NAME: NIKHIL BAGUL INTERN_ID: 277

Tool Name: x64dbg, x64dbgpy, x64dbg YARA Plugin

📄 **History:**

x64dbg was started in 2014 by **mrexodia** and contributors from the reverse engineering community. Designed as a modern replacement for OllyDbg, it was built to support both 32-bit and 64-bit Windows executables. Over time, a powerful plugin ecosystem grew around it — most notably **x64dbgpy** for Python scripting, and the **YARA plugin** for real-time memory scanning during debugging.

📌 **What Is This Tool About?**

This suite is a **modular debugger system** for Windows that combines core debugging power (x64dbg), dynamic scripting and automation (x64dbgpy), and malware signature scanning (x64dbg YARA). It helps reverse engineers analyze binaries at runtime, detect behaviors, automate tasks, and scan memory for known patterns — all from within a single interface.

⭐ **Key Characteristics / Features:**

1. Debugs both x86 and x64 binaries
2. Clean, modern GUI with modular views
3. Integrated disassembler and assembler
4. Supports Python scripting via x64dbgpy
5. Memory scanning with YARA rules
6. Conditional, memory, and hardware breakpoints
7. Live memory patching and hex editing
8. Commenting, labeling, and renaming
9. PE header, import/export analysis
10. Stack and thread inspection
11. Plugin SDK and plugin manager
12. Debug logging with filtering
13. Instruction tracing and logging
14. Graph view for control flow
15. Active community and documentation

🔧 **Types / Modules Available:**

- **x64dbg (Main Application)**
- **x64dbgpy** – Python Scripting Plugin
- **YARA Plugin** – Memory Scanner using custom or public YARA rules
- **TitanEngine** – Debugging backend
- **Plugin Manager** – Install/Manage add-ons
- **Scylla Plugin** – IAT repair/unpacking
- **PEB/DLL/Handle Viewer** modules
- **Symbol Loader & Function Resolver**

🎯 **How Will This Tool Help?**

- Debug malicious binaries without needing kernel-mode tools
- Automate repetitive reversing tasks with Python scripts
- Write unpacking scripts, automate breakpoints, or memory inspection
- Scan live memory space of malware using YARA signatures
- Extract or decrypt strings at runtime
- Build unpackers for packed binaries
- Detect runtime shellcode or injected payloads
- Use scripts to build custom behavioral triggers inside the debugger

