ARM 프로세서에서 고급수준 난독처리의 성능분석

장혜영○ 조성제
단국대학교 정보컴퓨터학부
hychang@dankook.ac.kr, sjcho@dankook.ac.kr

Performance Evaluation of High-Level Obfuscation on ARM Processor

Hyeyoung Chang○, Seongje Cho
Division of Information and Computer, Dankook University

1. 서론

소프트웨어의 복제 및 도용은 개발자와 사회에 지대한 피해를 끼친다. 이는 재정적 손실뿐 아니라 창의적인 개발 의지를 상실하게 하여 전반적인 생산성을 낮추게 된다. 소프트웨어의 지적재산권을 보호하는 대표적인 기법에는 불법복제를 방어하기 위한 워터마킹(watermarking), 변조공격을 방어하기위한 변조방지(tamper-proofing)와 암호화, 역공학을 방어하기위한 난독처리기술(obfuscation)등이 있다[1-3]. 이처럼 여러 역공학 도구 및 기술로부터 소프트웨어를 보호하는 방법들이 많이 제안되었지만, 가장 강력하고 널리 연구되고 있는 방법 중의 하나가 난독처리기술이다[4-6]. 난독처리기술은 역공학을 방어하기 위해 공격자가 이해할 수 없도록 코드를 변환(transformation)하여 코드를 최대한 복잡하고 혼란스럽게 만드는 것이다. 본 논문에서는 ARM 기반의 임베디드 시스템에서 소스 분석 공격 및 역 어셈블 공격으로부터 C와 C++소스프로그램을 보호하기 위한 난독처리기술을 구현하여 실험하였다. 또한, 난독처리기술을 적용하기 전과 후의 프로그램 코드를 복잡도(potency), 복원력(resilience), 비용(cost), 어셈블리 코드 등의 면에서 비교하여 분석하였다.

2. 난독처리 알고리즘구현과 성능평가

본 연구 수행을 위해 Intel XScale PXA255 400MHz CPU, SDRAM 128Mbyte, Flash ROM 32Mbyte, Embedded Linux(커널 2.4.19)에서 아래의 난독처리 알고리즘을 구현하여 실험하였다.

2.1 변수분할 알고리즘

변수분할 알고리즘은 논리형(boolean)이나 크기가 제한적인 다른 기본타입 변수들을 여러 개의 변수로 나누거나 변수의 사용자 정의 구조체 또는 값을 반환하는 함수로 대체하는 방법이다. 본 논문에서는 아래의 그림1과 같이 32비트의 정수형 변수를 16비트의 unsigned short형의 두 변수로 나누어 단항 연산자, 다항 연산자, 값을 할당 부분을 함수로 대체하는 방식으로 구현했다.

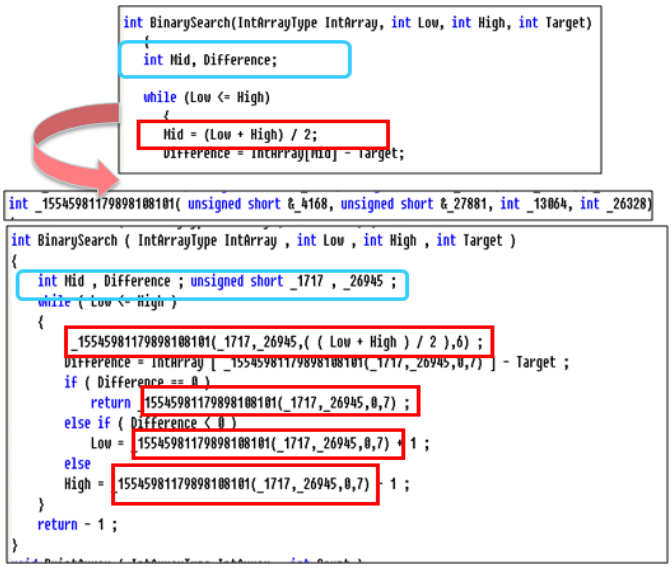

그림 1 변수분할 알고리즘 적용 전과 후

2.2 배열중첩 알고리즘

배열 중첩 알고리즘은 배열의 특성 및 용도를 공격자 및 역공학도구가 분석하기 어렵도록 만드는 기법으로 본 논문에서는 1차원 배열을 2차원 배열로 차원을 증가시키는 배열 중첩 알고리즘을 구현하였으며, 그림 2에 결과가 나타나 있다.

