■ Code Analysis Results

- scanned files: 13
- total findings: 148
- input\00000021.py : 13 findings
- input\api_redirection.py : 7 findings
- input\controlFlow.py : 5 findings
- input\controlflow_flattening.py : 9 findings
- input\deadcode.py : 17 findings
- input\dynamic_loading.py : 6 findings
- input\inlineExpansion.py : 17 findings
- input\instruction_substitution.py : 13 findings
- input\junkcode.py : 18 findings
- input\mixed_language.py : 5 findings
- input\nameIdentifier.py : 9 findings
- input\opaque_predicate.py : 23 findings
- input\stringEncryption.py : 6 findings

===== String Encryption =====

input\stringEncryption.py: obf-line: # Match only well-formed numeric arrays like [65, 66, 67] -> deobf-line: # Match only well-formed numeric arrays like "khi" (XOR-encrypted string detected and decrypted)

===== Identifier Cleaner =====

input\junkcode.py: m1 -> var1 (Obfuscated identifier replaced (m1 -> var1))

input\junkcode.py: m2 -> var2 (Obfuscated identifier replaced (m2 -> var2))

input\junkcode.py: m1 -> var1 (Obfuscated identifier replaced (m1 -> var1))

input\junkcode.py: m2 -> var2 (Obfuscated identifier replaced (m2 -> var2))

input\junkcode.py: m1 -> var1 (Obfuscated identifier replaced (m1 -> var1))

input\junkcode.py: m2 -> var2 (Obfuscated identifier replaced (m2 -> var2))

input\junkcode.py: m1 -> var1 (Obfuscated identifier replaced (m1 -> var1))

input\junkcode.py: m2 -> var2 (Obfuscated identifier replaced (m2 -> var2))

input\junkcode.py: m1 -> var1 (Obfuscated identifier replaced (m1 -> var1))

input\junkcode.py: m2 -> var2 (Obfuscated identifier replaced (m2 -> var2))

input\junkcode.py: m1 -> var1 (Obfuscated identifier replaced (m1 -> var1))

input\junkcode.py: m2 -> var2 (Obfuscated identifier replaced (m2 -> var2))

===== Control Flow =====

-> (Unreachable branch (condition always False))

- replace `if True: body` with `body` (in-place)

- record lines removed

-> (Unreachable branch (condition always False))

-> print("Always runs")

-> print("Run")

input\deadcode.py: if(false) { ... }` blocks -> (Unreachable branch (condition always false))

input\deadcode.py: if(false) { ... } or if(false) statement; -> (Unreachable branch (condition always false))

input\deadcode.py: if(false) removed (dead code)"}) -> (Unreachable branch (condition always false))

input\deadcode.py: if(true) { ... }` with block contents (strip braces) -> { ... }` with block contents (strip braces) (Always-true condition simplified (kept statement/block, removed condition header))

input\deadcode.py: if(true) { block } => replace with block contents -> { block } => replace with block contents (Always-true condition simplified (kept statement/block, removed condition header))

input\deadcode.py: if(true) inlined (kept body)"}) -> inlined (kept body)"}) (Always-true condition simplified (kept statement/block, removed condition header))

===== Dead Code =====

def _wrapper -> import base64

- tiny wrappers that call another function and do nothing else -> """

- tiny wrappers that call another function and do nothing else

- multiple-level wrappers

" -> import re

Cons -> """

-> """# Dead Code Test Cases

- Python: eval, exec, compile, importlib, getattr -> """

- Python: eval, exec, compile, importlib, getattr/locals tricks

- C/C++: functi (Full cleaned file (Python deadcode removal))

2. Constant addition insid -> """# Inline Expansion Complex Test Cases with Loops

- x - (-1) -> x + 1

- x << 1 -> x * 2

- x + x -> """

- x - (-1) -> x + 1

- x << 1 -> x * 2

- x + x -> 2 * x (or x * 2)

- bitwise trick (Full cleaned file (Python deadcode removal))

Detect/remove junk code: NOPs, identity operations, redundant arithmetic, dead stores used only for obfuscation -> """

- inline assembly or ASM blocks inside C/C++

- pre -> """

- inline assembly or ASM blocks inside C/C++

- presence of JNI-like bridging cod (Full cleaned file (Python deadcode removal))

-> import re

if (2 + 2 == 4) and (3 > -> """for i in range(2):

return ''.join(chr(c ^ -> import re

===== Inline Expansion =====

def _wrapp -> import base64

- tiny wrappers that call another function and do nothing else

- multiple-level wrappers

Cl -> """

- tiny wrappers that call another function and do nothing else

- multiple-level wrappers

Detect and clean fake/unreachable conditions from so -> import re

input\controlflow_flattening.py: '// Suggested deobfuscated sequence\n' + '/* case1 */\n' -> '// Suggested deobfuscated sequence\n/* case1 */\n' (Constant folded)