---

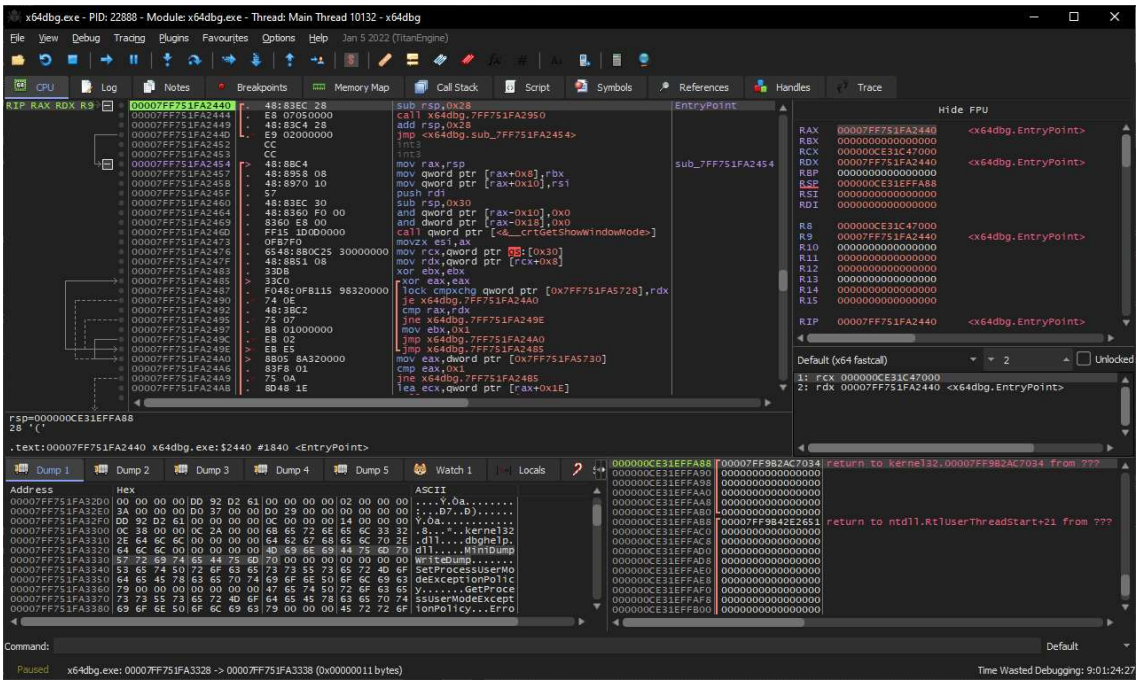
🖼️ **Proof of Concept (PoC) Images:**

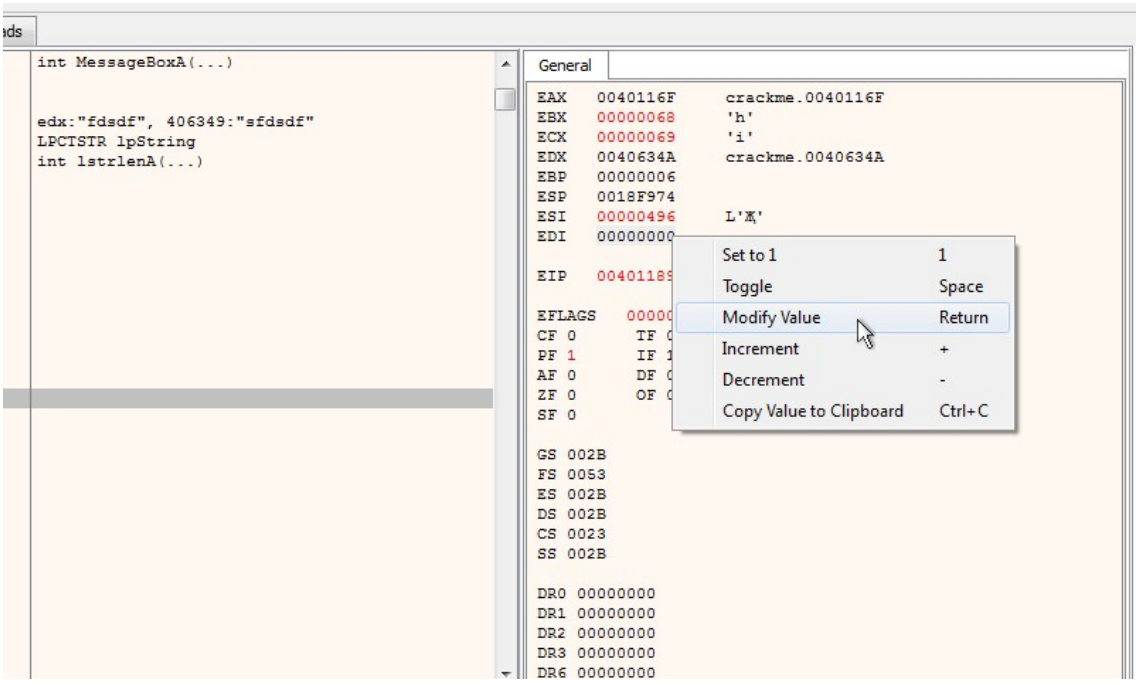
📸 *(Insert screenshots of the following for full visual proof)*

1. x64dbg main interface
2. Breakpoints and register view
3. Memory patching
4. Python console using x64dbgpy
5. YARA matches found inside a running malware sample
6. Control flow graph
7. Labeling and renaming functions
8. Hex dump of memory regions
9. Loaded plugins panel

10. Script editor showing automation routines

Proof of Concept (PoC) Images:





15-Liner Summary:

1. Powerful user-mode debugger for 32/64-bit Windows binaries

2. Plugin-based — extendable via C++, Python, or Lua
3. Clean GUI with split memory/register/thread views
4. Debug scripts and automate using x64dbgpy
5. Integrates with YARA for live memory rule matching
6. Supports patching, instruction injection, and rebuilding
7. Trace execution flow with fine-grained breakpoints
8. Annotate disassembly with notes and labels
9. Detect and track shellcode, payloads, injections
10. Graph control flow to trace logic paths
11. Plugin manager to download/install plugins from UI
12. Community-supported and frequently updated
13. Great tool for both beginners and advanced analysts
14. Useful for reverse engineering malware or crackmes
15. Completely free and open-source

⏱️ **Time to Use / Best Case Scenarios:**

- When analyzing packed malware to observe unpacking behavior
- To scan process memory for YARA indicators without dumping
- When dynamically reversing obfuscated binaries
- During development or debugging of exploits or malware loaders
- In CTF challenges involving Windows reverse engineering
- When performing behavioral malware analysis in a lab

🕵️ **When to Use During Investigation:**

- After identifying a suspicious binary and needing runtime analysis
- To analyze anti-debugging or code obfuscation techniques
- When traditional static tools (like Ghidra/IDA) fail to provide clarity
- For detecting runtime-decrypted payloads
- To catch reflective DLL injections or shellcode
- When confirming the presence of C2 or API hooking live in memory

---

👨‍💻 Best Person to Use This Tool & Required Skills:

Best User: Reverse Engineer, Malware Analyst, Exploit Developer, Security Researcher

Skills Required:

- Understanding of Windows internals (threads, memory, handles)
- Strong foundation in x86/x64 assembly
- Familiarity with debugging concepts (breakpoints, call stacks, heap inspection)
- Ability to write basic Python scripts (for x64dbgpy)
- Basic knowledge of YARA syntax and signature creation

🧩 Flaws / Suggestions to Improve:

- No kernel-mode debugging (limited to userland)
- Requires some manual configuration (e.g., plugin setup, Python path)
- Graph view is limited compared to Ghidra/IDA
- No built-in integration with deobfuscation frameworks
- Visualization of APIs and calls could be improved
- Memory-heavy on very large samples
- Plugin compatibility breaks on major updates occasionally

✅ Good About the Tool:

- Lightweight, portable, and fast
- Easy to script and automate with Python
- Combines debugging and scanning in one window
- Excellent for unpacking, tracing malware execution
- Rich plugin ecosystem (including Scylla, xAnalyzer, YARA, x64dbgpy)
- Great for both entry-level and expert analysts
- Completely open-source and transparent
- Community-backed, frequently updated with tutorials