더미 조건문을 사용한 동적 제어흐름 난독화 기법 개발*

이지원*, 진홍주*, 김선권**, 김기중*, 이동훈*

*고려대학교 정보보호학과

*고려대학교 정보보안학과

A New Dynamic Control Flow Obfuscation Using Dummy Condition

Jiwon Lee*, Hongjoo Jin*, Seon Kwon Kim**, Kijoong Kim*, Dong Hoon Lee*

*Graduate School of Information Security, Korea University

**Graduate School of Cyber Security, Korea University

요 약

코드 난독화는 역공학에 기반한 악의적인 소프트웨어 분석으로부터 프로그램의 지적재산권을 보호하기 위한 수단으로 사용된다. 그중 제어흐름 난독화는 프로그램의 제어흐름을 복잡하게 변환시켜 프로그램을 분석하기 어렵게 만든다. 그러나 일반적으로 제어흐름 난독화된 프로그램의 제어흐름은 정적으로 결정되기에 동적 분석에 취약하다. 이를 해결하기 위해 동적 불분명 술어를 이용한 기법이 제안되었다. 하지만 해당 기법은 처음 나오는 불분명 술어를 통해서 제어흐름이 결정되면 이어지는 제어흐름이 고정되는 한계점이 있다. 본 논문에서는 이러한 한계점을 해결하기 위해 새로운 동적 제어흐름 난독화 기법을 제안한다. 제안하는 기법의 역난독화를 위해서는 추가적인 역난독화 기술이 필요하며 기존 제어흐름 난독화 기법과 쉽게 결합하여 사용할 수 있다.

I. 서론

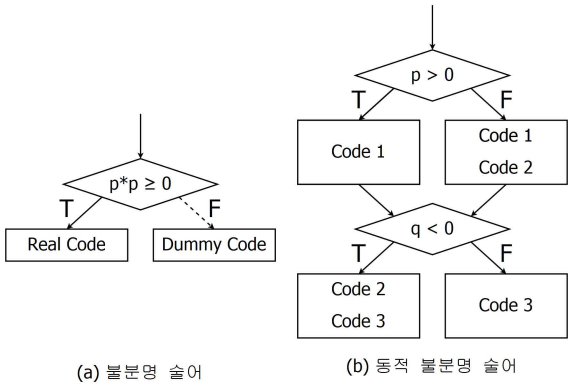
소프트웨어 역공학에 기반한 악의적인 프로그램 분석은 소프트웨어에 대한 지적재산권을 위협한다. 이러한 위협은 소프트웨어 무단 사용 및 복제 그리고 해킹툴 개발 등으로 이어질 수 있다. 프로그램의 분석을 어렵게하는 코드 난독화는 악의적인 역공학으로부터 소프트웨어를 보호하기 위한 해결책으로 사용된다.

일반적으로 코드 난독화란 프로그램의 기능성은 유지하면서 프로그램을 이해하기 어렵게 만드는 변환 프로세스이다. Colleberg 등은 코드 난독화 기술을 난독화 특성에 따라 데이터 난독화, 레이아웃 난독화, 제어흐름 난독화, 방지 난독화로 분류한다[1].

일반적으로 제어흐름 난독화는 프로그램의 불필요한 제어흐름을 만들어 프로그램의 실제 제어흐름을 숨기는 기술이다. 현재까지 프로그램에 제어문(조건문, 선택문, 반복문) 추가[1, 2], 불분명 술어 추가[3], 원본 코드를 여러 코드 블록으로 분할[4], 원본 코드를 여러 함수로 정의[5]하는 등 다양한 제어흐름 난독화 기법들이 제안되었다. 그러나 해당 기법을 포함한 많은 제어흐름 난독화 기법은 프로그램의 제어흐름이 정적으로 결정되기 때문에 동적 분석에는 취약하다[6-8]. 이러한 한계점을 극복하기 위해 동적 불분명 술어를 이용하여 프로그램의 제어흐름을 동적으로 결정하는 제어흐름 난독화 기법이 제안되었다[9, 10]. 그러나 해당 기법은 처음 나오는 불분명 술어를 통해서 제어흐름이 결정되면 이어지는 제어흐름이 고정되는 한계점이 있다.

