L-003

C2Rust を使用した C 言語プログラムのセキュア化の評価
Evaluation of C language program security using C2Rust

三浦 向平† 八槇 博史‡
Kohei Miura Hirofumi Yamaki

1. はじめに

C2Rust は、C 言語プログラムを Rust 言語で記述されたプログラムに自動変換するツールである。Rust 言語はメモリ安全性と実行速度の両立を目指した言語であるが、C2Rust で自動変換され生成されたプログラムは、メモリ安全性が保証されない Unsafe Rust によって記述されており、メモリ安全性を高める場合は手動で修正する必要がある。本論文では、メモリ脆弱性を持つ C 言語プログラムを作成し、それを C2Rust で変換したプログラムと、元のプログラムを参考に新たに作成した Rust 言語プログラムの動作・比較を行い、C2Rust の評価をメモリ安全性や実行速度、データ容量などの観点から行った。

2. 背景

2.1 C2Rust

C2Rust[1]は、既存の C 言語プログラムを Rust 言語で記述されたものに自動的に変換できるツールである。Ubuntu をはじめとした Linux や macOS などにインストールして利用できるほか、デモンストレーションを行う Web サイトも存在している[2]。C2Rust で変換した後の Rust 言語のコードは、メモリ安全でない Unsafe Rust で記述されており、コードをメモリ安全にするためにはさらにリファクタリングを行う必要がある。

2.2 先行研究

千脇らは、C 言語のライブラリを Rust 言語で書かれたものに置き換えることで、既存のプログラムのメモリ安全性を高める研究を行っている[3]。研究報告の結論では、Rust 言語内でメモリ操作が完結する場合は、メモリ脆弱性を防ぐことが可能になることや、C 言語と Rust 言語で情報をやり取りする際に C 言語から渡される情報が間違っていると、メモリ脆弱性による誤動作が発生してしまうこと、C 言語と比較して Rust 言語は大きなオーバーヘッドは発生しなかったことなどが示されている。

3. 評価手法

本研究で実際に評価を行った環境を以下に示す。

- ホスト OS：Windows10 ver.21H2
- CPU：Intel Core i7-7500U 2.70 GHz
- ホストメモリ：8.00 GB
- 仮想環境ソフト：VirtualBox 6.1.34
- ゲスト OS：Ubuntu 20.04 LTS
- ゲストメモリ：4 GB
- C 言語コンパイラ：gcc 9.4.0
- Rust 言語コンパイラ：rustc 1.60.0
- C2Rust バージョン：0.15.1

†東京電機大学 システムデザイン工学研究科
Tokyo Denki University Graduate School of System Design and Technology
‡東京電機大学 システムデザイン工学部
Tokyo Denki University, School of System Design and Technology

3.1 評価観点

はじめにメモリ脆弱性を持つ C 言語プログラムを作成し、C2Rust を使用して Rust 言語で記述されたものに自動変換した。さらに C 言語プログラムを参考に、Unsafe Rust を使用せずに同様の動作を行う Rust 言語プログラムを新たに作成した。これらの 3 種類のプログラムをそれぞれコンパイルし、生成された実行ファイルを実際に動作させ、メモリ脆弱性による動作が発生していないか確認を行う。また、それぞれの実行ファイルの実行速度およびデータ容量の比較も行う。

3.2 評価対象

本研究で C 言語プログラムの作成時に利用した、メモリ脆弱性を以下に示す。

- バッファオーバーフロー
- 配列の境界外アクセス
- ヒープ領域の二重解放
- ダングリングポインタ
- 関数 printf の書式指定文字列の脆弱性

4. 評価結果

3 種類のプログラムの評価結果を、実行時のメモリ脆弱性の発生の有無、データ容量、実行速度の観点からに分けて、以下に示す。

4.1 メモリ脆弱性の発生の有無

C 言語、C2Rust、Rust 言語のプログラムの実行結果を、メモリ脆弱性の種類ごとに分けて示す。

4.1.1 バッファオーバーフロー

C 言語プログラムでは、過度に大きい入力の値を与えた場合、指定したメモリ領域からあふれて書き込まれてほかの変数を上書きし、プログラムが停止した。C2Rust のプログラムでも同様の結果となった。Rust 言語のプログラムでは、入力された値に応じて動的にメモリが確保されるため、オーバーフローが発生しなかった。

4.1.2 配列の境界外アクセス

C 言語のプログラムでは、配列の境界外のアドレスへアクセスすることが可能だった。その際、配列外では実行ごとに格納されている値が変わっていた。C2Rust のプログラムでは、配列の境界外にアクセスした場合パニックが発生し、プログラムが停止した。Rust 言語の場合も同様に、実行途中でプログラムが停止した。

4.1.3 ヒープ領域の二重解放

C 言語および C2Rust のプログラムでは、一度関数 free でヒープ領域の解放が行われた後、2 回目の解放を行うタイ

ミングでプログラムがクラッシュし、停止した。Rust 言語のプログラムでは、ヒープ領域に確保したメモリは変数のスコープを抜ける際に自動的に解放されるため、二重解放は発生しなかった。

4.1.4 ダングリングポインタ

C 言語プログラムおよび C2Rust のプログラムでは、一度解放済みのポインタでも、代入や格納されている値の表示を行うことができるように、アクセス可能な状態であった。Rust 言語のプログラムでは、上述の二重解放の際と同様に、ヒープ領域の解放は自動的に行われるため、メモリ解放を明記する必要はない。