Conservative detection + optional simple u -> """

-> """# Dead Code Test Cases

- Python: eval, exec, compile, importlib, getattr/locals tricks

- C/C++: functi -> """

- Python: eval, exec, compile, importlib, getattr/locals tricks

- C/C++: functi (Full cleaned file (Python inline folding))

2. Constant addition insid -> """# Inline Expansion Complex Test Cases with Loops

- x - (-1) -> x + 1

- x << 1 -> x * 2

- x + x -> 2 * x (or x * 2)

- bitwise trick -> """

- x - (-1) -> x + 1

- x << 1 -> x * 2

- x + x -> 2 * x (or x * 2)

- bitwise trick (Full cleaned file (Python inline folding))

- 2'

Conservative cleaning: -> """

- inline assembly or ASM blocks inside C/C++

- presence of JNI-like bridging cod -> """

- inline assembly or ASM blocks inside C/C++

- presence of JNI-like bridging cod (Full cleaned file (Python inline folding))

LANG_KEYWORDS = {'python': set(keyword.kwlist) | set(dir(builtins)), 'c': {'auto', 'brea -> import re

f -> """for i in range(2):

return ''.join((chr(c ^ -> import re

===== Opaque Predicates =====

def _wrapp -> import base64
- tiny wrappers that call another function and do nothing else
- multiple-level wrappers
Cl -> """
- tiny wrappers that call another function and do nothing else
- multiple-level wrappers
Detect and clean fake/unreachable conditions from so -> import re
Conservative detection + optional simple u -> """
-> """# Dead Code Test Cases
- Python: eval, exec, compile, importlib, getattr/locals tricks
- C/C++: functi -> """
- Python: eval, exec, compile, importlib, getattr/locals tricks
- C/C++: functi (Full cleaned file (Python opaque predicate simplification))
2. Constant addition insid -> """# Inline Expansion Complex Test Cases with Loops
- x - (-1) -> x + 1
- x << 1 -> x * 2
- x + x -> 2 * x (or x * 2)
- bitwise trick -> """
- x - (-1) -> x + 1
- x << 1 -> x * 2
- x + x -> 2 * x (or x * 2)
- bitwise trick (Full cleaned file (Python opaque predicate simplification))
Conservative cleaning: -> """
- inline assembly or ASM blocks inside C/C++
- presence of JNI-like bridging cod -> """
- inline assembly or ASM blocks inside C/C++
- presence of JNI-like bridging cod (Full cleaned file (Python opaque predicate simplification))
LANG_KEYWORDS = {'python': set(keyword.kwlist) | set(dir(builtins)), 'c': {'auto', 'brea -> import re
f -> """for i in range(2):
return ''.join((chr(c ^ -> import re

===== Control Flow Flattening =====

input\00000021.py: -> (Detected Python flattened control-flow patterns. Manual reconstruction recommended.)

input\controlflow_flattening.py: -> (Detected Python flattened control-flow patterns. Manual reconstruction recommended.)

input\inlineExpansion.py: -> (Detected Python flattened control-flow patterns. Manual reconstruction recommended.)

===== Instruction Substitution =====

input\instruction_substitution.py: x - (-1) -> x + 1 (Canonicalized: neg-neg to plus)

input\instruction_substitution.py: x + x -> 2 * x (Canonicalized: x+x to 2*x)

input\instruction_substitution.py: x+x -> 2 * x (Canonicalized: x+x to 2*x)

input\instruction_substitution.py: x+x -> 2 * x (Canonicalized: x+x to 2*x)

- x - (-1) -> x + 1
- x << 1 -> x * 2
- x + x -> 2 * x (or x * 2)
- bitwise trick -> """
- x + 1 -> x + 1
- x << 1 -> x * 2
- 2 * x -> 2 * x (or x * 2)
- bitwise tricks ((Applied instruction substitution canonicalization (Python))

