Behind Heaven's Gate: Cross-Architecture Tracing of Mixed-Mode Execution in Windows Packers

Jaewoo Shim
The Affiliated Institute of ETRI
Daejeon, Republic of Korea
tlawodn94@gmail.com

Yeongung Park
The Affiliated Institute of ETRI
Daejeon, Republic of Korea
santapark5@gmail.com

Jaechan Moon
The Affiliated Institute of ETRI
Daejeon, Republic of Korea
answocks@gmail.com

Seokwoo Choi✉
The Affiliated Institute of ETRI
Daejeon, Republic of Korea
seogu.choi@gmail.com

Abstract

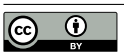
Windows packers and malware can execute 64-bit code within 32-bit processes using Heaven's Gate (HG), which switches execution to the 64-bit code segment. Most dynamic binary instrumentation (DBI) frameworks select one architecture per process and therefore cannot trace this transition. Prior work has demonstrated methods for detecting or bypassing individual gates but has not quantified the code executed through HG across protector versions. This measurement helps determine whether a black-box bypass that reproduces a gate's known effects without observing its internal execution provides sufficient behavioral fidelity. We present Janus, a hybrid Pin–Unicorn tracer that detects five HG transition forms, emulates their 64-bit execution windows, preserves state across mode transitions, forwards system calls to the native kernel, and invokes functions in the 64-bit `ntdll`. In VMProtect 3.10.5, a parallel 64-bit virtual machine produces values that affect subsequent 32-bit execution. A tracer that omits this window cannot recover the complete execution path to the protected program's entry point. In contrast, Themida 3.1.2 uses a 24-instruction window that performs a single system call. A longitudinal analysis of 22 versions of each protector shows that HG appears in only one of the evaluated Themida versions, whereas VMProtect retains and expands its 64-bit execution footprint. Evaluation on ACRID, HijackLoader, and Lumma samples further demonstrates HG tracing across multiple malware families, although some samples required manual neutralization of unrelated anti-analysis mechanisms before reaching the HG path. These results show that cross-architecture instrumentation is needed when subsequent execution depends on state produced within an HG window.

CCS Concepts

• **Security and privacy → Software reverse engineering**.

Keywords

Heaven's Gate, WoW64, dynamic binary instrumentation, anti-analysis, packers, emulation



This work is licensed under a Creative Commons Attribution 4.0 International License.
SURE '26, The Hague, Netherlands
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-3019-1/2026/11
https://doi.org/10.1145/3847158.3847382

ACM Reference Format:
Jaewoo Shim, Yeongung Park, Jaechan Moon, and Seokwoo Choi. 2026. Behind Heaven's Gate: Cross-Architecture Tracing of Mixed-Mode Execution in Windows Packers. In *2nd Workshop on Software Understanding and Reverse Engineering (SURE '26), November 15–19, 2026, The Hague, Netherlands.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3847158.3847382

1 Introduction

Automated malware analysis workflows, including unpacking, behavioral logging, and deobfuscation, frequently employ dynamic binary instrumentation (DBI) to obtain programmable instruction-level traces without modifying the target binary [20, 31]. On Windows platforms, these workflows must accommodate mixed-architecture execution. For example, a 32-bit process running under WoW64 may use Heaven's Gate (HG) to execute native 64-bit code via a far control transfer to the x64 code segment, then return to 32-bit mode. Commercial protectors, including VMProtect and Themida, and malware loaders such as HijackLoader [32] and Lumma [26], leverage this technique to evade analysis and prevent the evaluated single-architecture tools from tracing the 64-bit execution.

This transition introduces an analysis blind spot in packed and malicious binaries. A 32-bit DBI instance is limited to a 32-bit instruction decoder, register model, and code cache. Consequently, it may crash, fail to detect the transition, or incorrectly decode instructions during a 64-bit HG window. In our baseline experiments, Pin [17], DynamoRIO [3], and Frida [22] failed to trace the 64-bit window and exhibited distinct failure modes (Section 4.1). Common user-mode debuggers, such as x64dbg, IDA Pro, and Binary Ninja, also lost the execution trace at the 32-to-64-bit transition. User-mode record-and-replay tools, including WinDbg Time Travel Debugging (TTD), did not follow the transition. WinDbg's `.effmach` command enables the disassembler to switch between x86 and x64 modes and accurately decode instructions in both modes. However, manual tracing does not scale to HG windows containing millions of executed instructions.

Existing black-box bypass methods [9] can reproduce the known effect of a fixed gate but do not trace the intervening 64-bit instructions or capture the program state upon return. Consequently, detecting or bypassing HG does not reveal the code executed behind the gate. Prior research has not quantified the executed or

reachable 64-bit code surface across different versions of commercial protectors. These measurements allow the coverage of an HG analysis tool to be evaluated.

This work presents Janus, a cross-architecture tracer that integrates Pin and Unicorn to trace 32-bit binaries employing HG to execute 64-bit code. Janus instruments 32-bit execution with Pin and detects HG transitions at runtime. During each 64-bit window, it emulates instructions using Unicorn, forwards native system calls to the operating system, and invokes functions in the live 64-bit `ntdll` when needed. After emulation, Janus transfers the resulting state back to Pin and resumes 32-bit instrumentation, maintaining trace continuity.

Using Janus, we found that HG execution ranges from a 24-instruction, single-syscall window in Themida 3.1.2 to a parallel 64-bit VM with more than 400,000 statically reachable instructions in VMProtect 3.10.5, whose outputs are consumed by the surrounding 32-bit execution. In ACRID, HG instead serves as a reusable mechanism for invoking 64-bit `ntdll` functions from 32-bit code. These cases show that reproducing a fixed effect may suffice for a small window, whereas tracing large, state-dependent windows requires cross-architecture emulation.

This work presents three primary contributions:

(1) **An automated cross-architecture tracer for Heaven's Gate.** We design and implement Janus, which combines Pin and Unicorn to detect the five transition forms in our taxonomy and emulate their 64-bit execution windows while preserving state across mode transitions. Janus forwards native system calls and 64-bit `ntdll` calls and records instruction-level traces with resolved `module!export` names; its detection and emulation paths are validated using a 20-sample synthetic corpus.

(2) **A longitudinal characterization of HG usage in commercial protectors.** We analyze 22 versions each of Themida and VMProtect to identify when HG appears and how its 64-bit execution footprint changes across releases. HG occurs in only one evaluated Themida version, whereas VMProtect introduces a minimal HG window in version 3.5.1 and expands it into a parallel virtual machine beginning with version 3.8.1.

(3) **An analysis of state-dependent HG execution across packers and malware.** We show that VMProtect's parallel 64-bit virtual machine produces values consumed by subsequent 32-bit execution, making cross-architecture emulation necessary to recover the complete execution path. We further evaluate Janus on ACRID, HijackLoader, and Lumma samples, demonstrating HG tracing across multiple malware families while identifying cases in which unrelated anti-analysis mechanisms had to be manually neutralized before the HG path could be analyzed.

The remainder of the paper is organized as follows. Section 2 describes WoW64 and Heaven's Gate, defines the HG transition taxonomy, and specifies the threat model and assumptions. Section 3 presents the design and implementation of Janus. Sections 4 and 5 evaluate Janus using a synthetic corpus, commercial protectors, and malware samples. Sections 6–8 discuss the implications, limitations, related work, and conclusions.

2 Background and threat model

2.1 WoW64 and Heaven's Gate

WoW64 enables 32-bit x86 applications to execute on x64 Windows. A WoW64 process can transition between 32-bit compatibility mode and 64-bit mode within the same process address space [10, 19]. On x64 Windows, the code-segment selectors `0x23` and `0x33` conventionally reference 32-bit and 64-bit user-mode code segments, respectively.

"Heaven's Gate" uses this transition mechanism to execute 64-bit instructions from a 32-bit WoW64 context [23]. A program transfers control to the 64-bit code segment, executes 64-bit instructions, and then returns to 32-bit compatibility mode. Consequently, analysis tools restricted to the 32-bit execution path may fail to observe or interpret the 64-bit instruction sequence [9].

2.2 Heaven's Gate Variant Taxonomy

Heaven's Gate transitions can be classified into five syntactic variants based on the type of control-transfer instruction and the source of the 64-bit code-segment selector: direct far jump, direct far call, indirect far call, far return, and interrupt return. In the direct `jmp_far` and `call_far` variants, the selector is encoded in the instruction stream and can be identified statically. In the `retf` and `iretd` variants, the selector is obtained from the stack, whereas the indirect `call_far` variant obtains it from memory. (Table 1).

Table 1: Heaven's Gate transition idioms, example byte encodings, and whether detection is static or runtime-based.

