گزارش نهایی پروژه(فاز 1) :ابزار مبهم‌ساز برای زبان CMINI

اعضای گروه: محمد **صادق حیدری(شماره دانشجویی:40117683)**

مقدمه

هدف از این پروژه، طراحی و پیاده‌سازی یک **ابزار مبهم‌ساز** برای زبان ساده‌شده‌ای از زبان برنامه‌نویسی سی به نامCMINI بوده است. این ابزار با بهره‌گیری از تکنیک‌های مختلف، ساختار ظاهری کد را به گونه‌ای تغییر می‌دهد که درک آن برای افراد دشوارتر شود، در حالی که **رفتار اجرایی برنامه بدون تغییر باقی می‌ماند**. این کار باعث دشوارتر شدن تحلیل معکوس (وارونه‌سازی برنامه) می‌شود.

روش‌های به‌کاررفته در مبهم‌سازی

در این پروژه سه روش اصلی برای انجام عملیات مبهم‌سازی پیاده‌سازی شده‌اند که هرکدام در ادامه معرفی می‌شوند:

۱. تغییر نام متغیرها و توابع

در این روش، نام تمامی متغیرها و توابع به جز تابع اصلی برنامه (که معمولاً با نام «main» شناخته می‌شود) با نام‌هایی تصادفی و بی‌معنا جایگزین می‌گردند.

دلیل عدم تأثیر بر عملکرد:

از آنجا که این تغییر فقط در نام‌گذاری ظاهری انجام می‌شود و مرجع‌دهی به متغیرها و توابع در سطح درونی زبان تغییری نمی‌کند، **عملکرد برنامه کاملاً حفظ می‌شود**.

۲. درج کد بی‌اثر (کد مرده)

در این روش، خطوطی از کد که تأثیری بر اجرای اصلی برنامه ندارند، به صورت تصادفی در بخش‌هایی از کد (مانند درون بلوک‌ها) اضافه می‌شوند. این کدها معمولاً شامل تعریف متغیرهایی هستند که هیچ‌گاه استفاده نمی‌شوند.

دلیل عدم تأثیر بر عملکرد:

چون این کدها هرگز به کار گرفته نمی‌شوند و در جریان اجرایی برنامه نقشی ندارند، تأثیری بر نتیجه‌ی برنامه نخواهند داشت. در برخی موارد حتی ممکن است توسط بهینه‌ساز کامپایلر نادیده گرفته شوند.

۳. پیچیده‌سازی عبارات ریاضی

در این روش، عبارات ساده ریاضی مانند جمع و تفریق به شکل‌های معادل اما پیچیده‌تر بازنویسی می‌شوند تا درک آن‌ها برای خواننده سخت‌تر شود.

دلیل عدم تأثیر بر عملکرد:

عبارات بازنویسی‌شده از نظر ریاضی **کاملاً با نسخه‌ی اصلی برابر** هستند و فقط ساختار نوشتاری آن‌ها متفاوت است.

چالش‌های فنی در روند پیاده‌سازی

در فرآیند توسعه‌ی این ابزار، چالش‌هایی فنی وجود داشت که برخی از مهم‌ترین آن‌ها به شرح زیر است:

1. **کار با ابزار (ANTLR)** : درک کامل ساختار درخت نحوی و نحوه‌ی دسترسی به توکن‌ها و جایگزینی آن‌ها نیاز به تمرین و تحلیل دقیق داشت.
2. **جایگزینی ایمن توکن‌ها:** برای جلوگیری از به‌هم‌ریختگی ساختار نحوی کد، باید فقط بخش‌های مجاز تغییر داده می‌شدند. مدیریت دقیق توالی توکن‌ها کاری حساس و زمان‌بر بود.
3. **درج کد بی‌اثر در محل مناسب:** کد مرده باید فقط در بلوک‌هایی درج می‌شد که ساختار نحوی مناسبی داشتند تا باعث خطای نحوی در کامپایل نشوند.
4. **اطمینان از حفظ عملکرد کد:** پس از هرگونه مبهم‌سازی، لازم بود برنامه‌ی حاصل کامپایل و اجرا شود و خروجی آن با نسخه‌ی اصلی مقایسه گردد تا صحت عملکرد آن تأیید شود.

