安全なシステム記述言語および高信頼 OS 記述言語

代表者　東京大学大学院情報理工学系研究科コンピュータ科学専攻 米澤 明憲
協力企業 日立製作所, とめ研究所, エーエヌラボ
http://www.yl.is.s.u-tokyo.ac.jp/e-society

1. 安全・高信頼な基盤ソフトウェアの開発技術

高度情報化に伴い，コンピュータウィルスや不正アクセス，情報漏洩などの問題が，社会の安全性を脅かす深刻な事態となりつつある．今後，さらなる情報化は不可避であり，社会基盤としてのコンピュータシステムの安全性を保証することは急務である．

すでに我々の周りには数多くの応用ソフトウェアが稼動しているが，それらの多くは，ソフトウェアの安全性の問題が社会的に顕在化する以前に開発されたプログラミング言語や OS などの基盤的ソフトウェアに依存している．このような深刻な問題に対処するためには，対症療法的な解決ではなく，理論的な方法・系統的な方法に基づいて，安全性や信頼性が保証されなければならない．

本プロジェクトでは，型理論をはじめとするプログラムの解析技術やプログラム検証・証明技術の開発によって，既存の基盤ソフトウェアの安全性・信頼性を強化する技術を開発しその有効性を示した．

2. プロジェクトの目標・手法・成果とその検証

2.1. 目標

本プロジェクトの具体的な研究目標は次の二つである．

1) 安全な C 言語システムの構築

C 言語は，ソフトウェアの安全性が社会問題化する以前，今から三十年以上前に開発されたプログラミング言語であり，安全性に問題があることが知られている．しかし，言語機能の柔軟性が高いため, 未だ多くの基盤的ソフトウェアが, この C 言語を用いて実装されている．この問題に対処するため，我々は 2 種類の安全な C 言語のコンパイラを構築した．

2) 安全／高信頼な OS を構築するための記述システムの開発

OS (オペレーティングシステム) はコンピュータシステムで最も基礎的なソフトウェアであり，この OS に問題があると，コンピュータシステム全体の安全性･信頼性に大きく影響する．これに対し我々は，安全／高信頼な OS を構築するため，型理論に基づいたシステムソフトウェア記述システムの開発を目指した．

なお，本プロジェクトで実現を目指した安全性／高信頼性とは具体的には次の三つである．

1) メモリ安全性

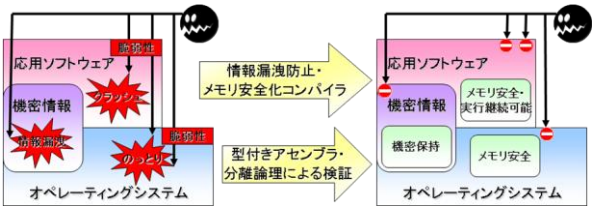
プログラムが不正なメモリアクセスを行わないこと．

2) 機密情報の漏洩防止

プログラムが機密情報を (例えその一部でも) 外部に漏洩しないこと．

3) 実行継続可能性

不正なメモリアクセスを検出した後でも，機密情報が外部に漏洩することなく安全に実行を継続できること．



2.2. 目標達成のための手法

本プロジェクトでは以下の四つの手法について研究を行った．

1) 安全なコードを生成する C 言語コンパイラの開発

2) C 言語ソースコード上の情報流の系統的解析手法の考案

3) OS 記述用アセンブリ言語の型システム／型検査の考案

4) OS の安全性の証明手法／検証手法の考案

上記四手法の概要については，第 3 節以降で述べる．

2.3. 研究開発の主な成果とその検証

1) メモリ安全なコードを生成する ANSI 標準 C 言語のコンパイラの構築と公開[1]

検証： OpenSSL (ネットワーク通信の暗号化や通信相手の認証等を行うプログラム, ソースコード約 300,000 行), BIND9 (インターネット上での名前解決に広く用いられているサーバ, ソースコード約 350,000 行), などの基盤的システムをコンパイル・実行した．

2) メモリ安全で機密情報漏洩を防止できるコードを生成する C 言語（＋付加記述）のコンパイラ VITC の構築

検証：thttpd (小規模ウェブサーバ, 約 1 万行) をコンパイル・実行した．

3) メモリ管理やスレッド管理など, 今まで不可能とされていた OS 機能の型付き言語による記述に成功

検証： 新しい型付きアセンブリ言語 TALK を考案し，それにより上記機能を実現した．

4) 高次論理系に基づく証明支援系 Coq の枠組で「分離論理」体

[1] この開発は 平成１６年度末まで米澤研究室で実施され，その後の主たる開発は, 産業技術総合研究所で継続され公開された. (http://www.rcis.aist.go.jp/project/FailSafeC-ja.html)

