# プログラムの静的解析に基づく侵入検知システム

## Intrusion Detection System Based on Static Analyses

岡 瑞起
Mizuki OKA
筑波大学大学院修士課程理工学研究科
Master's Program in Science and Enginnering, University of Tsukuba
mizuki@osss.is.tsukuba.ac.jp

阿部 洋丈
Hirotake Abe
筑波大学大学院博士課程工学研究科
Doctoral program in Enginnering, University of Tsukuba
habe@osss.is.tsukuba.ac.jp

大山 恵弘
Yoshihiro Oyama
東京大学大学院情報理工学系研究科
Graduate School of Information Science and Technology, University of Tokyo
oyama@yl.is.s.u-tokyo.ac.jp

Emil Meng
Emil Meng
Bachelor of Science program in Computer Science, University of Colorado
Bachelor of Science program in Computer Science, University of Colorado
menge@colorado.edu

加藤 和彦
Kazuhiko Kato
筑波大学電子・情報工学系/科学技術振興事業団
Institute od Information Sciences and Electronics, University of Tsukuba / Japan Science and Technology Corporation
kato@is.tsukuba.ac.jp

**Summary**

プログラムの脆弱性を悪用してプログラムが辿る制御フローをプログラマの意図に反するものに変え，プログラムに異常な動作をさせる攻撃が深刻な問題となっている．我々はその攻撃による被害を防ぐ手段として，プログラムが辿る制御フローの異常を検知する侵入検知システムを開発した．そのシステムは，プログラムが辿る制御フローがバイナリコードが規定する制御フローの規則に従っていることを検査しながらプログラムを実行する．検査はシステムコール呼び出し時のスタックを調べることによって行われる．スタック内の制御情報の移り変わりが，バイナリコードが規定する制御フローの規則に従っていないとき，異常が発生したと判断する．我々のシステムはスタックの情報を利用するので，システムコールの情報だけを利用する既存の手法よりも制御の場所を高い精度で把握できる．それはオーバヘッドの縮小と検出精度の向上をもたらす．

加えて、我々のシステムはプログラムが辿る全ての制御フローを静的に列挙することにより、検査時のオーバヘッドを削減する。

我々は実験で異常検知を行うことによるオーバヘッドを測定した．また、脆弱性のあるプログラムに実際の攻撃をしかけ検知できるかどうかの実験を行なった。

Attacking and exploiting vulnerable programs by manipulating control flow is becoming a common occurrence, and thus a severe problem in today's computing world. We have implemented an intrusion detection system that detects anomalous behavior of a program. The system ensures that the behavior of a program never strays from its intended control flow, which is extracted from its binary code.

Our system utilizes the call stack's information to detect anomalous behavior. Call stack information is useful to determine the current execution state, and therefore our system can achieve higher precision and lower overhead when contrasted to other intrusion detection systems.

Furthermore, our system statically obtains all the transition paths of possible control flow. This technique further reduces overhead during runtime. We will display our results by measuring the overhead via experiments with real attack code on vulnerable programs.