Variant	Byte encoding (example)	Detection
jmp_far	`EA 78 56 34 12 33 00` *jmp far 0x33:0x12345678*	static / disasm (selector in bytes)
call_far (imm)	`9a 78 56 34 12 33 00` *call far 0x33:0x12345678*	static / disasm (selector in bytes)
call_far (`FF /3`)	`FF 1D 00 10 40 00` *call fword ptr [0x401000]* *(CS:EIP read from* `[m]`*)*	static site / runtime selector (selector in memory)

retf	`6A 33 E8 00 00 00 00`
	`83 04 24 0D CB`
	push 0x33; call $+5;
	add [esp],0xD; retf
		runtime hook
		on every `retf`
		(selector built
		at run time)

iretd	`9C 6A 33 E8 00 00 00 00`
	`83 04 24 05 CF`
	pushfd; push 0x33; call $+5;
	add [esp],5; iretd
	(frame [EIP][CS][EFLAGS])
		runtime hook
		on `iretd`
		(CS=0x33 in the
		iret frame,
		on the stack)

The `jmp_far` variant encodes the selector `0x33` directly in the far-pointer operand, enabling identification of the transition in signature-based disassembly scans. This variant has been observed in VMProtect [9], HijackLoader [32], and Lumma [26], as well as in prior analyses of Heaven's Gate [15, 25]. The direct `call_far` variant, encoded with opcode `0x9A`, similarly includes selector `0x33` in its far-pointer operand and can be detected using the same static signature. In contrast to `jmp_far`, `call_far` saves the current `CS`

and EIP on the stack. Consequently, the 64-bit instruction sequence can return to the original 32-bit context by executing retf.

When the selector is not encoded in the control-transfer instruction, a far-branch signature cannot statically identify the transition. The retf variant instead constructs a far pointer on the stack. In the position-independent sequence push 0x33; call $+5; add [esp], <$\delta$>; retf, the target EIP is computed at runtime by adding a fixed displacement <δ> to the pushed return address. An iretd instruction transitions to 64-bit mode when the CS field of its stack frame is set to 0x33. It pops a three-dword [EIP][CS][EFLAGS] frame and, by reloading EFLAGS atomically during the mode switch, can also clear the trap flag. Its selector, similar to that of retf, is constructed on the stack.

The indirect call_far variant, which is encoded as FF /3, retrieves its target offset and selector from a memory operand. An opcode-based static scan can identify the call site but cannot directly recover the selector 0x33. Similar to the direct form, the instruction saves the return CS:EIP and can return using retf.

2.3 Observed Limitations of Existing Dynamic Tools for Heaven's Gate

Our preliminary analysis indicates that the evaluated dynamic tools do not provide continuous tracing across the WoW64 transition from 32-bit to 64-bit execution. In the evaluated configurations, each DBI tool uses an architecture-specific decoder and instrumentation engine. Pin [17] dynamically translates and instruments 32-bit code using its x86 JIT compiler. The 64-bit instructions reached through a far transfer fall outside this execution model.

DynamoRIO [3] similarly decodes and translates basic blocks using an x86 execution context and code cache. Its x86 engine does not switch to a 64-bit decoder when a far transfer changes the execution mode within the same process. Frida Stalker [22] uses an architecture-specific backend for instruction decoding, code generation, and event collection. An x86 Stalker session does not activate an x64 backend for the 64-bit instructions reached through a Heaven's Gate transition.

Time Travel Debugging (TTD) [2] uses record-replay rather than DBI, but its trace remains associated with the initial 32-bit execution context. In our experiments, TTD did not record the 64-bit instructions executed after the mode transition. Section 4.1 presents the tool-specific failure modes and quantitative results for Pin, DynamoRIO, Frida Stalker, and TTD.

Hwang et al. [9] address the DBI limitation through a black-box bypass. Their method intercepts far control-transfer instructions, observes the execution state at the transition boundary, and bypass the transition reflecting its effects. This approach bypasses the transition without instrumenting the 64-bit instructions.

Among the evaluated interactive debuggers, only WinDbg could inspect both 32-bit and 64-bit execution within the same WoW64 process. WinDbg supports architecture switching through .effmach; however, tracing workloads containing millions of executed instructions incurs substantial overhead. In the configurations we examined, x64dbg, Binary Ninja, and IDA did not support continuous tracing across the 32-to-64-bit transition. Thus, WinDbg provides cross-mode functionality with limited scalability, whereas the other evaluated debuggers lack continuous cross-mode tracing. Janus addresses these limitations by emulating the 64-bit instructions under programmable instrumentation while preserving the surrounding 32-bit execution (Section 3).

2.4 Anti-Analysis Primitives within Heaven's Gate

Commercial protectors and malware may execute anti-analysis logic within the 64-bit instructions reached through Heaven's Gate. This logic includes debugger and DBI detection, timing checks, and direct system calls that remain outside the execution model of single-architecture analysis tools. Section 5.1 examines these mechanisms in VMProtect and evaluates how Janus detects, measures, and neutralizes them.

2.5 Threat Model and Assumptions

The analyst possesses the target binary and seeks a semantically faithful, instruction-level trace of its user-mode execution, including instructions executed in 64-bit mode. The adversary comprises the packer, protector, or malware logic contained in the binary. It controls its user-mode code, data, and execution flow; may transition between 32-bit and 64-bit modes using any Heaven's Gate variant described in Section 2.2; and may detect instrumentation through PEB or TEB inspection, timing checks, direct system calls, or deliberate faults.

We assume that the analyst can execute the binary under a DBI framework on an analyst-controlled Windows host. The analysis requires no kernel-mode component, hypervisor, or hardware-assisted virtualization. Protections that prevent the binary from executing are outside the scope of this work.

2.6 Detectability

Executing 64-bit instructions under Unicorn rather than on the native CPU can be detected by timing deviations, inconsistent cpuid results, or incomplete SIMD and x87 support (Section 6). We therefore do not claim that the emulation is undetectable.

Janus handles rdtsc using monotonic timestamps and passes through cpuid, maintaining consistency with the surrounding 32-bit execution. These measures do not eliminate general emulator detection. However, the termination predicates confirmed in Section 5.1 inspect the debugger state rather than the CPU or emulator identity, and none of the executed paths rely on unsupported SIMD or x87 behavior.

We apply no sample-specific spoofing beyond the anti-analysis handling used for the surrounding 32-bit execution. Each evaluated sample reaches its entry point under the same policy. Adversaries designed specifically to fingerprint Unicorn are outside the threat model. The objective is sufficiently accurate emulation for dynamic analysis, not an execution environment indistinguishable from native hardware.

3 Design and implementation

3.1 Architecture: a Pin tool plus an emulator

Janus executes two components built with distinct toolchains within a single address space. janus32.dll is a Pin tool that instruments 32-bit code and links against Pin's C runtime. hge.dll is a Heaven's

Gate emulator (HGE). Pin instruments only the 32-bit instruction stream; therefore, `hge.dll` emulates the 64-bit execution windows that Pin cannot trace. Figure 1 shows the resulting data flow.

`hge.dll` is a 32-bit Heaven's Gate emulator based on Unicorn. It links against the standard MSVC CRT because Unicorn's static libraries depend on the host CRT. The 32-bit Pin tool loads `hge.dll` into the WoW64 target when emulation is required. The two CRTs coexist within the same process, and their interaction is restricted to a narrow C application binary interface (ABI).

The ABI consists of several exported functions and a register-state structure. The input state contains the 16 general-purpose registers (GPRs), RIP, and EFLAGS; the output contains the return CS and the emulation outcome. This interface transfers the 64-bit machine state across each mode transition. The tool loads the emulator and resolves its exports during initialization. If either operation fails, Janus logs the error and terminates, preventing execution with an invalid emulator configuration.

3.2 Heaven's Gate detection

Janus detects the five transition variants described in Section 2.2. All variants invoke a common emulation handler. For the `jmp_far` variant and the immediate (`9A`) form of `call_far`, the 64-bit selector is encoded in the instruction. Janus statically identifies these variants and inserts a handoff at each trace tail ending in a far jump or far call to selector `0x33`.

For the remaining variants, the selector is determined at runtime and cannot be obtained from the static instruction bytes. The `retf` and `iretd` variants retrieve the selector from the stack, whereas the indirect (`FF /3`) form of `call_far` retrieves it from a memory operand. Janus inserts a hook before every `retf`, `iretd`, and indirect far call. The hook reads the pending selector from `[esp+4]` for `retf` and `iretd`, or from the far-pointer operand for `FF /3`. It invokes the emulation handler when the selector equals `0x33` and records the transition site upon its first invocation. The handler also records the detected variant, preserving the distinction among the five transition forms in the resulting trace.

The ACRID sample described in Section 5.2 loads the flat data selector `0x2b` into FS before its position-independent `retf` gate. This operation changes the FS base from the thread environment block (TEB) to zero. Pin assumes that FS retains its TEB-based value and consequently faults when its runtime performs an FS-relative access before the sample restores the register. Janus suppresses the FS load because the subsequent 64-bit window is emulated and the sample's original FS state remains unchanged.

3.3 Cross-architecture state marshaling

Each handoff is performed separately, but the 64-bit machine state must persist across Heaven's Gate transitions. Pin's `CONTEXT` exposes only the 32-bit-visible halves of the registers. Janus therefore maintains a thread-local shadow state containing the upper register halves and registers R8–R15. Upon entry, Janus reconstructs the full 64-bit registers from Pin's `CONTEXT` and the shadow state. Upon exit, it separates the resulting state into the corresponding 32-bit and shadow components. This procedure preserves values across distinct 64-bit execution windows.

