난독화 바이너리 분석 방안

A Practical Approach to Analyze Obfuscated Binaries

저자
(Authors)

최석우
Choi, Seokwoo

출처
(Source)

정보과학회지 36(3), 2018.3, 26-31 (6 pages)
Communications of the Korean Institute of Information Scientists and Engineers 36(3), 2018.3, 26-31 (6 pages)

발행처
(Publisher)

한국정보과학회
KOREA INFORMATION SCIENCE SOCIETY

URL

http://www.dbpia.co.kr/Article/NODE07403015

APA Style

최석우 (2018). 난독화 바이너리 분석 방안. 정보과학회지, 36(3), 26-31.

난독화 바이너리 분석 방안

국가보안기술연구소 ▌**최석우**

1. 서 론

코드 난독화(code obfuscation)는 프로그램 변환 기법의 일종으로 프로그램의 원래 의미를 보존하면서 읽거나 분석하기 어렵게 만드는 프로그램 보호 기법이다[1]. 개발자들이나 프로그램 소유자는 프로그램의 저작권 보호 또는 프로그램 안에 숨겨져 있는 중요 자산 보호를 위해 코드 난독화를 적용한다. 악성코드 제작자들은 악성코드 탐지 방지, 악성코드 분석 방지 등 자신의 악성 행위를 숨기기 위해 코드 난독화를 적용한다.

코드 역난독화(code deobfuscation)는 난독화된 코드를 분석하기 쉽게 바꾸거나 이해할 수 있는 형태로 바꾸거나 코드로부터 유용한 정보를 추출하기 위한 프로그램 분석 기법이다. 난독화 코드의 안전성을 검증하거나 난독화 된 악성코드를 분석하기 위하여 코드 역난독화를 적용한다.

코드 난독화와 코드 역난독화 기법은 암호와 같이 수학적으로 안전하다고 증명할 수 없다[2]. 코드 역난독화 문제는 일반적인 프로그램 분석에 비해서 더 어렵다고 할 수 없다[3]. 암호 해독은 암호화 알고리즘을 알고 있다고 암호를 풀 수 없지만 난독화 알고리즘을 알게 되면 프로그램 분석 기법에 의해서 역난독화를 수행할 수 있는 것이다. 따라서 난독화 기법의 평가는 프로그램의 복잡도나 역공학자를 대상으로 하는 실험과 같은 소프트웨어 공학에서의 측정 방식으로 이루어진다[4,5].

이론적인 한계에도 불구하고 코드 난독화 기법을 적용하는 것은 난독화 코드를 분석하는데 시간이 많이 소요되기 때문이다. 역난독화 기법을 적용하기 위해서는 먼저 난독화 기법에 대한 식별이 필요한데 난독화 기법은 다양한 조합으로 사용될 수 있기 때문에 각 기법을 식별하는 것이 어려울 수 있다. 또한 난독화 기법은 포인터 에일리어스 기법을 적극적으로 이용하고 있기 때문에 자동화 된 프로그램 분석 기법을 활용하는 것이 매우 어렵게 된다[6]. 난독화 기법에 대한 대응 방법이 알려지더라도 이를 무력화하기 위한 난독화 방법을 쉽게 만들 수 있다. 예를 들어 최근에 심볼릭 실행을 통한 역난독화 기법이 알려지면서[7] 이에 대응하기 위한 난독화 기법이[8] 바로 제안되는 것을 볼 수 있다.

이와 같이 난독화 기법과 역난독화 기법은 새로운 기법이 제안 되면 이에 대응하기 위한 기법이 다시 제안 되는 군비경쟁 상태에 있다고 볼 수 있다. 따라서 난독화 된 바이너리 코드를 분석하기 위해서는 전통적인 난독화 기법 뿐 아니라 새롭게 제안되고 있는 역난독화 기법까지 이해하고 이에 대한 대응 방안을 지속적으로 연구할 필요가 있다.

이 연구에서는 악성코드가 가장 많이 이용하고 있는 바이너리 실행파일 난독화를 분석하기 위해 바이너리 난독화 및 역난독화 기법에 대해서 소개한다. 2장에서는 전통적인 난독화 기법을 소개하고 3장에서는 난독화 코드를 분석하기 위한 역난독화 기법과 향후 연구 방향을 제시한다.

2. 바이너리 난독화 기법

2.1 컴파일 단계별 코드 난독화

코드 난독화 기법은 프로그램 컴파일과 빌드 과정의 각 단계에서 적용 가능하다.