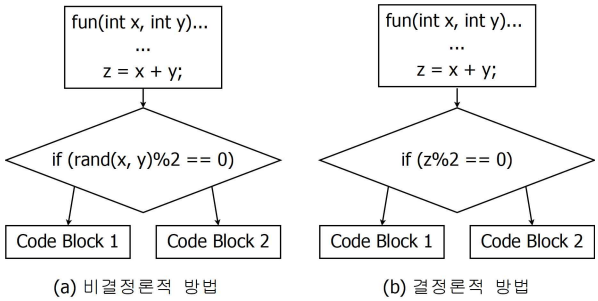
본 논문에서는 더미 조건문과 데이터 난독화를 이용하여 프로그램의 제어흐름을 파악하기 어렵하는 새로운 동적 제어흐름 난독화 기법을 제안한다. 제안하는 기법은 더미 조건문을 사용

* 본 연구는 고려대 암호기술 특화연구센터(UD210027XD)를 통한 방위사업청과 국방과학연구소의 연구비 지원으로 수행되었습니다.



(a) 불분명 술어 (b) 동적 불분명 술어

Fig. 1. 불분명 술어



(a) 비결정론적 방법 (b) 결정론적 방법

Fig. 2. 동적으로 결정되는 더미 조건문

하여 동적 불분명 술어가 가진 제어흐름이 고정되는 한계점을 해결하고 데이터 난독화를 이용하여 공격자에게 역난독화에 필요한 요구사항을 추가한다. 공격자가 동적 제어흐름 난독화에 대한 역난독화를 수행하기 위해서는 데이터 역난독화 기술이 필요하며 반복적으로 코드 블록을 병합해가며 역난독화를 수행해야 한다. 또한 불분명 술어를 이용한 난독화 기법과 결합하면 더 강력한 난독화 기능을 제공한다.

II. 배경지식

2.1 불분명 술어(Opaque Predicate)

불문명 술어는 조건문의 결과가 항상 참 또는 거짓인 논리문이다[3, 9, 10]. 일반적으로 불분명 술어는 Fig. 1의 (a)처럼 더미 코드와 함께 사용되어 새로운 제어흐름 구조를 생성한다. 생성된 제어흐름 구조는 프로그램 실행마다 항상 고정된 제어흐름을 가진다[8].

동적 불분명 술어는 프로그램 실행에 따라 조건문의 결과가 달라지는 불분명 술어이다[9, 10]. 동적 불분명 술어는 Fig. 1의 (b)처럼 2개의 이상의 불분명 술어가 세트가 되어 새로운 제어흐름 구조를 생성한다. 이때 조건문의 결과는 프로그램 실행 중에 결정되므로 정적분석을 통해 제어흐름을 파악하기 어렵다. 그러나 사용되는 불분명 술어들은 처음 조건문 결과가 정해지면 나머지 조건문도 정해진다. 예를 들어 Fig. 1의 (b)에서 처음 나오는 불분명 술어 $p > 0$이 참(T)이라면 뒤에 나오는 불분명 술어 $q < 0$는 반드시 (T)가 된다.

2.2 더미 조건문

더미 조건문은 조건문의 결과가 프로그램의 기능성과는 상관없는 조건문을 말한다. 더미 조건문은 조건문의 결과가 정적으로 결정되는 조건문과 조건문의 결과가 동적으로 결정되는 조건문으로 구성된다. 조건문의 결과가 동적으로 결정되는 조건문은 정적분석으로 제어흐름을 파악하기 어려우므로 본 논문에서 제안하는 기법에는 동적으로 결정되는 조건문을 사용한다.

동적으로 결정되는 더미 조건문은 Fig. 2처럼 비결정론적 방법과 결정론적 방법을 이용해 생성할 수 있다. 비결정론적 방법은 입력된 값이나 내부 연산 결과에 상관없이 조건문의 결과가 매번 다르게 도출하도록 생성한다. Fig. 2 (a)는 랜덤한 값을 이용하여 생성한 비결정론적 조건문을 보여준다. 조건문의 결과는 매개변수 x, y의 값이나 $x+y$의 연산 결과에 영향을 받지 않는다. 결정론적 방법은 입력된 값 혹은 내부 연산 결과에 의존하여 조건문의 결과가 동일한 조건에서는 동일하게 도출되도록 생성한다. Fig. 2 (b)는 지역변수 z를 이용하여 생성한 결정론적 조건문을 보여준다. 조건문의 결과는 매개변수 x, y의 값에 영향을 받아 동일한 x, y에 대해 동일한 결과가 도출된다.