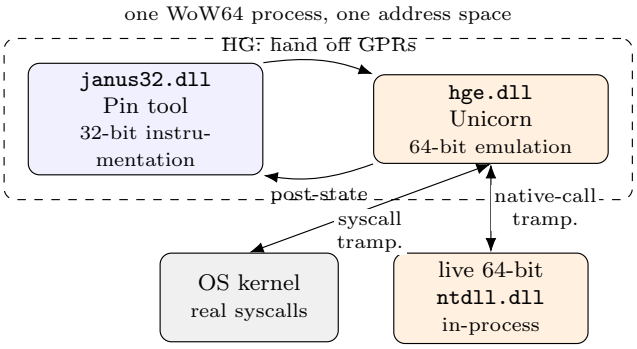


Figure 1: Janus architecture within a WoW64 process. The 32-bit Pin tool (`janus32.dll`) transfers each Heaven's Gate transition to the Unicorn-based emulator (`hge.dll`). The emulator uses trampolines to dispatch 64-bit operations to the OS kernel or the in-process `ntdll.dll` (Section 3.6).

The five transition variants have different stack semantics. The target RIP is obtained from the jump immediate for `jmp_far`, the stacked return address for `retf`, the pushed return frame for `call_far`, or the interrupt-return frame for `iretd`. HGE reproduces the corresponding semantics by adjusting RSP and, for `iretd`, EFLAGS. Unicorn then executes the 64-bit window until a far return to selector `0x23`, after which HGE returns the resulting machine state to Janus.

3.4 Syscall trampoline

Unicorn cannot execute a `syscall` against the operating-system kernel. HGE therefore dispatches each emulated `syscall` through a hand-assembled, position-independent trampoline. A Unicorn instruction hook invokes a shim that transfers the guest registers to a shared buffer and calls the trampoline. During initialization, HGE copies the trampoline to a readable, writable, and executable (RWX) page and patches its absolute addresses. Runtime address resolution allows the trampoline to operate under address space layout randomization (ASLR).

The trampoline saves the host context and enters 64-bit mode through a far return. It then loads the guest GPRs from the shared buffer. The guest RSP is loaded last so that the kernel can obtain stack arguments from the guest stack. The trampoline executes the actual `syscall`, stores the resulting registers in the shared buffer, and returns to 32-bit mode. Both mode-switch targets are computed at run time. HGE also suspends the other application threads during emulation, preventing them from observing the temporary mode switch. This approach has a known limitation: the kernel validates user pointers against the process's actual address space rather than Unicorn's mapped address space. Section 6 analyzes the resulting syscall-boundary limitation.

HGE uses a similar mechanism when a 64-bit execution window calls an `ntdll` function at an address greater than or equal to 4 GB, as in the ACRID X64Call gadget. An analogous native-call trampoline dispatches the indirect call to the live 64-bit function. HGE resolves each target to a `module!export` symbol at run time (Section 3.6) and records the resolved call in the trace.

3.5 Embedding Unicorn under Pin

HGE statically links Unicorn into `hge.dll` and identity-maps the required application pages into the emulator. The emulated 64-bit code and the surrounding 32-bit code therefore access the same virtual addresses, making emulated writes visible upon return to the 32-bit path. However, not all process mappings are replicated in Unicorn, resulting in the syscall-boundary limitation discussed in Section 6.

Embedding Unicorn under Pin creates a conflict in how access violations are handled. Unicorn reserves a 32 MB `code_gen_buffer` and commits its pages lazily through a vectored exception handler (VEH). Under Pin, access violations originating from JIT-translated code are intercepted before Unicorn's VEH can commit the required page, causing the first Tiny Code Generator (TCG) write to fault in `tcg_prologue_init`. We therefore modified Unicorn to precommit the translation buffer. HGE also registers a first-in-chain VEH for fallback page commitment and exception logging.

The ablation results in Section 4.1 show that precommitment is required for JIT-intensive workloads. Because the synthetic benchmarks do not exercise this condition, the integration is validated through end-to-end execution of the packer sample.

3.6 Native 64-bit calls behind the gate

Some 64-bit windows call an `ntdll` function at an address above 4 GB, as in the ACRID X64Call gadget. Unicorn cannot directly emulate these calls because its WoW64-range mappings do not include the target addresses. HGE therefore dispatches each indirect call through a native-call trampoline derived from the syscall mechanism described in Section 3.4.

The native-call trampoline enters 64-bit mode, loads the marshaled guest GPRs, and loads guest RSP last. It then calls the live `ntdll` target and returns the resulting registers. A companion trampoline performs the reads above 4 GB required for target resolution.

Because each call target is determined at run time, HGE resolves its symbolic name. At each dispatch, HGE traverses the 64-bit `PEB→Ldr→export table` and records the corresponding `module!export` symbol and arguments. This information identifies calls that would otherwise appear in the trace only as runtime addresses.

This execution path exposes an additional interaction between Unicorn and Pin. Under Pin, the stack pointer restored by `uc_emu_stop`'s `siglongjmp` lies outside the `[StackLimit, StackBase]` range recorded in the thread's TEB. The CRT therefore rejects the unwind during hardened `longjmp` validation and terminates the process through `KiFastFailDispatch`, bypassing structured and vectored exception handling. HGE temporarily extends the TEB-defined stack bounds during the resume loop and restores them afterward, allowing the unwind without modifying processor-level exception handling.

4 Evaluation

Section 4.1 validates Janus using a synthetic corpus that reproduces the Heaven's Gate mechanisms identified in our analysis. It also compares Janus with three unmodified DBI frameworks—Pin, DynamoRIO, and Frida—and WinDbg Time Travel Debugging (TTD). Section 4.3 presents a longitudinal survey of Themida versions. Sections 4.4 and 4.5 characterize the endpoints of the observed Heaven's Gate (HG) usage spectrum: minimal HG in Themida 3.1.2 and a parallel-VM implementation in VMProtect 3.10.5.

The primary experiments were conducted on a single workstation running the x64 edition of Windows 11. We used Pin 4.2 and a 32-bit build of the Unicorn backend incorporating the patch described in Section 3.5. Each packer sample was generated by applying the corresponding protector to the same base program, a small interactive dice game (`original.exe`), to ensure comparability across protectors.

4.1 Validation and baseline comparison using a synthetic corpus

We constructed a purpose-built corpus of 20 PE32 binaries (tests 1–20) to validate Janus's HG detection and emulation paths and to provide a common workload for baseline comparison. The corpus is organized into four groups (Table 2). Each binary performs a Heaven's Gate transition, executes 64-bit code, returns through `retf`, and exits with a status determined by the 64-bit computation.

Tests 1–11 evaluate 64-bit instruction execution within an emulated HG window. They cover shared-memory access, cross-transition register persistence, RIP-relative and SIB addressing, arithmetic and logical operations, conditional branches, stack operations, indirect calls, bit operations, sign and zero extension, and segment-aware addressing. Janus does not implement a separate 64-bit instruction decoder; Unicorn decodes and executes all 64-bit instructions. These tests therefore evaluate the integration of Unicorn with Janus, including memory synchronization and cross-mode state transfer.

Tests 12–14 evaluate more complex operand-size and register-update semantics. Test 12 covers three-operand `IMUL`, `MOVSXD`, `CDQE`, and `CQO`, including full 64-bit writes to the shadow register file. Test 13 covers `REP MOVSQ`, `XADD`, `CMPXCHG`, and `REP STOSQ`. Test 14 covers `CMOVcc` variants.

Tests 15–17 reproduce mechanisms observed in the ACRID case study (Section 5.2) and provide sample-independent regression tests. Test 15 reproduces ACRID's position-independent `retf` gate preceded by the `mov fs, 0x2b` reload that triggers the FS blocker. Tests 16 and 17 exercise the X64Call path by invoking a 64-bit function through the gate. Test 17 calls the actual `ntdll!RtlGetVersion` routine at an address of ≥4 GB. It exercises the read64 trampoline, high-page mapping, and runtime resolution of `ntdll.dll!RtlGetVersion`.

Tests 18–20 cover the structural gate idioms defined in Section 2.2 that are not in the other samples. Test 18 executes the gate from an RWX region allocated by `VirtualAlloc`, rather than from the loaded image, to exercise detection in executable memory allocated by the sample. Test 19 implements a `call_far` gate as a `FF /3` far call through a memory far pointer with selector `0x33`. Test 20 implements an `iretd` gate using a three-dword `[EIP][CS][EFLAGS]` frame with `CS = 0x33`. These tests exercise the executable-region tracker and the detector and emulator paths for `call_far` and `iretd`.

To validate Janus, we executed each binary natively and with HG emulation enabled and compared the resulting exit codes. All 20 binaries produced identical exit codes in both configurations.

Table 2: Synthetic correctness corpus: per-test decoder, addressing, and feature concern exercised.

Test	Decoder / addressing / feature concern exercised
1	minimal HG round-trip + shared-memory read/modify/write-back
2	cross-HG register persistence
3	RIP-relative load/store + reg-reg ALU
4	`0x81`/`0x83` immediate-arith group (every `/digit`, imm8+imm32)
5	SIB addressing (scale·index+disp32) + `C1` shift/rotate
6	`Jcc` conditional branches + `push`/`pop`
7	indirect `call` (`FF /2`), `LEA`, `TEST`, `NOT`, `BSWAP`
8	`INC`/`DEC`, `D1`/`D3` shifts (by 1 / by `CL`), bit-test (`BT`/`BTS`/`BTR`/`BTC`)
9–11	sign/zero extension, segment-aware addressing
12	3-operand `IMUL`, `MOVSXD`, `CDQE`/`CQO`
13	`REP MOVSQ`, `XADD`, `CMPXCHG`, `REP STOSQ`
14	`CMOVcc` variants
15	position-indep. `retf` gate + `mov fs,0x2b` (ACRID)
16–17	X64Call gadget; 17 calls real ≥4 GB `ntdll` export
18	gate from `VirtualAlloc`'d RWX region (exec-region tracking)
19	`call_far` gate (`FF /3` via `m16:32`, `CS = 0x33`)
20	`iretd` gate (3-dword `[EIP][CS][EFLAGS]` frame)

For the baseline comparison, we evaluated the same binaries using bare Pin 4.2, DynamoRIO 11.91, Frida 17.10.1 Stalker, and WinDbg Time Travel Debugging (TTD). The comparison measured whether each system captured instructions from the 64-bit HG execution window. Janus traced all 20 windows; none of the baseline systems successfully traced or recorded any of them (Table 3).