소스코드 단계에서 난독화 기법을 적용하는 언어는 자바스크립트, VB스크립트와 같은 소스코드 형태로 배포 하는 스크립트 언어에 해당한다. 웹 상에서 스크립트 형태로 작동하는 악성코드들은 소스 코드 난독화 기법을 적용하고 있다. 공개된 형태의 난독화 도구와 역난독화 도구들이 많이 존재하고 있다. 난독화와 역난독화가 비교적 용이하기 때문에 악성코드 제작자들은 스크립트 단계에서 직접 악성 행위를 수행하기 보다 바이너리 형태의 악성코드를 생성하는 드로퍼(dropper) 용도로 스크립트 언어를 활용하고 있다.

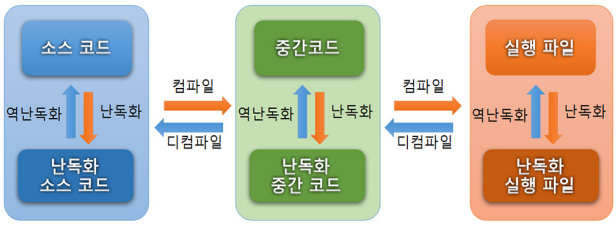

그림 1 컴파일 단계별 코드 난독화 적용

중간코드 단계에서 난독화는 컴파일 단계에서 생성한 중간 코드를 대상으로 난독화를 수행하는 것이다. 컴파일러 툴 체인에서 중간코드 단계에서 프로그램 분석과 변환을 수행한다. 이 단계에서 난독화를 수행하는 도구로 OLLVM이 있다[9]. OLLVM은 C/C++, OCaml 난독화를 지원하며 오픈 소스로 공개되어 이를 활용하여 다양한 플랫폼에서 적용할 수 있는 도구들이 만들어지고 있다.

대다수의 악성코드들은 실행파일 단계에 난독화를 적용하고 있다. 실행 파일을 난독화 할 수 있는 다양한 상용 도구들이 존재하기 때문에 실행 파일을 난독화 도구로 보호한 후 배포하는 형태를 이용할 수 있다. 악성코드 제작자들은 실행파일을 직접 수정하여 고유한 난독화 기법을 적용하기도 한다. 실행파일 단계에서 난독화를 적용하기 위해서는 실행파일을 분석 가능한 어셈블리 코드나 중간코드 형태로 변환한다. 여기에 난독화 변환을 수행한 후 다시 컴파일과 빌드 과정을 거쳐 난독화 된 실행 파일을 만든다. 실행파일이 자바 바이트코드 같이 풍부한 타입 정보를 포함하고 있는 경우에는 역난독화 변환이 용이하다. 실행파일이 윈도 PE나 리눅스 ELF와 같은 바이너리 형태인 경우 타입 정보를 제거할 수 있어 역난독화 변환이 쉽지 않다.

2.2 난독화 기법 분류

코드 난독화 기법의 분류 방법은 C. Collberg가 처음 제안 하였다[10]. Collberg의 난독화 기법 분류는 자바 바이트코드를 대상으로 한 것으로 제어 난독화(control obfuscation), 데이터 난독화(data obfuscation), 레이아웃 난독화(layout obfuscation), 방지 난독화(preventive obfuscation)의 네 가지를 포함한다. 제어 난독화는 더미코드 삽입, 가짜 분기문 삽입, 제어 흐름 평탄화 등 제어 흐름을 변형하여 분석을 방해하는 방식이다. 데이터 난독화는 변수 값을 바이트 단위로 쪼개고 연산하며 합치는 기법, 문자열은 인코딩하고 디코딩 하는 등 데이터를 읽기 어렵게 만드는 기법이다. 레이아웃 난독화는 자바 바이트코드 등 중간코드 형태에서 실행 파일에서 타입 정보를 제거하거나 식별자 이름을 바꾸는 등 프로그램의 겉보기 형태를 바꾸는 기법이다. 방지 난독화는 분석 도구를 방해하는 기법이다. 분석 도구를 방해하기 위한 포인터 에일리어스 도입 등이 이에 해당한다[11].

바이너리 난독화에 대한 분류는 데이터 난독화와 제어 난독화, 동적분석 방해와 정적분석 방해를 기준으로 할 수 있다[12]. 바이너리 난독화는 레이아웃 난독화가 없고 방지 난독화가 동적분석 방해와 정적분석 방해로 분류 되고 있으며 동시에 여러 분류에 포함될 수 있기 때문에 전통적인 방법에 비해 더 정확한 분류가 가능하다.