III.본론

본 장에서는 일반화된 동적 제어흐름 난독화 기법에 대해 자세히 설명한다. 먼저 본 논문에서 제안하는 난독화 기법에 사용되는 더미 조건문의 개념을 설명한다. 그런 다음 일반화된 동적 제어흐름 난독화 기법을 설명한다.

3.1 동적 제어흐름 난독화

먼저 동적 제어흐름 난독화 과정을 설명하기 전 그림의 기호가 가진 의미는 다음과 같다.

1. 숫자는 하나의 명령어(코드)를 의미한다. 숫자 뒤에 프라임(′)이 붙으면 데이터 난독화 기법으로 난독화된 명령어를 의미한다. 동일한 숫자 뒤에 붙은 프라임 개수가 다른 것은 동일한 명령어에 상이한 데이터 난독화를 적용한 것을 의미한다.
2. 직사각형은 명령어(코드)들이 모인 코드 블록을 의미한다.
3. 마름모는 더미 조건문을 나타낸다.
4. 화살표는 제어흐름의 이동을 나타낸다. 블록과 마름모 사이의 화살표는 제어흐름의 조건부 이동을 의미한다.

Fig. 3은 제어문이 없는 직선코드 블록에 대한 일반화된 동적 제어흐름 난독화 기법을 보여준다. 먼저 하나의 더미 조건문을 이용하여 직선코드 블록의 복사본을 만든다. 그 후 임의의 위치에 대해 2개 이상의 코드 블록을 분할한다. 분할된 코드 블록 사이에 더미 조건문을 추가해 프로그램의 제어흐름을 변경한다. 이때 추가되는 더미 조건문은 분할된 코드 블록마다 다르거나 일부는 동일하게 추가될 수 있다. Fig. 3의 (c)는 코드 블록을 분할한 후 더미 조건문을 추가하는 과정을 1번 수행한 것이고 Fig. 3의 (d)는 해당 과정을 여러 번 수행한 것을 보여준다. 마지막으로 각 코드 블록에 대해 코드 난독화를 수행한다. 여기서 같은 명령어들로 이루어진 코드 블록은 서로 다른 데이터 난독화 기법을 적용하여 서로 다른 기능성을 가진 코드 블록으로 보이게 한다.

3.2 동적 제어흐름 난독화에 대한 역난독화

본 논문에서 제안하는 동적 제어흐름 난독화를 역난독화하기 위해서는 공격자는 조건문 분석 및 코드 블록 분석을 수행해야 한다. 조건문 분석의 경우 공격자는 먼저 조건문이 더미 조건문이라고 판단할 수 있어야 한다. 그러나 2.2장에서 설명한 결정론적 방법을 사용하는 경우 공격자는 조건문 정보만으로는 해당 조건문이 더미 조건문이라고 판단하기 어렵다. 만약 더미 조건문이라고 판단하더라도 Fig. 3의 (e)처럼 다음 제어흐름을 가진 코드 블록이 [1′2′], [1″2″] 그리고 [1‴2‴3‴4‴]라면 코드 블록을 분석하지 않고는 쉽게 역난독화할 수 없으므로 코드 블록에 대한 분석이 반드시 필요하다.

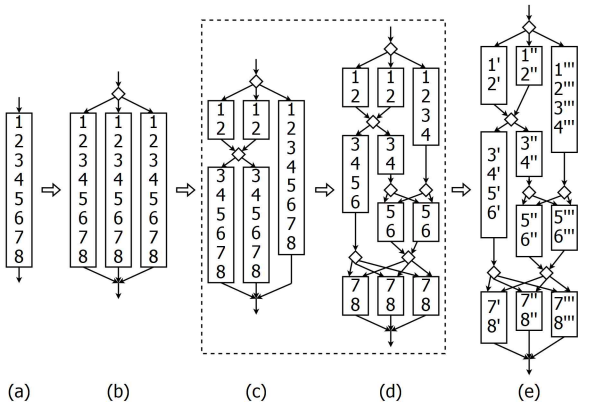


Fig. 3. 직선코드 블록에 대한 동적 제어흐름 난독화 과정

코드 블록 분석의 경우 공격자는 더미 조건문 이후 제어흐름을 가지는 두 개 이상의 코드 블록이 동일한 기능성을 가진다고 판단할 수 있어야 한다. 그러나 각 코드 블록은 데이터 난독화 기법이 적용되어 있으므로 공격자는 데이터 난독화에 대한 역난독화 기술이 반드시 필요하다. 또한 Fig. 3의 (e)처럼 다음 제어흐름을 가진 코드 블록이 [1′2′]과 [1‴2‴3‴4‴]라면 두 코드 블록의 기능성은 서로 달라서 공격자는 추가적인 분석이 필요하다. 이에 따라 공격자는 단순히 코드 블록만 비교하는 것뿐만 아니라 각 코드 블록을 병합해가며 동일한 기능성 여부를 확인해야 하므로 쉽게 역난독화할 수 없다.