Bare Pin, which provides the instrumentation substrate for Janus, does not enter the 64-bit window. Fifteen samples fault at the gate; the other five are rejected at the far transfer or blocked by the gate setup sequence. In test 19, the far call aborts within Pin's 32-bit JIT. In test 20, Pin permits the `iretd` transition to execute natively, after which the process faults.

Frida Stalker treats the gate—either `jmp far 0x33` or the far return in the `retf` and X64Call samples—as an opaque transition and emits it without instrumentation. Control returns to Stalker at the first 32-bit instruction after the transition. The application executes the 64-bit window natively, but Stalker does not trace it.

DynamoRIO follows the transition but continues decoding in 32-bit mode. It interprets the `0x48` REX.W prefix as `dec eax`, causing the code cache to misexecute. Consequently, all 20 processes fault or abort: 14 produce `0xC0000005`, and the remaining six abort in DynamoRIO or return corrupted exit codes.

WinDbg TTD uses record–replay rather than instrumentation. For all 20 samples, TTD records the 32-bit stream through the Heaven's Gate far transfer but records no instructions from the 64-bit window. Each program nevertheless executes that window natively during recording. Every trace contains the gate and the subsequent return to 32-bit mode, but no intervening instruction with `CS=0x33`. These results are consistent with an x86-32 tracing engine that lacks a 64-bit decoder for the WoW64 window.

4.2 Performance

We measured end-to-end wall-clock time for the two case-study endpoints in three configurations: native execution, Pin without an analysis tool (the single-architecture baseline), and Janus with emulation. The test system used an Intel Core Ultra 9 185H processor and Windows 11 build 26200. Table 4 reports the median of three runs. Run-to-run variation was within a few hundred milliseconds. A concurrent watcher dismissed the anti-debug message box to exclude user-response latency.

Table 3: Mainstream DBI frameworks and Time Travel Debugging on the 20-sample synthetic Heaven's Gate corpus.

Framework	64-bit window	Outcome (20 samples)
Pin	not entered	access violation 15; far transfer refused 5
Frida	native, untraced	runs natively; 64-bit window traced: 0/20
DynamoRIO	misdecoded 32-bit	crash 14; abort/corrupt 6
WinDbg TTD	native, untraced	32-bit stream recorded; 64-bit window: 0/20
Janus	**emulated**	**traced: 20/20**

Table 4: End-to-end runtime at the two ends of the HG-usage spectrum (median of 3 runs). Pin-only is time-to-failure; Janus reaches OEP.

Sample	native (s)	Pin-only (s)	Janus →OEP (s)	emul. insns
Themida 3.1.2	0.17	1.25†	7.5 (44×)	48
VMProtect 3.10.5	0.94	1.22‡	54.2 (58×)	9.6 M

†abnormal exit; ‡gate fault. "emul. insns" totals Unicorn-executed 64-bit instructions across all HG transitions (2 for Themida, 14 for VMProtect).

Bare Pin does not complete either run. Its single-architecture JIT aborts at the `retf` gate for Themida and raises an access violation at the `jmp_far` transition for VMProtect. Janus is therefore the only instrumented configuration that reaches OEP, and its runtime is compared with native execution. The sources of overhead differ between samples. For Themida, only 48 instructions are emulated; the 44× slowdown primarily reflects Pin's 32-bit instrumentation and anti-debug handling. For VMProtect, the 58× slowdown includes approximately 9.6M emulated 64-bit instructions executed at approximately 180k instructions/s through the Unicorn-in-Pin path. The transition windows contain approximately 690k instructions on average, with a minimum of 24. The VMProtect trace reaches the OEP in 54.2 seconds despite the parallel VM workload. ACRID was excluded from the timing analysis. Its three 24-instruction X64Call windows have an emulation workload comparable to the minimal Themida case.

4.3 A longitudinal survey of 22 Themida versions

We analyzed 22 Themida versions, ranging from 2.1.8 to 3.2.5, from a licensed collection. These versions sample the release history but do not include every point release. For each version, we protected `original.exe` with the anti-debug option enabled and traced the

Table 5: Themida 3.1.2's two HG handlers. Each is an 18-instruction handler (24 emulated steps incl. the stub) that reaches `NtQueryInformationProcess` via `call rsi`.

Site	rdx	Query	64-bit return
1	0x07	ProcessDebugPort	0x0
2	0x1e	ProcessDebugObjectHandle	0xC0000353

resulting binary using a `retf`-aware HG detector with emulation disabled. The detector records an installation event when it first observes a `retf` whose stacked selector is `0x33`, using the runtime-only detection method described in Section 3.2. Events are written synchronously, which preserves the record when Pin aborts at the far return.

Heaven's Gate occurs in one of the 22 versions: Themida 3.1.2. The 11 versions through 3.1.1 and the 10 versions from 3.1.3 onward produce no HG detections and terminate through `ExitProcess` with exit code 0 under detection-only execution. This set includes versions 3.1.1 and 3.1.3, which immediately precede and follow 3.1.2. Version 3.1.2 triggers HG detection but does not terminate normally when emulation is disabled because Pin aborts at the far return. The emulation experiment in Section 4.4 executes this version to completion.

The observed distribution is consistent with the introduction of a `retf`-based Heaven's Gate mechanism in version 3.1.2 and its removal in version 3.1.3. The result also shows that point releases of a commercial packer can differ in their anti-analysis mechanisms, making exact version identification necessary for comparisons across studies. The sole detected instance uses a position-independent `retf` variant, which cannot be identified by a far-branch selector signature (Section 3.2). This observation supports the use of runtime `retf` monitoring as a packer-independent detection method.

4.4 Themida 3.1.2: minimal Heaven's Gate

Themida 3.1.2 represents the minimal endpoint of the observed HG-usage spectrum. With emulation enabled, the sample completes execution after both HG transitions return to 32-bit mode.

The sample contains two HG sites. Both use the position-independent `retf` sequence `push 0x33; call $+5; add [esp], <delta>; retf`. Because the 64-bit target EIP is computed at runtime, these sites are not detected by a far-branch selector signature (Section 2.2). Each transition executes 24 emulated 64-bit instructions: an 18-instruction handler and the invoked stub.

Both handlers follow the same structure. Each handler realigns the stack for the 64-bit ABI, initializes the argument registers, and calls the 64-bit `ntdll!NtQueryInformationProcess` stub through `rsi`. The current-process pseudo-handle is passed in `rcx = -1`; the handlers differ only in the information class passed in `rdx` (Table 5). The stub's `syscall` accounts for the remaining emulated instructions. The tail constructs a `0x23`:EIP far pointer on the stack and returns to 32-bit mode through `retf`.

Themida 3.1.2 implements Heaven's Gate as a 24-instruction auxiliary routine without a second VM, a dispatcher, or obfuscation other than the runtime-computed target. In contrast, VMProtect 3.10.5 implements a parallel VM. The two samples differ by four orders of magnitude in the amount of reachable 64-bit code (Section 6).

4.5 VMProtect longitudinal survey

We evaluated 22 VMProtect versions from 2.13.6 through 3.10.5 using the licensed collection described in Section 4.3. The versions sample the release history but do not include every point release. We used each version to protect `original.exe` in its default and anti-debug configurations, producing 44 binaries. A dedicated detection pass using the analyzer from Section 3.2 measured HG presence and the reachable 64-bit instruction set. A version was classified as using HG if the analyzer observed at least one `jmp far 0x·, 0x33` tail. Detected entry targets seeded the reachability analysis.

The `jmp_far` gate executes independently of debugger masking. For example, the two gate installations in the anti-debug build of VMProtect 3.5.1 are identical with and without masking. We therefore used a single detection channel for all 44 binaries to ensure that HG-presence and reachable-surface measurements were comparable across configurations (Figure 2).

VMProtect versions through 3.5.0 contain no detected HG transition, and the user-mode evasion baseline reaches OEP. HG first appears in version 3.5.1 as a minimal window. Versions in this range contain 5–9 `jmp_far` entry sites and 319–445 statically reachable 64-bit instructions. This scale is comparable to, although larger than, Themida 3.1.2's 24-instruction auxiliary routine and does not constitute a parallel VM.

We performed a configuration sweep across 22 versions, 2 build types, and 2 masking states, for a total of 88 emulated runs. All versions reach OEP when masking and emulation are enabled. The direct user-mode anti-debug path is independent of HG and is treated as part of the baseline anti-debug handling. A separate user-mode trap-flag check occurs starting with version 3.1.2. When masking is disabled, this check terminates anti-debug builds with `STATUS_SINGLE_STEP`, including versions without an HG gate. Trap-flag evasion neutralizes this check. From version 3.5.1 onward, default builds reach OEP with emulation alone, whereas anti-debug builds require both emulation and masking. Without masking, emulation completes the HG window, but execution subsequently terminates at the trap-flag check.