===== Dynamic Code Loading =====

input\00000021.py: -> (Detected dynamic constructs (eval/exec/compile/reflection). Manual review required: [{'type': 'dynamic_py', 'lineno': 81, 'snippet': 'getattr(', 'reason': 'dynamic execution/reflection'}])

input\controlFlow.py: -> (Detected dynamic constructs (eval/exec/compile/reflection). Manual review required: [{'type': 'dynamic_py', 'lineno': 14, 'snippet': 'compile(', 'reason': 'dynamic execution/reflection'}, {'type': 'dynamic_py', 'lineno': 14, 'snippet': 'compile(', 'reason': 'dynamic execution/reflection'}, {'type': 'dynamic_py', 'lineno': 14, 'snippet': 'compile(', 'reason': 'dynamic execution/reflection'}, {'type': 'dynamic_py', 'lineno': 15, 'snippet': 'compile(', 'reason': 'dynamic execution/reflection'}, {'type': 'dynamic_py', 'lineno': 15, 'snippet': 'compile(', 'reason': 'dynamic execution/reflection'}, {'type': 'dynamic_py', 'lineno': 15, 'snippet': 'compile(', 'reason': 'dynamic execution/reflection'}])

input\controlflow_flattening.py: -> (Detected dynamic constructs (eval/exec/compile/reflection). Manual review required: [{'type': 'dynamic_py', 'lineno': 11, 'snippet': 'compile(', 'reason': 'dynamic execution/reflection'}])

input\deadcode.py: -> (Detected dynamic constructs (eval/exec/compile/reflection). Manual review required: [{'type': 'dynamic_py', 'lineno': 228, 'snippet': 'compile(', 'reason': 'dynamic execution/reflection'}, {'type': 'dynamic_py', 'lineno': 237, 'snippet': 'compile(', 'reason': 'dynamic execution/reflection'}, {'type': 'dynamic_py', 'lineno': 252, 'snippet': 'compile(', 'reason': 'dynamic execution/reflection'}, {'type': 'dynamic_py', 'lineno': 129, 'snippet': 'getattr(', 'reason': 'dynamic execution/reflection'}, {'type': 'dynamic_py', 'lineno': 144, 'snippet': 'getattr(', 'reason': 'dynamic execution/reflection'}])

input\dynamic_loading.py: -> (Detected dynamic constructs (eval/exec/compile/reflection). Manual review required: [{'type': 'dynamic_py', 'lineno': 11, 'snippet': 'compile(', 'reason': 'dynamic execution/reflection'}, {'type': 'dynamic_py', 'lineno': 11, 'snippet': 'compile(', 'reason': 'dynamic execution/reflection'}, {'type': 'dynamic_py', 'lineno': 11, 'snippet': 'compile(', 'reason': 'dynamic execution/reflection'}, {'type': 'dynamic_py', 'lineno': 11, 'snippet': 'compile(', 'reason': 'dynamic execution/reflection'}, {'type': 'dynamic_py', 'lineno': 11, 'snippet': 'compile(', 'reason': 'dynamic execution/reflection'}, {'type': 'dynamic_py', 'lineno': 11, 'snippet': 'compile(', 'reason': 'dynamic execution/reflection'}])

input\inlineExpansion.py: -> (Detected dynamic constructs (eval/exec/compile/reflection). Manual review required: [{'type': 'dynamic_py', 'lineno': 153, 'snippet': 'compile(', 'reason': 'dynamic execution/reflection'}])

input\instruction_substitution.py: -> (Detected dynamic constructs (eval/exec/compile/reflection). Manual review required: [{'type': 'dynamic_py', 'lineno': 11, 'snippet': 'compile(', 'reason': 'dynamic execution/reflection'}, {'type': 'dynamic_py', 'lineno': 11, 'snippet': 'compile(', 'reason': 'dynamic execution/reflection'}, {'type': 'dynamic_py', 'lineno': 11, 'snippet': 'compile(', 'reason': 'dynamic execution/reflection'}, {'type': 'dynamic_py', 'lineno': 11, 'snippet': 'compile(', 'reason': 'dynamic execution/reflection'}, {'type': 'dynamic_py', 'lineno': 32, 'snippet': 'compile(', 'reason': 'dynamic execution/reflection'}, {'type': 'dynamic_py', 'lineno': 32, 'snippet': 'compile(', 'reason': 'dynamic execution/reflection'}])

input\opaque_predicate.py: -> (Detected dynamic constructs (eval/exec/compile/reflection). Manual review required: [{'type': 'dynamic_py', 'lineno': 198, 'snippet': 'eval(', 'reason': 'dynamic execution/reflection'}, {'type': 'dynamic_py', 'lineno': 192, 'snippet': 'compile(', 'reason': 'dynamic execution/reflection'}, {'type': 'dynamic_py', 'lineno': 235, 'snippet': 'compile(', 'reason': 'dynamic execution/reflection'}])

input\stringEncryption.py: -> (Detected dynamic constructs (eval/exec/compile/reflection). Manual review required: [{'type': 'dynamic_py', 'lineno': 18, 'snippet': 'compile(', 'reason': 'dynamic execution/reflection'}])

===== Junk Code =====

-> """# Dead Code Test Cases

Conservative cleaning: -> """

f -> """for i in range(2):

===== API Redirection =====

===== Mixed Language Obfuscation =====

No cases detected.

===== Identifier Obfuscation =====