گزارش نهایی پروژه(فاز 2) :ابزار غیر مبهم‌ساز برای زبان CMINI

توضیح تکنیک‌ها

در این فاز هدف، بازگرداندن کد مبهم‌شده‌ی Mini-C به یک نسخه‌ی ساده، تمیز و قابل‌فهم است، بدون اینکه تغییری در رفتار اجرایی برنامه رخ دهد. برای این منظور تکنیک‌های زیر پیاده‌سازی شدند:

1. **ساده‌سازی عبارات (Expression Simplification)**
   - حذف الگوهای پیچیده و غیرضروری مانند:

```
a + b)) → a + b)-*1-)
x - (-y) → x + y
```

2. **حذف کد مرده (Dead Code Elimination)**

- شناسایی متغیرها و دستورات بی‌استفاده مانند:
  `int unused = 42;`

این کدها هیچ نقشی در خروجی برنامه ندارند و صرفاً برای مبهم‌سازی اضافه شده‌اند.

3. **ساده‌سازی جریان کنترل (Control Flow Simplification)**
   - بازنویسی ساختارهای غیرضروری به ساختار ساده‌تر.

4. **بازگرداندن نام‌ها (Identifier Renaming)**

   - جایگزینی شناسه‌های بی‌معنی با نام‌های معنادار.

   مثال: `obf_result → total`، `var1 → x`، `fxz → sum`

مقایسه خط به خط کد

به عنوان نمونه، یک تابع جمع مبهم‌شده:

```
Copy code

int sum(int a, int b) {
    return a + b;
}
```

پس از de-obfuscation:

```
Copy code

int fxz(int x1, int x2) {
    int a39 = x1 - (-x2);  // کد پیچیده‌شده
    int unused = 1234;     // کد مرده
    return a39;
}
```

تغییرات خط به خط:

- خط دوم: عبارت `(2x1 - (-x)` ساده شد به `a + b`
- خط سوم: متغیر `unused` حذف شد
- نام تابع `fxz` به `sum` تغییر کرد
- پارامترها `2x1, x` به `a, b` تغییر کردند

همچنین در `main`، ساختار `while + switch` که اجرای خطی را شبیه‌سازی می‌کرد، به دستورات مستقیم فراخوانی تابع و چاپ خروجی ساده شد.

چالش‌های حذف ابهام

1. **تشخیص الگوهای پیچیده:**
گاهی عبارات در چندین سطح تو در تو سادسازی شده بودند (مثلاً `(((x - (-y)) -*1-)`). استخراج و سادسازی بازگشتی این موارد نیازمند طراحی الگوریتم دقیق بود.
2. **شناسایی کد مرده:**
تفکیک بین متغیرهای واقعاً بی‌استفاده و متغیرهایی که فقط کم‌استفاده‌اند، یکی از چالش‌ها بود. برای حل این مسئله شمارش تعداد دفعات استفاده از شناسه‌ها در بدنه‌ی تابع انجام شد.
3. **بازگرداندن نام‌های معنادار:**
چون در مرحله‌ی obfuscation نام‌ها به صورت تصادفی تغییر یافته بودند، بازگردانی صددرصد دقیق ممکن نبود. بنابراین با **حدس مبتنی بر context** (مثلاً وجود عملگر جمع در return → احتمال زیاد تابع sum است) نام‌های معنادار انتخاب شدند.
4. **حفظ صحت اجرایی:**
در تمامی مراحل باید دقت می‌شد که تغییرات تنها روی بخش‌های غیرضروری اعمال شوند و منطق اصلی برنامه تغییر نکند. برای اطمینان، هر دو نسخه‌ی مبهم و ساده اجرا و خروجی آن‌ها مقایسه شد.

جمع‌بندی

ابزار طراحی‌شده در این پروژه توانسته است سه روش مختلف مبهم‌سازی را به‌خوبی بر روی زبان سی‌مینی پیاده‌سازی کند. با وجود تغییرات گسترده در ظاهر کد، **رفتار نهایی برنامه کاملاً بدون تغییر باقی می‌ماند.** این ابزار می‌تواند به عنوان پایه‌ای برای توسعه ابزارهای پیشرفته‌تر مبهم‌سازی یا به‌کارگیری در پروژه‌های امنیتی مورد استفاده قرار گیرد.