The minimal `jmp_far` window present in versions 3.5.1–3.7.2 expands into a parallel VM in version 3.8.1. The statically reachable 64-bit surface increases by approximately three orders of magnitude, from approximately 445 instructions in version 3.7.2 to approximately 384k in version 3.8.1. It remains between 330k and 432k through version 3.10.5. HG occurs continuously in every evaluated build from version 3.5.1 onward. Thus, VMProtect introduces minimal HG in version 3.5.1 and expands it into a parallel VM in version 3.8.1. This longitudinal pattern differs from Themida, in which HG occurs only in version 3.1.2 within the surveyed releases.

Three qualifications apply to these counts. First, the longitudinal analysis uniformly seeds every detected `jmp_far` target, whereas the case-study analysis additionally seeds the common dispatcher.

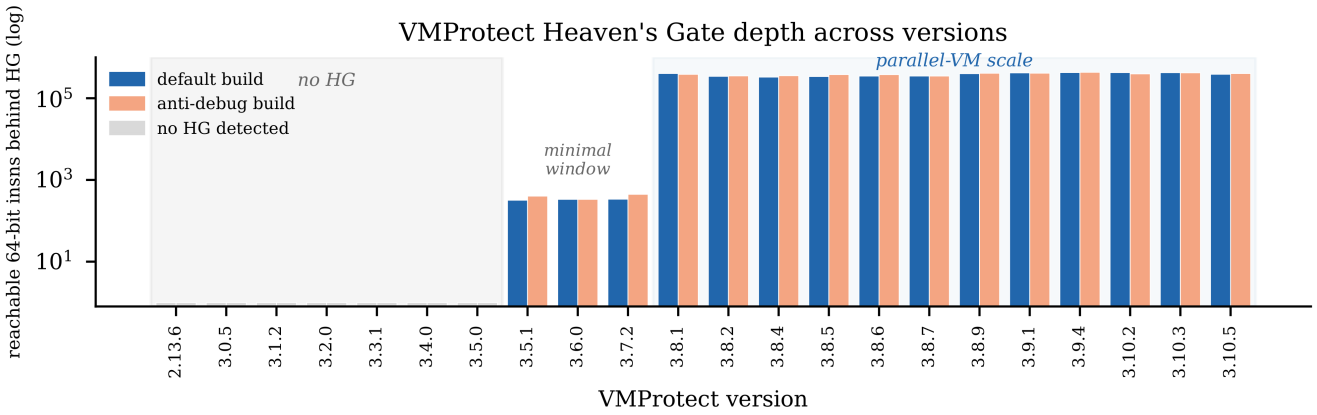


Figure 2: Statically reachable 64-bit instructions behind Heaven's Gate per VMProtect version (log scale), default vs. anti-debug builds. HG is absent through 3.5.0, minimal at 3.5.1–3.7.2, and grows to parallel-VM scale from 3.8.1 onward.

Both analyses count distinct instructions reachable by direct control flow. The longitudinal results are therefore conservative lower bounds and differ from the curated count for version 3.10.5: approximately 403k versus 427,565 instructions. Second, the observed HG onset in version 3.5.1 revises an earlier classification of versions 3.5.1–3.7.2 as containing no HG. All six builds contain the `jmp_far` idiom, which executes independently of debugger masking, although the minimal HG window is not required to reach OEP in this range. Third, HG becomes required for successful execution in version 3.8.5, when the emulation trace increases from approximately 69 MB to approximately 1.4 GB. Thus, the static `jmp_far` surface reaches parallel-VM scale in version 3.8.1 (Fig. 2), whereas the transition becomes required in version 3.8.5. All versions reach OEP when masking is enabled.

Version 3.10.5 is the parallel-VM endpoint of the survey. Its `jmp_far` entry sites reach 427,565 distinct instructions through a common dispatcher. Section 5.1 analyzes this version in detail. Across the two endpoints, the same WoW64 mechanism conceals 24 instructions in Themida 3.1.2 and 427,565 instructions in VMProtect 3.10.5, a difference of four orders of magnitude.

5 Case studies

We examine a detailed VMProtect case study and evaluate Janus on malware samples from the ACRID, HijackLoader, and Lumma families. We analyze ACRID in detail and use the additional families to assess whether Janus can trace HG execution in the presence of other malware-specific analysis obstacles.

5.1 VMProtect 3.10.5: a parallel 64-bit VM

5.1.1 Entry sites and the common dispatcher. VMProtect 3.10.5 enters 64-bit execution through the `jmp_far` variant of Heaven's Gate (`jmp far ptr 0x33:imm32`). A representative run executes four entries, all of which converge on a common 64-bit dispatcher for the verification machinery analyzed below. Pin reports 5–9 candidate sites across runs because of debugger masking, ASLR, and speculative tracing; candidates without a subsequent emulated entry are not executed.

5.1.2 Static reachability and the anti-analysis surface. Forward reachability is computed from the Heaven's Gate entries and common dispatcher observed in the representative run. The analysis follows direct calls, jumps, and conditional branches and terminates at indirect dispatches. This procedure identifies 427,565 distinct instructions reachable through direct control flow. Instructions reachable only through the VM's indirect dispatch are excluded; the resulting count is therefore a conservative lower bound on the size of the 64-bit code.

The count includes the sample's 64-bit code executed after the transition. It excludes the 32-bit instruction stream, code following the `retf` transition back to 32-bit mode, and the 64-bit `ntdll` functions invoked from the window. Within this set, 383 instructions belong to nine classes commonly associated with anti-analysis: `rdtsc`, `cpuid`, direct `syscall`/`sysenter`, `int N`/`int3`, `iretd`, and privileged instructions that induce faults (Table 6). These labels identify instruction classes rather than inferred intent. Each class may contain legitimate operations, junk instructions, or dead code, and the VM-obfuscated body cannot be fully classified statically. The counts therefore provide an upper bound on the active verification surface rather than an exact enumeration. Behavioral intent is assigned only to the dynamically observed syscall sequence in the anti-debug build (Section 5.1.3).

Table 6: Statically reachable instances of instruction classes commonly used for anti-analysis in VMProtect 3.10.5's 64-bit code (reachable from the representative run's HG entries + dispatcher). Counts bound the verification surface; they are not a census of confirmed checks.

Category	Count	What it probes
`rdtsc`	99	timing / single-stepping latency
`cpuid`	70	hypervisor & feature fingerprint
`syscall`	54	direct kernel entry (hook bypass)
`sysenter`	53	legacy direct kernel entry
`iretd`	40	deliberate-fault / SEH control flow
`int3`	34	breakpoint / debugger-presence fault
`int N`	27	deliberate-fault primitives
`int 0x2e`	3	legacy syscall gate
`wbinvd/invd`	3	privileged-instruction fault probe

The distribution indicates multiple potential check classes. The prevalence of `rdtsc` and `cpuid` is consistent with timing analysis and environment fingerprinting. The number of `iretd`, `int N`, and `int3` instructions is consistent with fault-based dispatch, in which the VM raises exceptions and processes them through its own 64-bit structured exception handling (SEH) path.

The 64-bit code dynamically resolves syscall numbers. Each `syscall` instruction is preceded by a `call` to a runtime helper that computes the service number, rather than by the conventional `mov eax, imm32`. Consequently, a backward scan does not recover an immediate service number.

5.1.3 Dynamically executed syscalls: plain vs. anti-debug variant. Static reachability identifies potentially executable primitives but does not determine which primitives are exercised during a specific run. The HGE syscall trampoline (Section 3.4) records each syscall dispatched by the 64-bit VM, allowing the executed surface to be measured directly. Both VMProtect variants of the same program were executed under emulation in sample-isolated runs. Each service number was resolved against the host's 64-bit `ntdll.dll`.

Of the 54 statically reachable syscall sites, the plain variant executes 14 syscalls corresponding to six distinct Nt functions. The anti-debug variant executes 30 syscalls corresponding to 12 distinct Nt functions. Both variants execute the same baseline sequence (Table 7, top group): eight calls to `NtProtectVirtualMemory`, followed by a section-mapping sequence comprising `NtOpenFile`, `NtCreateSection`, `NtMapViewOfSection`, `NtUnmapViewOfSection`, and `NtClose`. This sequence implements VMProtect's baseline self-protection in both variants. It changes page protections around virtualized code and creates a self-mapping used to compute a CRC over the image. The CRC operation detects modifications such as software breakpoints and inline hooks. Unlike the additional queries in the anti-debug variant, however, it does not query the operating system for an attached debugger.

The 16 syscalls executed only by the anti-debug variant constitute its 64-bit anti-debugging sequence (Table 7, bottom group). The `NtQueryInformationProcess` calls query `ProcessDebugPort`, which returns a port value of 0, and `ProcessDebugObjectHandle` using the current-process pseudo-handle. `NtQueryObject` examines the handle namespace. `NtSetInformationThread` uses the `ThreadHideFromDebugger` information class, and `NtCreateDebugObject` attempts to create a debug object for self-debugging.

Table 7: 64-bit syscalls actually executed in the HG window, resolved to Nt names against the host's 64-bit `ntdll`. Top group: the variant-invariant VMProtect baseline self-protection (page-protection toggles plus a CRC anti-tamper self-check). Bottom group: the anti-debug battery, present only in the anti-debug build. *SSN*: system service number.

