김일성종합대학학보 주체107(2018)년
정보과학 제64권 제4호

원천코드준위에서의 조종흐름혼란에 의한 응용프로그람보안의 한가지 방법

박철준, 권세흔, 리옥

프로그람에 대한 보안은 쏘프트웨어제품개발에서 필수적인 공정으로서 프로그람기술이 발전함에 따라 그것에 대한 요구수준도 높아지고있다.

선행연구[1]에서는 허위코드삽입에 의한 Android응용프로그람보안방법을 론의하였으며 선행연구[2]에서는 아쎔블리코드에 대해 코드혼란을 실현하기 위한 여러가지 수법들을, 선행연구[3]에서는 코드혼란방법에 대한 모형화를 제기하였다.

그러나 우의 모든 방법들은 실행화일준위에서의 적용을 전제로 하고있으며 원천코드에 대해서는 아무런 처리도 진행하지 않는다.

Java에 대해 Proguard를 비롯하여 원천코드준위에서의 혼란도구들이 리용되고있으나 이름혼란을 기본으로 하고있다.

론문에서는 선행연구[1]의 방법을 개선하여 원천코드준위에서 코드혼란을 실현하기 위한 한가지 방법에 대하여 론의하였다.

1. 원천코드준위에서의 코드혼란방법

코드혼란은 프로그람을 의미적으로 동등(같은 결과를 출력하는)하나 해석하기 어려운 다른 프로그람으로 변환하는 수법이며 원천코드준위에서의 코드혼란은 원천코드를 그것과 동등하나 해석하기 어려운 다른 원천코드로 변환하는 방법이다.

이때 실행화일을 대상으로 하는 경우에 비해 다음과 같은 우점을 가진다.

첫째로, 실행화일에 대한 복잡한 해석과정이 필요없으며 임의의 프로그람언어에 대해서도 적용가능하다.

둘째로, 프로그람의 중요한 부분들에 대해서는 혼란깊이를 마음대로 조절하여 보다 높은 강도의 보안을 실현할수 있다.

원천코드준위에서 적용할수 있는 코드혼란방법에는 이름혼란, 문자렬혼란, 자원암호화, 조종흐름혼란 등 여러가지가 있지만 그중에서 가장 중요한것은 조종흐름혼란이다.

조종흐름혼란은 프로그람의 기능성을 보존하면서 실행흐름을 복잡하게 만드는 과정이다. 조종흐름혼란은 함수내에서의 혼란과 함수호출관계의 혼란으로 이루어진다.

함수내에서의 혼란방법에 대해서는 선행연구[1, 2]들에서 론의하였으므로 여기서는 함수호출관계의 혼란방법만을 론의한다.

실행흐름의 견지에서 하나의 프로그람은 함수들과 그것들사이 호출관계의 조합으로 정의할수 있다.

$$P = (F, C)$$

여기서 F는 $F = \{f_1, f_2, \cdots, f_n\}$으로 표시되는 함수목록이고 C는 $C = \{(f_i, f_j)|f_i$로부터 f_j를 호출$\}$로 표시되는 호출관계목록이다.

실행흐름의 복잡도는 함수들에서 다른 함수들을 호출하는 회수 즉 함수당 평균함수 호출개수에 의해 평가할수 있다.

$$CR(P) = |C| / |F|(|C|: \text{ 함수호출의 총회수}, |F|: \text{ 함수개수})$$

우리의 목적은 P를 기능적으로 동등한 P'로 변환하되

$$CR(P') > CR(P)$$

가 되도록 하는것이다.

선행한 방법들에서는 조종흐름혼란을 적용할 때 모든 함수들을 동일하게 취급하므로 혼란깊이가 함수마다 같게 되며 전반적인 혼란깊이를 크게 할수 없었다. 우리는 함수들에 대한 혼란깊이를 함수의 중요도에 따라 변화시키도록 하였다.

또한 선행한 방법들에서는 아무리 혼란을 적용하여도 결과적인 목적함수는 하나의 함수로 넘겨진다.

우리는 중요한 함수들에 대해서는 여러개의 허위코드를 복사하여 해석자가 목적함수의 내용에 대한 일정한 정보를 알고있다고 하여도 정확한 함수를 찾기 어렵게 하였다.

원천준위에서 코드혼란을 실현하기 위한 알고리듬은 다음과 같다.

우선 실행흐름조종변수목록

$$V = \{v_1, v_2, .\cdots, v_m\}$$

을 준비하고 그것의 초기값을 할당한다.

다음 매개 함수 f_i에 대하여 다음의 과정을 수행한다.

① 함수의 중요도 $I(f_i)$에 따라 혼란깊이 q_i와 중복도 d_i를 결정한다.

$$q_i = Q(I(f_i)),\ d_i = D(I(f_i))$$

② d개의 복제함수 $f_{ij}^{'}$들을 창조한다.

$$f_{ij}^{'} = Dup(f_{i,j})\ (j = 1,\ \cdots,\ d)$$

여기서 Dup는 입력된 함수와 파라메터에 따라 원함수의 변종들을 생성하는 알고리듬이다.

③ 새로운 분기함수 Sf_{ik}들을 생성한다.($k = 1, 2,\ \cdots, K_i, K_i \geq q_i d_i$)

④ f_i를 호출하는 함수 $f \in \{f_j \mid (f_j, f_i) \in C\}$들에 대하여 함수 f_i에 대한 호출을 Sf_{ix}(x=1, 2, $\cdots, K_i$)의 호출로 치환한다.