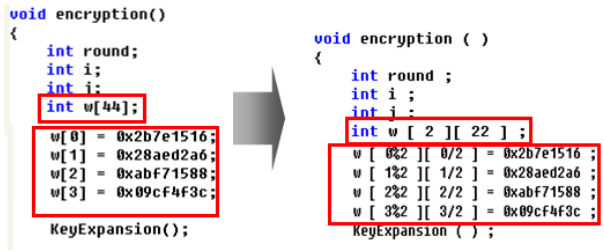

그림 2 배열 중첩 알고리즘 적용 전과 후

2.3 루프 조건 확장 알고리즘

조건문이나 반복문에 조건을 추가하여 종료 조건을 복잡하게 만드는 기법이다. 이때 어떠한 조건식을 추가하더라도 본래 의도한 종료시점에는 변화

가 없어야 한다. 본 논문에서 '&&' 연산자에 무조건 참이 되는 조건식을 추가하고 '||' 연산자는 무조건 거짓이 되는 조건식을 추가하여 원 소스의 조건식에는 영향을 주지 않게 구현하였다.

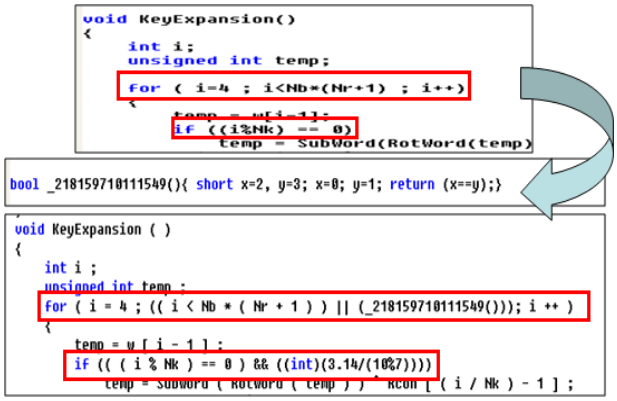

그림 3 루프 조건 확장 알고리즘 적용

2.4 부가 피연산자 삽입 알고리즘

부가 피연산자 삽입 알고리즘은 간단한 연산식에 피연산자를 추가하여 수식을 좀 더 복잡하게 만들어주는 기법이다. 그림 4와 같이 함수의 인자 값으로 사용된 수식 "cy-4"에 형 변환된 추가적인 수식이 삽입된 것을 확인할 수 있다.

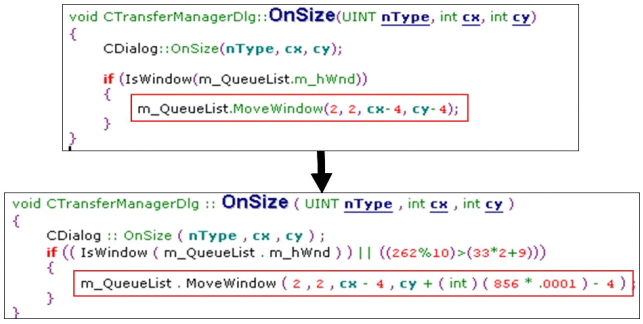

그림 4 부가 피연산자 삽입 알고리즘 적용

2.5 성능평가

난독처리기술의 품질(quality)를 평가하는 기준으로 크게 복잡도(potency), 복원력(resilience), 비용(오버헤드)이다[7,8].