Nt function	SSN	Plain	Anti-debug
`NtProtectVirtualMemory`	80	8	8
`NtCreateSection`	74	1	1
`NtMapViewOfSection`	40	1	1
`NtUnmapViewOfSection`	42	1	1
`NtOpenFile`	51	1	2
`NtClose`	15	2	3
`NtQueryInformationProcess`	25	0	6
`NtQueryObject`	16	0	3
`NtSetInformationThread`	13	0	2
`NtCreateDebugObject`	171	0	1
`NtSetInformationProcess`	28	0	1
`NtQueryInformationThread`	37	0	1
Total		**14**	**30**

Themida 3.1.2 and VMProtect issue the same `ProcessDebugPort` and `ProcessDebugObjectHandle` queries through `NtQueryInformationProcess`, despite executing 24 and more than 400,000 statically reachable instructions, respectively. The two VMProtect variants differ by 16 additional 64-bit syscalls; the plain variant issues no debugger-detection syscall within the Heaven's Gate window. The 32-bit path subsequently consumes the results of the 64-bit anti-debugging operations.

The `ProcessDebugObjectHandle` query also evaluates syscall-parameter fidelity. A deliberately misaligned buffer returns `STATUS_DATATYPE_MISALIGNMENT`, confirming preservation of the supplied alignment. Two other calls return `STATUS_ACCESS_VIOLATION` despite targeting buffers that `VirtualQuery` identifies as committed and writable, indicating incomplete preservation of the user-mode address space across trampoline calls. These calls are excluded from the anti-debugging results, and the address-space limitation is discussed in Section 6.

5.1.4 The 64-bit code is VMProtect's VM. The 427,565 reachable 64-bit instructions implement VMProtect's virtualization machinery. Static analysis identifies thousands of indirect-dispatch sites but no recoverable handler table; candidate accesses use `lea reg, [regK + disp]`, indicating register-computed dispatch. During execution, the virtual instruction pointer (`r12` in this build) traverses several hundred handlers. The resulting structure implements VMProtect's 32-bit virtualization architecture in 64-bit code, placing its verification logic outside the visibility of the 32-bit DBI.

VMProtect generates a distinct VM for each packing operation. Across three packings of the evaluated program and one packing of an unrelated program, the reachable surface ranged from 390,000 to

434,000 instructions. The virtual instruction pointer used a different register in each build: `rsi`, `r10`, `r12`, or `r15`. The counts of instruction classes associated with anti-analysis also varied. The selected register and exact instruction counts are therefore build-specific. In contrast, the use of a comparably sized parallel 64-bit VM, accessed via Heaven's Gate, was consistent across the evaluated packings and programs.

5.1.5 Why the gate must be executed. An unmodified Pin configuration cannot trace VMProtect 3.10.5 beyond the initial `jmp_far` transition to 64-bit mode and terminates with an access violation. Handling this transition alone is insufficient because the surrounding 32-bit code depends on values produced by the 64-bit VM. These values include outputs from the dynamically numbered system calls described in Section 5.1.3, `rdtsc`, `cpuid`, and fault-based checks. A 32-bit evasion mechanism cannot generally reconstruct these values.

The need to execute the transition depends on the complexity of the protected code. Themida 3.1.2 executes 24 instructions and one system call during the WoW64 transition (Section 4.4). A black-box bypass (Section 7) may suffice in this limited case. In contrast, VMProtect 3.10.5 uses an extensive 64-bit VM. This VM performs integrity and anti-debugging checks and returns values consumed by the 32-bit path. Tracing must execute the transition and preserve these values to recover the original entry point (OEP).

Table 8: ACRID's three Heaven's Gate calls during unpacking, named by the runtime resolver. All reach 64-bit `ntdll` and return results from the native 64-bit functions.

#	Resolved target	Class / arg	Purpose
1	`RtlGetVersion`	`rcx`=&buf	OS version (PEB64)
2	`NtQuerySystemInfo.`	class 0, `r8`=0x40	#CPUs, page/mem size
3	`NtQuerySystemInfo.`	class 1, `r8`=0xC	processor architecture

5.2 ACRID: Heaven's Gate as a generic 64-bit call primitive

The preceding packer samples are licensed builds of the same benign program, enabling comparisons across protectors under a common input. This configuration does not represent the use of Heaven's Gate in deployed malware. Janus was therefore evaluated without modification on ACRID [14], a 32-bit C++ information stealer obtained from VirusTotal. The sample was executed only in an isolated VM without outbound network access (Section 6.4).

Unlike Themida 3.1.2 and VMProtect 3.10.5, ACRID uses Heaven's Gate as a reusable thunk for invoking 64-bit `ntdll` functions from 32-bit code. Observing its environment reconnaissance requires emulating the window and resolving the call targets at runtime.

ACRID uses the `retf` transition in position-independent form (`push 0x33; call $+5; add [esp], <delta>; retf`). Because the 64-bit target is computed at run time, the transition is not identified by a static far-branch selector signature. This form matches the idiom used by Themida 3.1.2 and is detected through the same runtime far-return hook (Section 3.2).

Before the transition, the 32-bit code executes `mov fs, 0x2b`, which changes FS to the flat data selector rather than the TEB selector. The gadget separately saves and restores the original FS value. This instruction prevents execution under Pin because Pin stores thread-local state relative to FS. The subsequent FS-relative access faults within Pin's CRT, terminating the process before it reaches the gate. Janus suppresses the FS load and emulates the subsequent window. This transformation preserves the sample's externally visible FS state because the original gadget restores FS before returning.

The 24-instruction entry window implements the ReWolf `wow64ext X64Call` thunk [24]. It maps up to four integer arguments to the x64 calling convention, places additional arguments in a spill array, reserves shadow space, and executes `call qword ptr [rbp+8]` to a caller-supplied 64-bit address. The same gadget can therefore invoke multiple 64-bit functions.

Emulation of the transition alone is insufficient because the call target is supplied dynamically. Janus resolves each target at run time by traversing the 64-bit `PEB→Ldr→`export table through `NtWow64ReadVirtualMemory64`. The resolved address is then mapped to a `module!export` name. Static naming is unreliable because ASLR changes module addresses across runs.

During unpacking, the gadget executes three times for native environment fingerprinting (Table 8). It invokes `RtlGetVersion` once and `NtQuerySystemInformation` twice with information classes 0 and 1. The corresponding `r8` lengths, `0x40` and `0xC`, match the sizes of the x64 structures. The class 1 query reports the native `AMD64` architecture, providing a direct indication of WoW64 execution on an AMD64 host. These calls also bypass hooks or modifications applied only to the 32-bit user-mode API surface. A 32-bit-only DBI therefore cannot observe this reconnaissance sequence, as demonstrated by the failures of Frida, DynamoRIO, and unmodified Pin in Section 4.1.

After the third call, ACRID reaches its original entry point and executes its 32-bit payload. In the isolated run, the payload enters an HTTP(S) command-and-control path implemented with WinINet, WinHTTP, and Winsock. The path supports proxy configuration and TLS certificate validation. Because outbound network access is unavailable, the payload remains in its communication loop. The trace nevertheless includes execution of the unpacked payload rather than only the unpacking routine.

5.3 Additional malware evaluation

To evaluate Janus beyond the ACRID case study, we applied it to samples from the HijackLoader and Lumma families that use Heaven's Gate. Janus detected and emulated the HG transitions in both families and traced the corresponding 64-bit execution. Some samples employed additional techniques unrelated to HG, including DLL stomping, that prevented the unmodified analysis environment from reaching or completing the HG path. In these cases, we first identified the blocking mechanism and then manually neutralized only that mechanism before repeating the experiment. These results demonstrate Janus's ability to trace HG execution once orthogonal anti-analysis mechanisms have been addressed; they do not imply fully automated handling of every evasion technique used by these malware families.

6 Discussion, limitations, and ethics

6.1 Implications of the HG-usage spectrum

The measured spectrum spans more than four orders of magnitude in reachable instruction count across two samples that use the same WoW64 primitive. Themida 3.1.2 uses a 24-instruction window, whereas VMProtect 3.10.5 executes a 427,565-instruction parallel VM. A tool's claim to "support Heaven's Gate" is therefore incomplete unless it specifies the supported range of this spectrum. At the minimal end, a black-box bypass is sufficient because the 32-bit code consumes no result from the 64-bit computation other than the effect of the single syscall issued by each gate. This fixed effect can be replicated without executing the gate. Such a bypass is insufficient at the parallel-VM end because the verification executes in an architecture invisible to the DBI, and the 32-bit code consumes its results (Section 5.1). Cross-architecture instrumentation is therefore required at the upper end of the spectrum.

The ACRID malware sample (Section 5.2) shows that window size alone does not determine the required analysis capability. Its 24-instruction window implements a reusable thunk that invokes caller-selected 64-bit `ntdll` functions, including `RtlGetVersion` and `NtQuerySystemInformation` for native environment fingerprinting. An HG tracer must therefore emulate the window and resolve and execute its 64-bit call targets. This capability is required both for parallel VMs that return state to 32-bit execution and for call primitives such as ACRID's.