이 장에서는 대표적인 x86 바이너리 난독화 도구들인 OLLVM[13], Themida[14], VMProtect[15]에서 제공하고 있는 난독화 기법들 중 대표적인 기법들에 대해서 소개하려고 한다.

2.3 제어 흐름 평탄화

제어 흐름 평탄화(CFG Flattening)는 제어 흐름 그래프 상에 나타나는 각 베이식 블록(basic block)을 평평하게 펼쳐 두고 스위치 문과 유사한 역할을 하는 디스패처(dispatcher)를 통해 각각에 접근하게 하는 방법이다[16]. 이 방법을 이용하면 제어 흐름이 한 곳으로

표 1 바이너리 난독화 기법 분류의 예

	Target		Against	
	Control	Data	Static	Dynamic
CFG flattenging	●		●	
Jump encoding	●		●	
Opaque predicates	●			
VM(virtual-machines)	●	●	●	●
Polymorphism(self-modification, resource ciphering)	●		●	
Call/Stack tampering	●		●	
Anti-debug/anti-tampering	●	●		●
Signal/Exception	●		●	

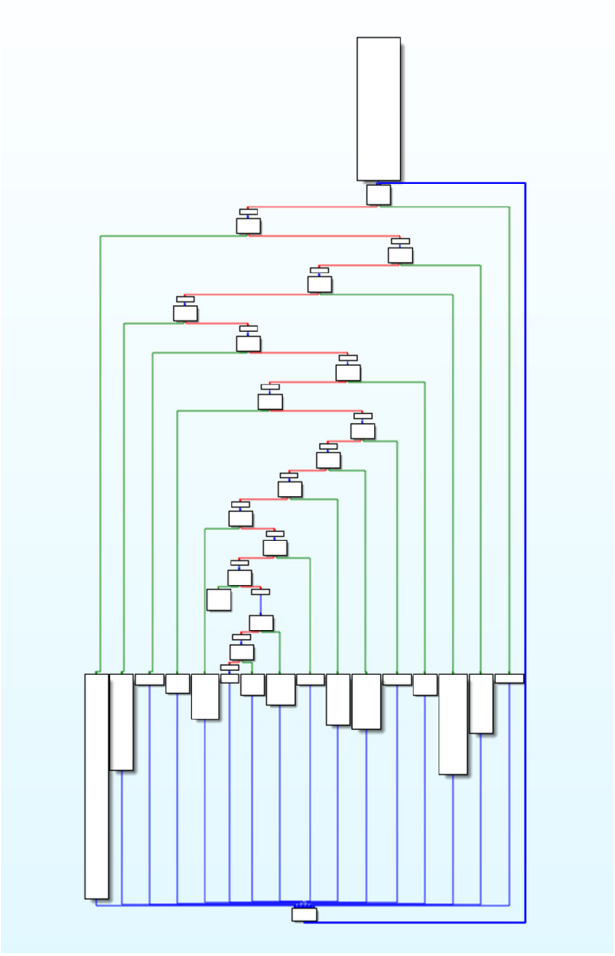

그림 2 제어 흐름 평탄화를 적용한 프로그램

모이고 인코딩 된 값을 통해서 다음 실행할 베이식 블록을 결정하게 된다. 분석가는 제어 흐름 구조를 볼 수 없기 때문에 한 눈에 분석하는 것이 어렵게 된다. 다음 베이식 블록을 결정하기 위한 값만 계산하면 되므로 난독화에 의한 실행 속도 저하가 적고 프로그램 크기도 많이 차이 나지 않는 방법이다.

OLLVM은 제어 흐름 평탄화 난독화 기법을 선택할 수 있도록 한다. C++ 프로그램의 특정 함수에 제어흐름 평탄화 기법을 적용한다. 그림 2는 OLLVM에 의해 C++ 프로그램에 제어 흐름 평탄화를 적용한 후 IDA Pro 디스어셈블러에 의해 변형된 CFG를 보여 준다. 이 그림에서 실제로 의미 있는 부분은 제어 흐름이 전부 모이는 가장 아래의 베이식 블록에 연결된 부모 베이식 블록들이다.