系[2]を定式化し，ヒープメモリやポインタへの操作の安全性を証明する体系を構築した．

検証： 教育目的で実用されている Topsy OS のメモリ管理モジュールの安全性を，Coq 証明支援系を用いて証明・検証し，Topsy 内のバグも発見した．

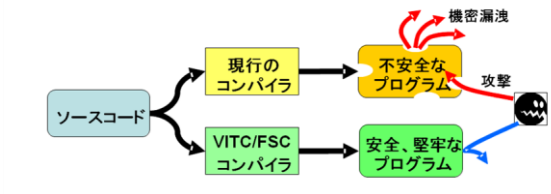
3. 高安全な C 言語コンパイラの開発

C 言語は，その実行速度やハードウェアメモリモデルについての記述力から，Java などの新言語登場後の現在もシステムプログラムの記述では最も一般的であり，多くのアプリケーションにも広く使われている．

しかし，これらの C 言語で書かれたプログラムが一般的なバグにより誤動作したり，メモリ管理のバグによってセキュリティーホールが出現し，攻撃を受けることでシステムが破壊されたり機密情報を盗まれることが多発している．これは，C言語がメモリ保護機構を欠くためや言語自体にはそもそも情報の機密性という概念がないことに起因する．

3.1. メモリ安全C言語システム(FSC)の開発

メモリ安全性を保証し，コンピュータウィルスによるメモリ脆弱性攻撃を検知，緊急停止するコードを生成する C 言語コンパイラを設計し実装した[3]．我々の研究成果によって得られた型システムに基づくアルゴリズムを用いて，ANSI 規格を包含する C 言語プログラムを機械語コードにコンパイルする段階で，安全性を検査するコードを自動的に挿入するコンパイラを構築した．



VITC/メモリ安全 C コンパイラ

3.2. ＶＩＴＣ言語システムの開発

C 言語プログラムにおけるメモリ安全性を保証することに加えて，機密情報の漏洩を防止できるコードを生成する VITC 言語コンパイラを構築した．ここでは，プログラム内の機密情報の流れを追う，いわゆる情報流解析を行う機構をコンパイラに導入している．この情報流解析では，変数の持つ値の機密性の高さを「型」と見なすことによって，新しい型システムを考案し，この型システムによってプログラムを解析している．

また，情報流解析により機密情報を漏洩する危険がなくなるため，VITC でコンパイルされたプログラムは，従来の攻撃にさらされても安全に実行を継続することができる．

型による情報流解析の研究は，Java や ML など，理論的に厳正な静的型システムを与えることができる言語系においてもっぱら行われ，情報流もほぼコンパイル時に，すなわちプログラム実行前に解析することができた．それに対し，C 言語の型システムは非常に柔軟ではあるが正確性を欠く．我々の VITC の研究では，静的情報流解析のみでは C 言語の表現力を保ちつつ機密漏洩を防ぐことは難しいため，実行時の情報流の動的検査を導入している．また，情報の機密度を表す型は，オリジナルな C 言語には備わっていないため，アノテーションの付記が必要だが，記述量は必要最小のものとなっている．

4. 安全／高信頼な OS 構築用記述システムの開発

4.1 OS 機能の型付きアセンブリ言語による記述

近年の型理論研究の進歩により，多くのアプリケーションプログラムが「強く型付けされたプログラミング言語」 (例: Java, C#) を用いて既に作成されるようになっている．これは，強く型付けされた言語で記述されたプログラムは，実行時に予期せぬメモリエラーを生じないことが保証されるためである．

ところが，コンピュータを動作させる上で最も基礎的で重要なプログラムである OS は，未だに強く型付けされていない言語を用いて作成され続けている．このため，従来 OS の安全性を保証・検証することは非常に困難であり，実際，安全性が証明された OS は (機能が限定された非常に小さな OS を除けば) 未だ存在していない．

そこで本研究では，OS カーネルの重要要素 (メモリ管理機構やスレッド管理機構など) の記述ができる，強く型付けされた安全なアセンブリ言語を設計・実装した．これで，バッファオーバーフロー等のセキュリティ脆弱性の原因が OS に存在しないことを保証する方法を与えることに成功した．

4.2 ヒープメモリ・ポインタ操作の安全性証明の体系の構築.

広く利用されている既存の OS は，既に C 言語で構築されてしまっているが，これを一から安全な言語で構築し直すことは常に現実的であるとは限らない．このため，既存の OS の安全性を検証することは重要である．これに対し本研究は，分離論理 [2] にもとづいて，ヒープメモリ・ポインタ操作の安全性を数理論的に証明する体系を構築し，これを用いて実際の OS カーネル (Topsy) のメモリ管理システムの正当性の検証を行った．

具体的には，定理証明支援系 Coq を用いて分離論理の体系を定式化し，ヒープメモリ・ポインタ操作の安全性を検証するための体系を構築した．これを用いて Topsy のメモリ管理システムのソースコードをこの定式化の枠組内で検証したところ，Topsy のメモリ管理システムには，メモリリーク，すなわち本来ならば再利用できるはずのメモリを再利用せず，結果としてシステムのメモリ全てを消費してしまうという問題が生じる可能性があることなど，幾つかの問題があることを発見できた．

---

[2]互いに交差しない（分離された）メモリ領域上で使われるプログラム変数やポインタの性質を自然に表現できる論理体系．

[3] 前出の脚注１を参照．