6.2 Implications of the longitudinal finding

The two surveys (Sections 4.3 and 4.5) show that Heaven's Gate usage is version-specific. Usage also follows different trajectories within the histories of individual protectors: it is transient in Themida but persistent and increasing in VMProtect. Comparisons of anti-analysis behavior across tools or time must therefore identify exact versions. "Themida" and "VMProtect" without build identifiers do not define reproducible analysis targets. Similarly, behavior observed in one sample cannot be generalized to an entire packer family. The relevant question is not whether packer X uses Heaven's Gate, but which versions use it and at what scale.

6.3 Limitations

Emulator fidelity at the syscall boundary. HGE dispatches syscalls through a trampoline (Section 3.4). The two engines share a coherent memory view only for pages that HGE identity-maps into the process. The kernel validates user pointers against the address space rather than Unicorn's address space. Consequently, a guest pointer allocated in Unicorn but not committed at the same address in the process causes a fault when accessed by a syscall, although emulated instructions can access it successfully. We observed one such instance (Section 5.1.3). In the anti-debug variant, two of four `ProcessDebugObjectHandle` queries returned `STATUS_ACCESS_VIOLATION`, although `VirtualQuery` confirmed that the output buffer was committed. The fault therefore originates from another pointer passed by the query, most likely the optional `ReturnLength` output parameter on the guest stack. This error does not alter control flow in the evaluated sample. The predicate indicates debugger presence only when the query returns `STATUS_SUCCESS` with a non-null handle; `STATUS_ACCESS_VIOLATION` follows the "not debugged" branch. The values that control execution—resolved syscall numbers, `ProcessDebugPort`, timing and `cpuid` results, and VM bookkeeping—remain consistent with native execution. Only the status of a non-gating probe differs, and the sample reaches the OEP, confirming that this status does not control the gate. More generally, an emulator that dispatches native syscalls must provide identity-backed mappings for every user pointer passed to a syscall. Until this limitation is resolved, trampolined status codes should be reported as faithful only when independently corroborated.

Host and OS diversity. Correct execution, OEP reachability, and syscall recovery were reproduced on a second machine. Execution speed (Section 4.2) and the dynamic trace of the sample's environment-scanning anti-debug logic differed between hosts. The process and disk enumeration routines inspect host-specific debugger artifacts, making their instruction sequences host-dependent, although their final verdict remained unchanged. The evaluation did not cover a broad range of Windows builds. However, the analysis produced consistent results on the tested Windows 10 and Windows 11 systems. VMProtect invokes the same `Nt` functions on both systems. Its run-time syscall resolution accounts for the small number of service numbers, including `NtCreateDebugObject`, that differ between builds.

Other Unicorn fidelity gaps. Unicorn has documented edge cases in SIMD and x87/FPU semantics beyond the syscall-pointer limitation. These cases did not affect the evaluated samples. However, samples whose HG windows depend on these execution paths may diverge from native execution. Such divergence can be detected by comparison with a native baseline.

HG-variant coverage. Janus supports the five transition idioms in our taxonomy (Section 2.2). It detects and emulates `jmp_far` and `retf`, which occur in our real-world corpus, as well as `call_far` and `iretd`, which are validated using the synthetic corpus (Section 4.1) but are not observed in the real-world samples. Variants beyond these five idioms remain unsupported. One example is a manually constructed `SS:RSP` reload through `iretq`. Because no real-world sample containing this variant was available, it is excluded from the current coverage claims.

Performance portability. We report end-to-end wall-clock time for the two case studies (Section 4.2). A broader performance evaluation across multiple hosts should additionally measure trace throughput, peak resident set size, and per-transition latency distributions over the complete synthetic corpus and the commercial sample set. Without these measurements, the reported performance results should not be treated as portable across systems.

Evasion handling and malware breadth. Janus was additionally evaluated on HijackLoader and Lumma samples. Some samples required manual neutralization of analysis obstacles unrelated to HG, including DLL stomping, before the HG execution path could be traced. Janus therefore does not provide fully automated handling of all anti-analysis mechanisms used by these families; the experiments isolate its ability to detect and emulate HG after such orthogonal blockers are removed. Evaluation on additional samples and malware families is needed to determine how broadly the observed HG patterns generalize.

6.4 Ethics

The commercial-packer samples were generated from a single benign program using purchased licenses. The malware sample was obtained from the public VirusTotal repository and executed only in a network-isolated VM. The released artifacts do not bypass protector licensing. The implemented techniques are limited to the analysis of protected and malicious binaries.

7 Related work

Heaven's Gate. Practitioners documented Heaven's Gate before its adoption by commercial packers. Ionescu's "Closing Heaven's Gate" [11] provides the canonical account of the CS-selector switch and the corresponding kernel-level countermeasures. Prior dynamic analyses remain limited. Hwang et al. [9] manually reproduce version-specific gate behavior but report only the outcomes, omitting the executed 64-bit instructions. Analyses of malware that uses Heaven's Gate, including TrickBot [15], HijackLoader [32], and Lumma Stealer [26], examine individual cases rather than provide a general tracing method. These studies neither produce instrumented execution traces of the 64-bit window nor quantify the code executed behind the gate across packers.

Hwang et al. also provide the closest prior measurement of Heaven's Gate in VMProtect. Our longitudinal survey (Section 4.5) confirms their principal observation but yields a more precise version history. Their evaluation examines only VMProtect 3.6 and characterizes Heaven's Gate as a version-independent "default protection feature." Our VMProtect 3.6.0 traces contain the same executed `jmp_far` idiom (Figure 2), but the single-version analysis omits two distinctions. First, the scale of Heaven's Gate execution varies by version. The mechanism is absent through version 3.5.0, appears in version 3.5.1 as a window of 319–445 instructions, and expands to a parallel VM of approximately 400k instructions only in version 3.8.1. Version 3.6 therefore belongs to the minimal-window regime rather than the parallel-VM regime. Second, Heaven's Gate is not limited to VMProtect. Themida 3.1.2 uses a runtime-only `retf` variant (Section 4.4) that a `jmp_far`-only breakpoint sweep cannot detect. The black-box bypass proposed by Hwang et al. is sufficient for minimal windows but does not support instruction-level analysis of the parallel VM.

Dynamic binary instrumentation. Pin [17], DynamoRIO [3], and Frida [22] are established user-mode dynamic binary instrumentation frameworks. Each framework instruments one architecture per process and therefore cannot follow execution across a Heaven's Gate transition (Section 2). HGE uses the Unicorn CPU emulator [27], which is derived from QEMU [1]. QEMU-based whole-system analysis platforms, including PANDA [7], DECAF [8], and S2E [4], can observe execution in both modes. However, they do not provide Pin-style selective instrumentation while retaining native execution speed for 32-bit code.

Hardware-assisted debuggers such as HyperDbg [13] use a different execution model. HyperDbg operates as a thin hypervisor based on Intel VT-x and extended page tables. Its position below the processor's privilege modes allows it to observe user- and kernel-mode execution across Heaven's Gate transitions. Recent versions also support programmable instruction-level tracing across the user–kernel boundary. HyperDbg therefore operates at ring −1, outside the user-mode threat model defined in Section 2.5. In contrast, Janus retains native Pin instrumentation for 32-bit code and emulates only the 64-bit window. This design provides cross-mode visibility within a programmable user-mode DBI workflow.

Automated unpacking and emulation-assisted analysis. Pinicorn [16] is the closest system in architectural design because it also combines Pin and Unicorn. Pinicorn targets generic unpacking, whereas Janus targets cross-architecture tracing of Heaven's Gate. Janus extends the shared Pin–Unicorn substrate with detection for five Heaven's Gate variants, cross-mode state marshalling, a real-syscall trampoline, and longitudinal measurements of execution scale. Prior work on x64 unpacking [5], hidden-code extraction [12], and generic unpacking [18] addresses recovery of the original-entry-point image. This objective is orthogonal to cross-mode tracing. We use unpacked images only as verification ground truth (Section 5.1).

Anti-DBI and anti-analysis. Commercial protectors such as VMProtect [28] implement the anti-analysis primitives described in Section 2. Debuggers such as x64dbg [29] and anti-anti-debugging plugins such as ScyllaHide [30] address these primitives from the debugger layer [25]. Prior anti-DBI research catalogs instrumentation-detection techniques and demonstrates that DBI frameworks expose detectable artifacts. Polino et al. [21] measure and counter anti-instrumentation techniques in malware. D'Elia et al. [6] systematize the mechanisms through which DBI frameworks reveal their presence. Our work instead measures the anti-analysis primitives that commercial packers execute behind Heaven's Gate and quantifies their execution scale.

8 Conclusion

Heaven's Gate encompasses execution patterns that vary in both transition form and execution scale. In the evaluated packers, 64-bit execution ranges from a 24-instruction window that issues one debugger-detection syscall in Themida 3.1.2 to a 427,565-instruction virtual machine in VMProtect 3.10.5, a difference of more than four orders of magnitude despite the use of the same WoW64 transition mechanism. Its adoption also varies across versions: Heaven's Gate appears in only one of the 22 evaluated Themida builds. In the real-world corpus, VMProtect uses the `jmp_far` form, whereas Themida and ACRID use the `retf` form with runtime-constructed targets. Janus additionally supports the direct and indirect `call_far` forms and the `iretd` form, which were validated using the synthetic corpus but were not observed in the evaluated real-world samples.