표 1 난독처리 전후의 파일 크기 및 시간 측정

	디프트리 프로그램				거품정렬 프로그램			
	원본	데이터	제어	데이터+제어	원본	데이터	제어	데이터+제어
소스파일	4360	6680	6888	8375	3007	4205	3358	5181
오브젝트파일	9836	10692	11640	13668	4832	5964	5688	7748
실행시간	9.63	9.65	15.30	19.65	0.16	0.23	0.17	0.25

그 결과 디프트리 프로그램은0.7194, 거품정렬 프로그램은 0.7885만큼 복잡도가 증가하였고 [8]에서 소개된 복원력 측정 기준을 참고했을 때10~13만큼 복원력이 증가하였다. 비용의 결과는 표1과 같다. 위의 알고리즘을 모두 적용했을 때 오브젝트 파일의 크기(byte)가 디프트리 프로그램에서 약1.4배, 거품정렬 프로그램에서는 약 1.6배 증가한 것을 확인할 수 있었다. 또한 디프트리 프로그램의 실행 시간을 측정하기 위해500회 삽입과 삭제 연산을 10회 수행한 후 평균값을 계산하였고 거품정렬 프로그램의 실행 시간을 측정하기 위해서는 500회 삽입하여 정렬을 10회 수행한 평균값을 계산하였다.

3. 결론

본 논문에서는 역공학 공격으로부터 임베디드 시스템의 소프트웨어를 보호하는데 매우 효과적인 방어 방법인 난독처리 기술을 연구하여 구현하였다. C와 C++ 소스의 데이터 구조를 변형시키는 데이터 난독처리기술, 제어흐름은 변형시키는 제어 난독처리기술, 레이아웃의 형태를 변환하는 레이아웃 난독처리기술을 같이 접목하여 ARM 기반 임베디드 보드에서 실험하였다. 실험 결과, 난독처리된 소스 코드가 실행시간 오버헤드를 일부 유발시키긴 하지만 복잡도가 증가하고 복원력이 강화되었음을 확인할 수 있었다. 또한 어셈블리 코드로 난독처리 적용 전 후의 코드를 비교한 결과 난독처리기술이 잘 적용됐음을 확인함으로써 프로그램 보호에는 효과적임을 알 수 있었다

참고문헌

[1] C. Collberg and C. Thomborson, "Watermarking, Tamper-proofing, and Obfuscation－Tools for Software Protection," *IEEE Trans. Software Eng.*, Vol. 28, no. 8, pp. 735-746, 2002.

[2] Bin Fu, Golden G. Richard III, Yixin Chen, and Adbo Husseiny, "Some New Approaches For Preventing Software Tampering," Proc. of the 44th ACM Southeast Regional Conference (ACM SE'06), pp. 655-660, Mar. 2006.

[3] P. C. van Oorschot, "Revisiting Software Protection", 6th ISC 2003, Springer-Verlag LNCS 2851, pp. 1-13, Oct. 2003

[4] M. R. Stytz and J. A. Whitaker, "Software Protection: Security's Last Stand?", IEEE Security & Privacy, 1(1), pp. 95-98, Jan. 2003.

[5] Christopher Kruegel, William Robertson, Fredrik Valeur and Giovanni Vigna, "Static Disassembly of Obfuscated Binaries," Proc. of the 13th USENIX Security Symposium, pp. 255-270, Aug. 2004.

[6] Colin W. Van Dyke, "Advances in Low-Level Software Protection," Ph. D. Thesis, Oregon State University, Jun. 2005.

[7] C. Collberg, C. Thomborson, and D. Low, "*A Taxonomy of Obfuscating Transformations*," Tech. report 148, Dept. of Computer Science, University of Auckland, New Zealand, 1997; www.cs.arizona.edu/~collberg/Research/Publications/ColbergThomborsonLow97a/

[8] B. Barak et al., ""On the (Im)possibility of Obfuscating Programs," *Advances in Cryptology－Crypto 2001, Proc. 21st Ann. Int'l Cryptology Conf.*, LNCS 2139, Springer-Verlag, pp. 1-18, 2001.