⑤ 분기함수 Sf_{ix}들에서 $V_i(V_i \in V)$의 값에 따라 다른 분기함수 Sf_{ix}들에로의 호출관계를 설정한다. 이때 호출한 함수에 대한 역호출과 자기자체에 대한 재귀적호출도 가능하게 한다.

⑥ 우의 과정을 혼란깊이 q_i만큼 반복한다.

알고리듬을 적용하기 전에 프로그람개발자는 함수의 중요도를 결정하는 척도 $I(f_i)$와 중요함수의 허위복제를 생성하는 알고리듬 Dup를 준비하여야 한다.

알고리듬의 실현에서는 다음의 요구를 지켜야 한다.

① 조종변수목록 V의 초기값에 따라 프로그람의 실행흐름이 달라져야 한다.

② V의 어떤 꼭 하나의 초기값에 대해서만 결과프로그람의 실행흐름이 원천프로그람과 같아야 한다.

우의 알고리듬에 의한 프로그람의 실행흐름변화과정을 그림에 보여주었다.

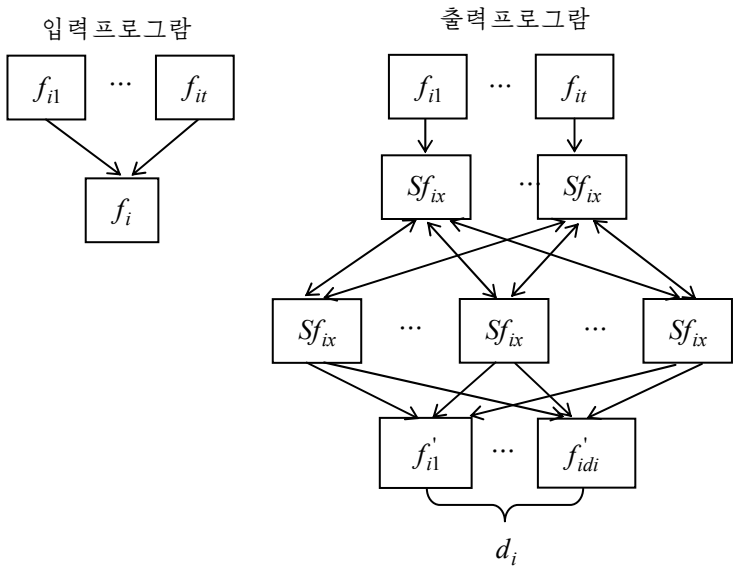

그림. 알고리듬에 의한 프로그람의 실행흐름변화과정

그림에서 보는것처럼 목적함수 f_i에 대한 호출은 q_i단계의 Sf_{ix}에 대한 호출로 변환되며 제일 마감에 복제된 함수 f'_{ij}들을 호출하게 된다.

분기함수 Sf_{ix}들은 조종변수들의 값에 따라 서로 호출하거나 재귀적호출을 진행할수 있다.

모든 함수들에 대하여 동일한 파라메터 q와 d를 리용하는 경우 그림에서와 같이 t개의 함수호출은 $t+q\cdot d$회의 함수호출로 변환되므로 호출관계의 복잡도증가비는 리론적으로 다음과 같이 된다.

$$CR(P')/CR(P)=(t+q\cdot d)/t=1+q\cdot d/t$$

2. 실험 및 결과분석

제안된 방법의 효과성을 검증하기 위하여 C++와 Java로 작성된 검사용프로그람들을 선행한 방법[1, 2]과 제안된 방법에 의해 처리하고 그것을 해석전문가가 주어진 시간내에 해석하도록 하는 실험을 진행하였다.(표) 이때 해석자에게는 프로그람에 대한 일정한 정보를 미리 제시하였다.

표. 해석실험결과(%)

보안방법	C++	Java
선행연구[1]의 방법		22.5
선행연구[2]의 방법	36.3	
제안한 방법	31.2	19.2
제안한 방법 + 실행화일준위의 혼란방법	20.4	9.7

표에서 보는바와 같이 제안한 방법이 선행방법들에 비하여 우월하다는것을 알수 있

다. 이 결과는 해석자가 프로그람에 대한 일정한 정보를 알고있다는 전제하에서 얻어진것이다. 또한 제안된 방법은 Java, C#과 같은 중간코드방식의 언어들에서 더 효과적이며 원천코드준위에서 제안된 방법을 적용하고 실행화일준위에서 다시 다른 보안방법을 적용하면 보안강도를 훨씬 높일수 있다는것을 보여준다.

참 고 문 헌

[1] **김일성**종합대학학보(자연과학), 61, 10, 28, 주체104(2015).

[2] Chandan Kumar Behera et al.; Procedia Computer Science, 70, 757, 2015.

[3] E. E. Ogheneovo et al.; International Journal of Engineering Science Invention, 3, 1, 2014.

주체107(2018)년 8월 5일 원고접수

A Method of Application Protection by Control Flow Obfuscation on Source Code Level

Pak Chol Jun, *Kwon Se Hun* and *Ri Ok*

The application protection is an essential process of software development, and the control flow obfuscation is a protection method of confusing running branches of program. In this paper, we introduced a method of application protection by applying control flow obfuscation on source code level.

Key words: control flow obfuscation, code protection