizing Ghidra for initial file characterization and packer detection, followed by IDA Free for precise logic analysis of complex routines.

REFERENCES

[1] R. Clancy, "A Quick Guide to Reverse Engineering Malware," *EC-Council*, pp. 1–16, 2022.
[2] A. Shytiuk, "What is Reverse Engineering in Software Engineering," *EPAM Solutions Hub*, 2025. [Online]. Available: https://solutionshub.epam.com/blog/post/what-is-reverse-engineering-in-software-engineering.
[3] M. T. Suleman, "Malware detection and analysis using reverse engineering," *Int. J. Electron. Crime Investig.*, vol. 8, no. 1, pp. 109–123, 2024.
[4] G. Sugiarto, "Analisa dan identifikasi pola malware menggunakan dynamic malware analysis dan statics malware analysis," B.S. thesis, Universitas Kristen Petra, Surabaya, 2024.
[5] A. Harper *et al.*, *Gray Hat Hacking: The Ethical Hacker's Handbook*, 6th ed. New York: McGraw-Hill Education, 2022.
[6] O. Bhat *et al.*, "Comparison of 3 Reverse Engineering Tools: IDA Pro, Ghidra, and Angr," *ResearchGate*, 2019. [Online]. Available: https://www.researchgate.net/publication/333907927.
[7] National Security Agency, "Ghidra Software Reverse Engineering Framework," *nsa.gov*, 2019. [Online]. Available: https://ghidra-sre.org/.
[8] M. Arias, "Reverse Engineering for Malware Analysis: A Comparison of Ghidra and IDA Pro," *Peacock Scholarship*, Saint Peter's University, Jersey City, NJ, 2025.
[9] S. Alam, "STATOS: A portable tool for secure malware analysis and sample acquisition," *University of Hertfordshire Research Archive*, 2025.
[10] T. R. Devine, M. Campbell, M. Anderson, and D. Dzielski, "SREP+SAST: A comparison of tools for reverse engineering machine code to detect cybersecurity vulnerabilities," in *2022 Int. Conf. Computational Science and Computational Intelligence (CSCI)*, Las Vegas, NV, USA, 2022, pp. 862–869.
[11] J. Kim *et al.*, "A Framework to Quantify the Quality of Source Code Obfuscation," *Applied Sciences*, vol. 14, no. 12, p. 5056, 2024.
[12] M. Shankarapani *et al.*, "Malware detection using assembly code opcode frequency," in *2010 Int. Conf. on Security and Cryptography (SECRYPT)*, Athens, Greece, 2010.
[13] M. Sikorski and A. Honig, *Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software*. San Francisco: No Starch Press, 2012.

INTERNATIONAL STANDARD SERIAL NUMBER INDIA

INTERNATIONAL JOURNAL OF MULTIDISCIPLINARY RESEARCH

IN SCIENCE, ENGINEERING AND TECHNOLOGY

| Mobile No: +91-6381907438 | Whatsapp: +91-6381907438 | ijmrset@gmail.com |

www.ijmrset.com