3.3 다른 제어흐름 난독화 기법과 결합

본 논문에서 제안하는 동적 제어흐름 난독화 기법은 다른 제어흐름 난독화 기법과 결합하여 보다 강력한 난독화 기능을 제공할 수 있다. Fig. 4는 불분명 술어와 더미 조건문을 동시에 사용한 제어흐름 난독화를 보여준다. 코드 블록 [5 6]은 불분명 술어 기법을 통해 생성된 더미 코드 블록으로 실제로는 실행되지 않는 코드 블록으로 다른 코드 블록과 동일한 기능성을 가지지 않는다. 따라서 더미 코드 블록 [5 6]은

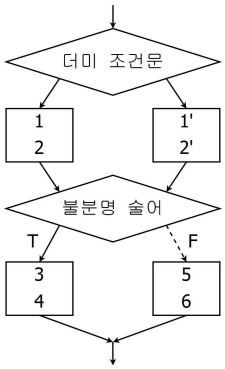


Fig. 4. 불분명 술어와 더미 조건문을 동시에 사용하는 제어흐름 난독화

코드 블록 분석을 통한 역난독화를 어렵게 한다.

IV. 결론

기존 제어흐름 난독화 기법은 고정된 제어흐름을 가지므로 동적 분석에 취약점을 가진다. 이러한 한계점을 해결하기 위해 동적으로 제어흐름을 결정하는 연구가 있었지만 처음 제어흐름이 결정되면 이후 제어흐름이 고정되므로 여전히 동적 분석에 취약하다.

본 논문에서는 기존 제어흐름 난독화 기법이 가진 한계점을 해결하기 위해 모든 제어흐름이 동적으로 결정되는 새로운 동적 제어흐름 난독화 기법을 제안한다. 본 논문에서 제안하는 제어흐름 난독화의 역난독화를 위해서는 추가적인 데이터 역난독화 기술이 필요하며 다른 제어흐름 난독화 기법과 결합하여 사용할 수 있다.

[참고문헌]

[1] C. Collberg, C. Thomborson and D. Low, A taxonomy of obfuscating transformations, Technical Report, Department of Computer Science, 1997.

[2] D. Low, Java control flow obfuscation, Master's Thesis, Department of Computer Science, Jun, 1998.

[3] C. Collberg, C. Thomborson, and D. Low, Manufacturing cheap, resilient, and stealthy opaque constructs, In Principles of Programming Languages 1998, Jan, 1998.

[4] T. Hou, H. Chen, and M. Tsai, Three control flowo bfuscation methods for Java software, IEE Proceedings-Software, pp. 80-86, Jan, 2006.

[5] T. Ogiso, Y.Sakabe, M.Soshi, and A. Miyaji, Software obfuscation on a theoretical basic and its implementation, IEICE TRANSACTIONS on Fundamentals of Electronics, pp. 176-186, Jan, 2003.

[6] S. K. Udupa, S. K. Debray and M. Madou, Deobfuscation: reverse engineering obfuscated code, 12th Working Conference on Reverse Engineering, pp. 45-54, November, 2005.

[7] J. H. Suk, Y. B. Lee and D. H. Lee, SCORE: Source Code Optimization & REconstruction, IEEE Access, pp. 129478-129496, 2020.

[8] Ming, Jiang, et al., Loop: Logic-oriented opaque predicate detection in obfuscated binary code, Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, pp. 757-768, October, 2015.

[9] Palsberg, Jens, et al., Experience with software watermarking, Annual Computer Security Applications Conference, pp. 308-316, December, 2000.

[10] Xu, Dongpeng, Jiang Ming, and Dinghao Wu, Generalized dynamic opaque predicates: A new control flow obfuscation method. International Conference on Information Security, September, 2016.