4.1.5 書式指定文字列の脆弱性

3 種類のプログラムは、標準入力からの入力を画面に表示するものである。C 言語のプログラムでは printf を利用しているが、printf の引数には fgets で入力を格納した変数のみを指定した。それぞれの実行時に echo コマンドを使用し、標準入力には表示させる文字列と書式指定子"%x"を指定した。C 言語と C2Rust のプログラムでは書式指定子が認識され、スタックの内容および文字列の 16 進数での値を表示した。Rust 言語のプログラムでは書式指定子がそのまま文字列として表示され、スタックの内容を確認することはできなかった。

4.2 データ容量

3 種類の実行ファイルのサイズを計測し、脆弱性ごとに以下の表 1 に示す。

表 1　実行ファイルのデータ容量の比較

	C 言語	C2Rust	Rust 言語
バッファオーバーフロー	17.0 kB	2.8 MB	3.8 MB
境界外アクセス	16.8 kB	2.6 MB	3.7 MB
二重解放	16.7 kB	2.6 MB	3.7 MB
ダングリングポインタ	16.8 kB	2.6 MB	3.7 MB
書式指定文字列	16.8 kB	2.6 MB	3.7 MB

どのプログラムも、C 言語のサイズが最も小さく、次いで C2Rust が大きく、Rust 言語のプログラムのサイズが最も大きかった。

4.3 実行速度

1000 回繰り返してプログラムの実行を行うシェルスクリプトを作成し、C 言語、C2Rust、Rust 言語プログラムの実行ファイルを対象にそれぞれ実行した。ただし二重解放に関しては、C 言語および C2Rust のプログラムで実行中止の 1000 回繰り返しに非常に時間がかかるため、計測および比較は行わなかった。これらを除いて実行速度を 3 度計測し、平均をとったうえで脆弱性の種類ごとにグラフにまとめた結果を、右の図 1 に示す。

結果として、4 種類のプログラム全てで C 言語が最も速く、C2Rust と Rust 言語では Rust 言語がわずかに上回った。

5. 考察

メモリ安全でない Rust の機能を使用するには Unsafe キーワードを使用して、指定したブロック内に限定して操作を

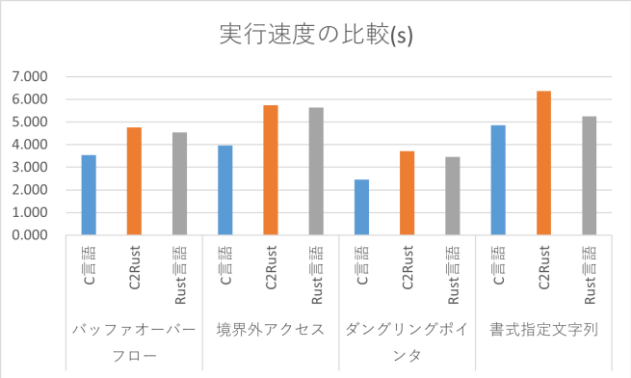

図 1　実行ファイルの実行速度の比較

行う必要がある。そのため、プログラムのメモリ安全性を高めるためには、プログラム内の安全な部分と危険な部分を詳細に分割する必要がある。本研究で C 言語プログラムを C2Rust で変換したものは、C 言語での main 関数の全体が Unsafe ブロックに格納されており、プログラム全体が Unsafe 状態であった。そのため C2Rust で変換したプログラムは、Unsafe な動作を行う部分のみを Unsafe ブロックに格納することを含め、適切な書き換えを行うことで、メモリ安全性を高めることができると考えられる。

本研究での計測では、C 言語プログラムと比較して、C2Rust での自動変換プログラムと Rust 言語プログラムの実行ファイルのデータ容量は大きかったが、これらのプログラムは最適化せずにコンパイルを行った。よって、様々な最適化を行うことによって C2Rust 及び Rust 言語プログラムの実行ファイルの容量を減少させ、さらに実行速度を上げることができる可能性がある。

6. まとめ

本稿では、メモリ脆弱性を持つ C 言語プログラムを作成し、それを C2Rust で変換したプログラムと、元のプログラムを参考に作成した、メモリ安全な Rust 言語プログラムの動作・比較を行い、C2Rust の評価をメモリ安全性や実行速度、データ容量などの観点から行った。C2Rust で変換した状態のプログラムではメモリ安全性を完全に保証されないが、Rust 言語が備えるメモリ安全性を部分的に活用することができた。また、C 言語のプログラムと比較して、実行ファイルのサイズの増大と、実行速度の低下が確認された。

C 言語に起因するメモリ脆弱性の多くは、C2Rust での自動変換のみでは改善されなかったが、配列外へのアクセスなど一部のメモリ脆弱性の発生を防ぐことができる。そのため、C 言語プログラムに対して C2Rust を使用することは、メモリ脆弱性を利用した攻撃への対策として一定の効果があると考えられる。

参考文献

[1] GitHub - immunant/c2rust: Migrate C code to Rust 。https://github.com/immunant/c2rust （参照 2022-06-13）
[2] C2Rust Demonstration。https://c2rust.com （参照 2022-06-13）
[3] 千脇 貴之, 橋本 正樹, “メモリ破損脆弱性を利用した攻撃に対する Rust を用いた対策手法”, 情報処理学会研究報告, Vol.2021-CSEC-92, No.9 (2021).