Janus combines Pin and Unicorn to detect all five transition forms and emulate their 64-bit execution windows. Syscalls encountered during emulation are forwarded to the native kernel. This design enables instruction counting, classification of the parallel virtual machine, and recovery of the invoked syscalls. In the ACRID malware sample, Janus identified 64-bit `ntdll` calls executed behind the gate and revealed native fingerprinting that bypasses 32-bit user-mode hooks. A black-box bypass can reproduce the outcome of a minimal execution window but cannot analyze a parallel virtual machine whose verification logic executes outside the architecture visible to the DBI framework. Heaven's Gate implementations that contain such state-producing logic therefore require cross-architecture instrumentation.

References

[1] Fabrice Bellard. 2005. QEMU, a Fast and Portable Dynamic Translator. In *Proceedings of the USENIX Annual Technical Conference (ATC), FREENIX Track*. 41–46.

[2] Sanjay Bhansali, Wen-Ke Chen, Stuart de Jong, Andrew Edwards, Ron Murray, Milenko Drinić, Darek Mihočka, and Joe Chau. 2006. Framework for Instruction-level Tracing and Analysis of Program Executions. In *Proceedings of the 2nd International Conference on Virtual Execution Environments (VEE)*. 154–163. https://doi.org/10.1145/1134760.1220164

[3] Derek L. Bruening. 2004. *Efficient, Transparent, and Comprehensive Runtime Code Manipulation*. Ph. D. Dissertation. Massachusetts Institute of Technology.

[4] Vitaly Chipounov, Volodymyr Kuznetsov, and George Candea. 2011. S2E: A Platform for In-Vivo Multi-Path Analysis of Software Systems. In *Proceedings of the 16th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. 265–278. https://doi.org/10.1145/1950365.1950396

[5] Seokwoo Choi, Taejoo Chang, Changhyun Kim, and Yongsu Park. 2020. x64Unpack: Hybrid Emulation Unpacker for 64-bit Windows Environments and Detailed Analysis Results on VMProtect 3.4. *IEEE Access* 8 (2020), 127939–127953. https://doi.org/10.1109/ACCESS.2020.3008900

[6] Daniele Cono D'Elia, Emilio Coppa, Simone Nicchi, Federico Palmaro, and Lorenzo Cavallaro. 2019. SoK: Using Dynamic Binary Instrumentation for Security (And How You May Get Caught Red Handed). In *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security (AsiaCCS)*. 15–27. https://doi.org/10.1145/3321705.3329819

[7] Brendan Dolan-Gavitt, Josh Hodosh, Patrick Hulin, Tim Leek, and Ryan Whelan. 2015. Repeatable Reverse Engineering with PANDA. In *Proceedings of the 5th Program Protection and Reverse Engineering Workshop (PPREW)*. https://doi.org/10.1145/2843859.2843867

[8] Andrew Henderson, Aravind Prakash, Lok Kwong Yan, Xunchao Hu, Xujiewen Wang, Rundong Zhou, and Heng Yin. 2014. Make It Work, Make It Right, Make It Fast: Building a Platform-Neutral Whole-System Dynamic Binary Analysis Platform. In *Proceedings of the 2014 International Symposium on Software Testing and Analysis (ISSTA)*. 248–258. https://doi.org/10.1145/2610384.2610407

[9] Seon-Jin Hwang, Assem Utaliyeva, Jae-Seok Kim, and Yoon-Ho Choi. 2023. Bypassing Heaven's Gate Technique Using Black-Box Testing. *Sensors* 23, 23 (2023), 9417. https://doi.org/10.3390/s23239417

[10] Intel Corporation. 2026. *Intel 64 and IA-32 Architectures Software Developer's Manual*. Intel Corporation. https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html Vol. 3A, sections on IA-32e compatibility mode and 64-bit mode.

[11] Alex Ionescu. 2015. *Closing "Heaven's Gate"*. Retrieved June 25, 2026 from https://www.alex-ionescu.com/closing-heavens-gate/

[12] Min Gyung Kang, Pongsin Poosankam, and Heng Yin. 2007. Renovo: A Hidden Code Extractor for Packed Executables. In *Proceedings of the 2007 ACM Workshop on Recurring Malcode (WORM)*. 46–53. https://doi.org/10.1145/1314389.1314399

[13] Mohammad Sina Karvandi, MohammadHossein Gholamrezaei, Saleh Khalaj Monfared, Soroush Meghdadizanjani, Behrooz Abbassi, Ali Amini, Reza Mortazavi, Saeid Gorgin, Dara Rahmati, and Michael Schwarz. 2022. HyperDbg: Reinventing Hardware-Assisted Debugging. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 1709–1723. https://doi.org/10.1145/3548606.3560649

[14] Kaspersky GReAT. 2024. *Crimeware report: Acrid, ScarletStealer and Sys01 stealers*. Retrieved June 25, 2026 from https://securelist.com/crimeware-report-stealers/112633/

[15] GoSecure Titan Labs. 2021. *TrickBot Leverages Zoom Work from Home Interview Malspam, Heaven's Gate and… Spamhaus?* Retrieved June 25, 2026 from https://gosecure.ai/blog/2021/12/03/trickbot-leverages-zoom-work-from-home-interview-malspam-heavens-gate-and-spamhaus/

[16] Gwangyeol Lee, Minho Kim, Jeong Hyun Yi, and Haehyun Cho. 2024. Pinicorn: Towards Automated Dynamic Analysis for Unpacking 32-Bit PE Malware. *Electronics* 13, 11 (2024), 2081. https://doi.org/10.3390/electronics13112081

[17] Chi-Keung Luk, Robert Cohn, Robert Muth, Harish Patil, Artur Klauser, Geoff Lowney, Steven Wallace, Vijay Janapa Reddi, and Kim Hazelwood. 2005. Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation. In *Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*. 190–200. https://doi.org/10.1145/1065010.1065034

[18] Lorenzo Martignoni, Mihai Christodorescu, and Somesh Jha. 2007. OmniUnpack: Fast, Generic, and Safe Unpacking of Malware. In *Proceedings of the 23rd Annual Computer Security Applications Conference (ACSAC)*. 431–441. https://doi.org/10.1109/ACSAC.2007.15

[19] Microsoft. 2020. WOW64 Implementation Details. Microsoft Learn. https://learn.microsoft.com/en-us/windows/win32/winprog64/wow64-implementation-details Accessed: 2026-09-03.

[20] Juhyun Park, Yun-Hwan Jang, Soohwa Hong, and Yongsu Park. 2019. Automatic detection and bypassing of anti-debugging techniques for microsoft windows environments. *Advances in Electrical and Computer Engineering* 19, 2 (2019), 23–28. https://doi.org/10.4316/AECE.2019.02003

[21] Mario Polino, Andrea Continella, Sebastiano Mariani, Stefano D'Alessio, Lorenzo Fontana, Fabio Gritti, and Stefano Zanero. 2017. Measuring and Defeating Anti-Instrumentation-Equipped Malware. In *Proceedings of the 14th Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)*. 73–96. https://doi.org/10.1007/978-3-319-60876-1_4

[22] Ole André Vadla Ravnås. 2014. *frida stalker*. Retrieved June 25, 2026 from https://frida.re/

[23] ReWolf. 2011. Mixing x86 with x64 Code. Technical article. https://blog.rewolf.pl/blog/?p=102 Accessed: 2026-09-03.

[24] ReWolf. 2014. *wow64ext: A Library for Calling 64-bit Code from a 32-bit WoW64 Process*. Retrieved June 25, 2026 from https://github.com/rwfpl/rewolf-wow64ext

[25] Sachiel. 2021. *Analysis of "Heaven's Gate" part 1*. Retrieved June 25, 2026 from https://sachiel-archangel.medium.com/analysis-of-heavens-gate-part-1-62cca0ace6f0

[26] SecurityScorecard. 2024. *Revisiting Heaven's Gate with Lumma Stealer*. Retrieved June 25, 2026 from https://securityscorecard.com/wp-content/uploads/2024/01/Whitepaper-Revisiting-Heavens-Gate-With-Lumma-Stealer.pdf

[27] unicorn engine. 2015. *Unicorn CPU Emulator*. Retrieved June 25, 2026 from https://www.unicorn-engine.org

[28] VMPSoft. 2016. *VMProtect*. Retrieved June 25, 2026 from https://vmpsoft.com/

[29] x64dbg. 2016. *x64dbg*. Retrieved June 25, 2026 from https://x64dbg.com/

[30] x64dbg contributors. 2023. *ScyllaHide: Advanced Usermode Anti-Anti-Debugger*. Retrieved June 25, 2026 from https://github.com/x64dbg/ScyllaHide

[31] Lei Xue, Yajin Zhou, Ting Chen, Xiapu Luo, and Guofei Gu. 2017. Malton: Towards On-Device Non-Invasive Mobile Malware Analysis for ART. In *26th USENIX Security Symposium (USENIX Security 17)*. 289–306.

[32] Zscaler ThreatLabz. 2025. *Analyzing New HijackLoader Evasion Tactics*. Retrieved June 25, 2026 from https://www.zscaler.com/blogs/security-research/analyzing-new-hijackloader-evasion-tactics

Generative AI Usage

The authors used Claude to improve the manuscript's language and readability and to refine figures and tables. Claude Code assisted with experiment automation, statistical analysis, and the implementation of analysis and exception-handling code. The research questions, technical claims, experimental design, dataset construction, and selection of citations were determined by the authors. The authors reviewed all AI-assisted text and code and validated the reported results by executing the developed tools and examining the resulting traces and logs.