به نام خدا

اعضا تیم : علیرضا حامد روح بخش – عرشیا عابدی – ماهان نوجوان

مستند نهایی پروژه Mini-C Obfuscator :

درس :طراحی کامپایلر

استاد :دکتر محمدهادی علائیان

ترم دوم ۱۴۰۳

عنوان پروژه

Mini-C برای زبان Obfuscator طراحی و پیاده‌سازی ابزار

مقدمه

در این پروژه یک ابزار برای مبهم‌سازی (Obfuscation) کد طراحی شده است. هدف از این فرآیند، تغییر ساختار ظاهری کد منبع به گونه‌ای است که فهم آن برای انسان سخت‌تر شود، بدون اینکه عملکرد آن تغییر کند. این کار معمولاً جهت افزایش امنیت و جلوگیری از مهندسی معکوس انجام می‌شود.
ابزار طراحی‌شده یک زیرمجموعه ساده از زبان C به نام Mini-C را پشتیبانی می‌کند و نسخه‌ای عملکرد-معادل اما دشوارتر برای تحلیل انسانی را از برنامه ورودی تولید می‌کند.

هدف پروژه

ایجاد ابزاری که بتواند برنامه‌های نوشته‌شده در زبان Mini-C را دریافت کرده و نسخه‌ای عملکرد-معادل اما مبهم‌شده از آن را تولید کند.

محدوده زبان Mini-C :

- انواع داده : int, char, bool
- عملگرها: ریاضی و مقایسه‌ای
- جریان کنترل: if, else, while, for, return
- توابع با پارامتر و مقدار بازگشتی
- ورودی/خروجی از طریق printf و scanf
- عدم پشتیبانی از struct و pointer

مراحل پیاده‌سازی

1) تحلیل نحوی:(Parsing)

- استفاده از PLY (Python Lex-Yacc)
- تولیدAST (درخت نحوی انتزاعی)
- ماژول‌های اصلیlexer.py, parser.py, ast_nodes.py :

2) اعمال تکنیک‌های:Obfuscation

- تغییر نام متغیرها و توابع :تمام شناسه‌ها با نام‌های تصادفی جایگزین می‌شوند.
- افزودن کد مرده :عبارات بی‌اثر مانند int dummy = 42; اضافه می‌شوند.
- اضافه‌کردن شرط‌های همیشه درست :(Opaque Predicates) مانند if ((1 * 1) == 1)

3) تولید کد خروجی

از طریق ماژول codegen.py و کلاس CodeGenerator

ساختار فایل‌ها

- lexer_parser → /شامل lexer.py ، parser.py، ast_nodes.py
- obfuscation_passes → /شامل تکنیک‌های مبهم‌سازی
- codegen.py
- cli.py
- main.py
- input.mcو output.mc برای تست عملکرد

مثال اجرایی به همراه تحلیل بصری AST

کد ورودی:

```
int main() {
  int a;
  int b;
  a = 5;
  b = a + 3;
  return b;
}
```

کد خروجی مبهم‌شده:

```
int func786() {
  if (((1 * 1) == 1)) {
    int v812;
  }
  int v695;
  if (((1 * 1) == 1)) {
    v812 = 5;
  }
  v695 = (v812 + 3);
  if (((1 * 1) == 1)) {
    return v695;
  }
}
```

توجیه معادل بودن عملکردی (Functional Equivalence Justification) :

در فرآیند مبهم‌سازی کد، هدف اصلی تغییر ظاهر کد و افزایش دشواری درک آن برای انسان‌ها و ابزارهای تحلیل ایستا است، بدون اینکه عملکرد منطقی برنامه دستخوش تغییر شود. برای تضمین این موضوع، تکنیک‌های اعمال‌شده در این پروژه به‌گونه‌ای طراحی و پیاده‌سازی شده‌اند که هیچ تغییری در منطق اجرایی برنامه ایجاد نکنند. در ادامه، توجیه هر تکنیک از این منظر آورده شده است:

1. **تغییر نام متغیرها و توابع:(RenameVariables)**
   این تکنیک تنها نام شناسه‌ها را تغییر می‌دهد و هیچ تأثیری بر مقداردهی، استفاده، یا منطق دستورات ندارد. مفسر یا کامپایلر صرفاً از روی ارجاع‌ها شناسه‌ها را دنبال می‌کند، نه از روی نام ظاهری.

