**Underlying Technologies of the Tools**

Here's a breakdown of the core technologies underpinning each tool:

**GDB with PEDA**

* **GDB:**
  + **Debugger API:** Interacts directly with the target process to control its execution, examine memory, and set breakpoints.
  + **Scripting Language:** Python is used to extend GDB's functionality, enabling automation and custom analysis.
* **PEDA:**
  + **Python scripting:** Leverages Python's powerful scripting capabilities to automate tasks and provide advanced features.
  + **GDB's API:** Interacts with GDB to access information about the target process, such as memory maps, registers, and disassembled code.

**Immunity Debugger with Mona.py**

* **Immunity Debugger:**
  + **Debugger API:** Provides a graphical interface and programmatic access to the target process.
  + **Scripting Language:** Supports scripting to automate tasks and customize the debugger's behavior.
* **Mona.py:**
  + **Python scripting:** Leverages Python's scripting capabilities to automate pattern generation, memory analysis, and exploit development.
  + **Immunity Debugger's API:** Interacts with the debugger to control the target process, examine memory, and set breakpoints.

**Radare2**

* **Reverse Engineering Core:**
  + **Disassembly:** Uses advanced algorithms to disassemble binary code into assembly instructions.
  + **Debugging:** Provides debugging capabilities, including single-stepping, breakpoints, and memory inspection.
  + **Scripting Language:** Supports scripting in various languages, including Python, to automate tasks and extend functionality.
* **Analysis Engine:**
  + **Static Analysis:** Analyzes the binary code without executing it to identify potential vulnerabilities and code patterns.
  + **Dynamic Analysis:** Executes the binary code to observe its behavior and gather information about memory usage and control flow.

**Binary Ninja**

* **Reverse Engineering Core:**
  + **Disassembly:** Uses advanced algorithms to disassemble binary code into assembly instructions.
  + **Analysis Engine:** Analyzes the binary code to identify functions, data structures, and control flow.
* **User Interface:**
  + **Graphical User Interface:** Provides a user-friendly interface for navigating and analyzing binary code.
  + **Scripting Language:** Supports scripting to automate tasks and customize the analysis process.

**Angr**

* **Symbolic Execution:**
  + **Symbolic Execution Engine:** Explores all possible execution paths of a program by treating input values as symbolic variables.
  + **Constraint Solving:** Uses constraint solvers to analyze the possible values of variables at different program points.
* **Static Analysis:**
  + **Control Flow Analysis:** Analyzes the control flow of a program to identify potential vulnerabilities.
  + **Data Flow Analysis:** Analyzes the flow of data through a program to identify potential data leaks and security issues.

**Cutter**

* **Radare2 Core:**
  + Leverages Radare2's powerful reverse engineering capabilities for disassembly, debugging, and analysis.
* **User Interface:**
  + Provides a user-friendly graphical interface for interacting with Radare2's features.