Debugger Evasion, Technique T1622 - Enterprise

Archived: 2026-04-05 16:27:03 UTC

Adversaries may employ various means to detect and avoid debuggers. Debuggers are typically used by defenders to trace and/or analyze the execution of potential malware payloads.[1]

Debugger evasion may include changing behaviors based on the results of the checks for the presence of artifacts indicative of a debugged environment. Similar to Virtualization/Sandbox Evasion, if the adversary detects a debugger, they may alter their malware to disengage from the victim or conceal the core functions of the implant. They may also search for debugger artifacts before dropping secondary or additional payloads.

Specific checks will vary based on the target and/or adversary. On Windows, this may involve Native API function calls such as `IsDebuggerPresent()` and `NtQueryInformationProcess()`, or manually checking the `BeingDebugged` flag of the Process Environment Block (PEB). On Linux, this may involve querying `/proc/self/status` for the `TracerPID` field, which indicates whether or not the process is being traced by dynamic analysis tools.[2][3] Other checks for debugging artifacts may also seek to enumerate hardware breakpoints, interrupt assembly opcodes, time checks, or measurements if exceptions are raised in the current process (assuming a present debugger would "swallow" or handle the potential error).[4][5][6]

Malware may also leverage Structured Exception Handling (SEH) to detect debuggers by throwing an exception and detecting whether the process is suspended. SEH handles both hardware and software expectations, providing control over the exceptions including support for debugging. If a debugger is present, the program’s control will be transferred to the debugger, and the execution of the code will be suspended. If the debugger is not present, control will be transferred to the SEH handler, which will automatically handle the exception and allow the program’s execution to continue.[7]

Adversaries may use the information learned from these debugger checks during automated discovery to shape follow-on behaviors. Debuggers can also be evaded by detaching the process or flooding debug logs with meaningless data via messages produced by looping Native API function calls such as `OutputDebugStringW()`.[8][9]

Source: https://attack.mitre.org/techniques/T1622