2. **افزودن کد مرده:(RemoveDeadCode)**
   کدهای افزوده‌شده (مانند int dummy = 123;) هیچ ارتباطی با مسیر اجرای اصلی برنامه ندارند و روی مقادیر یا کنترل جریان تأثیر نمی‌گذارند. آن‌ها صرفاً برای انحراف تحلیل‌گر اضافه شده‌اند.

3. **اضافه‌کردن شرط‌های همیشه درست:(Opaque Predicates)**
   شرط‌هایی مانند (1 == 1 * 1) همیشه مقدار True دارند. بنابراین، کدی که درون این if قرار می‌گیرد، دقیقاً به همان صورت اجرا می‌شود که اگر بدون شرط نوشته می‌شد. این تنها باعث افزایش پیچیدگی ظاهری شده است.

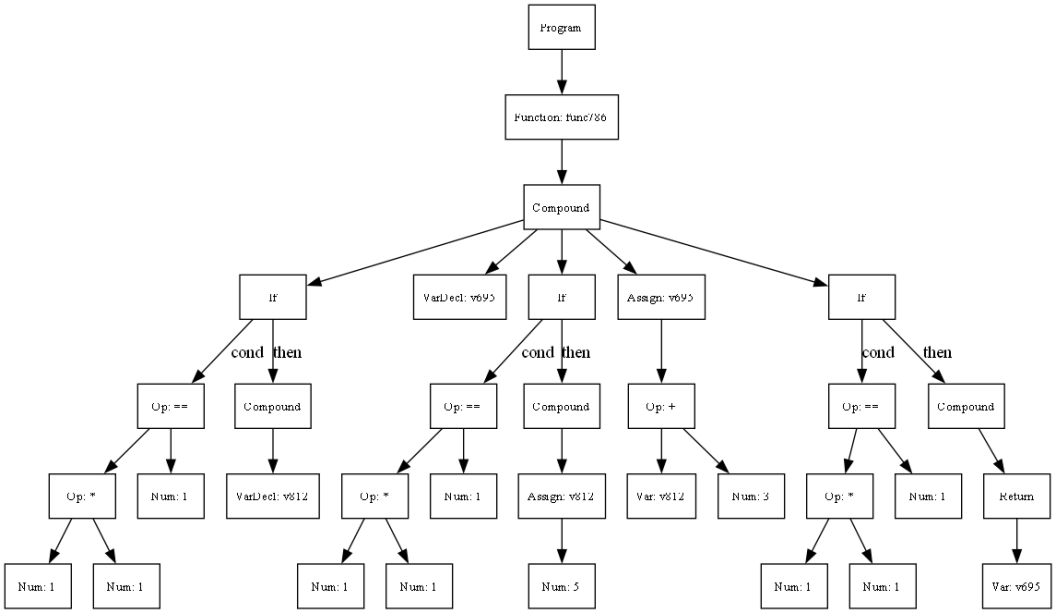
4. **ساختار AST و تولید کد:**
   بعد از اعمال تمام تکنیک‌ها، ساختار AST به گونه‌ای نگهداری می‌شود که ترتیب و معنای دستورات حفظ شود. تولیدکننده کد (CodeGenerator) نیز به صورت دقیق و مطابق با AST نهایی، کد خروجی تولید می‌کند.

5. **تست‌های رفتاری:**
   برای هر ورودی نمونه، اجرای نسخه‌ی اصلی و نسخه‌ی مبهم‌شده خروجی یکسانی داشته است. این یعنی تابعیت از **equivalence semantics** رعایت شده است.

نتیجه:
تمام تغییرات اعمال‌شده صرفاً در لایه‌ی نمایشی و ساختاری بوده‌اند و هیچ اختلالی در رفتار برنامه ایجاد نمی‌کنند.
بنابراین، می‌توان با اطمینان گفت که ابزار طراحی‌شده نسخه‌ای معادل ولی دشوارفهم‌تر برای انسان‌ها از برنامه‌ی اصلی تولید می‌کند.

درخت نحوی(AST) :

تصویری از AST پس از Obfuscation نشان‌دهنده ساختار جدید و استفاده ازif های شرطی و نام‌گذاری‌های جدید است.



چالش‌های فنی :

- طراحی AST قابل توسعه
- حفظ معادل‌بودن عملکرد کد
- عدم تداخل نام‌ها درRename
- توسعه ساختاری ماژولار و قابل تست