2.4 복잡한 조건식

코드 난독화에서 복잡한 조건식(opaque predicate)은 항상 같은 값을 가지는 조건식으로 프로그램 분석을 방해하기 위해서 복잡하게 만들어진 식을 의미한다. 예를 들어 $7y^2 - 1 = x^2$ 은 항상 거짓이며 Sandmark[18]에서 활용하고 있는 복잡한 조건식이다. 이 수식은 만족불가능(unsatisfiable)한 식이지만 Z3 Solver[19]에서 실행하면 UNSAT Unknown 또는 Timeout 결과를 낸다. 복잡한 조건식은 더미 코드 삽입이나 안티 디스어셈블에도 활용할 수 있다[20].

2.5 가상화 난독화

가상화 난독화(virtualization obfuscation)는 프로그램 내부에 가상 CPU에 대한 에뮬레이터를 내장하고 원본 프로그램의 일부를 가상 CPU가 실행할 수 있는 바이트코드(bytecode) 형태로 변환하고 실행하는 방식이다. 가상화 난독화 실행 파일은 가상 CPU 명령을 실행할 수 있는 명령어 핸들러(instruction handler), 가상 명령 프로그램인 바이트코드, 가상 명령어 주소 테이블 등으로 구성 되어 있다. Themida나 VMProtect 같은 난독화 도구들은 프로그램의 의미를 숨기기 위하여 가상화 난독화 기법을 적용한다. 가상화 난독화 기법은 더미 코드 삽입이나 복잡한 조건식, 데이터 난독화 등 여러 난독화 기법들과 함께 적용되어 코드 분석을 매우 힘들게 한다. 예를 들어 Themida나 VMProtect에서 가상화를 적용하는 경우 명령어 실행 트레이스가 100배~10000배 이상 늘어나 난독화 된 영역에 대한 실행 속도가 매우 느려지게 된다. 동시에 실행 트레이스를 텍스트 형태로 추출할 때 간단한 샘플 프로그램도 수 기가바이트의 용량을 가지게 되어 분석을 힘들게 한다. 가상화 난독화 방식의 단점은 실행 속도가 매우 느려진다는 것이다.

2.6 자가 변형

자가변형(self-modification)은 실행 중에 코드를 변경하면서 실행하는 방법으로 정적 분석을 방해하는데 효과적이다[21]. 실행 압축은 자가변형의 일종으로 실행 압축 된 실행파일은 압축된 원래의 실행 파일과 압축 해제 코드를 실행파일에 포함하여 실행시에 원래의 코드로 변경 후에 실행하는 방법이다. 난독화 도구들은 실행 압축과 자가 변형을 조합하여 정적 분석과 동적 분석을 동시에 어렵게 하고 있다. 예를 들어 Themida 같은 경우 프로그램을 실행하면서 다음 명령이 저장된 주소에 다른 명령을 기록하여 실행 전에 디버거로 확인한 코드와 실제로 실행 되는 코드가 다르다.

2.7 안티 리버싱

안티 리버싱(anti-reversing) 기법은 분석 도구를 사용하기 어렵게 만드는 기법이다. 안티 리버싱 기법에는 디버거를 탐지하거나 디버거의 버그를 활용하여 멈추게

하는 안티 디버깅(anti-debugging) 기법, 악성코드 분석을 위해서 활용하는 가상머신이나 분석용 샌드박스를 탐지하는 안티 가상머신(anti-VM) 기법, 디버깅 중에 값을 조작하는 등 변경된 내용을 탐지하기 위한 변조 방지(tamper proof) 기법 등이 포함된다.

3. 바이너리 역난독화 기법

바이너리 역난독화 기법은 난독화 된 바이너리에서 중요한 정보를 찾아내기 위한 프로그램 분석 기법이다. 디버깅 정보 제거나 이름 제거와 같이 단방향 변환에 의한 난독화를 이용했을 때는 역난독화가 불가능하지만 그렇지 않은 경우 프로그램 의미가 보존되기 때문에 이론적으로 역난독화가 가능하다. 궁극적으로는 원래의 코드를 복원하는 것을 목표로 하지만 중요한 정보를 찾아내는 것을 일차적인 목표로 한다. 중요한 정보는 잠재적 위협, 사용하고 있는 알고리즘, 암호 키 등이 해당 된다.

역난독화 자동화에 대한 시도는 지금까지 계속 이어졌으나 역난독화 기법이 알려졌을 때 이에 대응하기 위한 난독화 기법이 나오기 때문에 각 난독화 기법에 대해 개별적으로 대응하는 것이 필요하다.

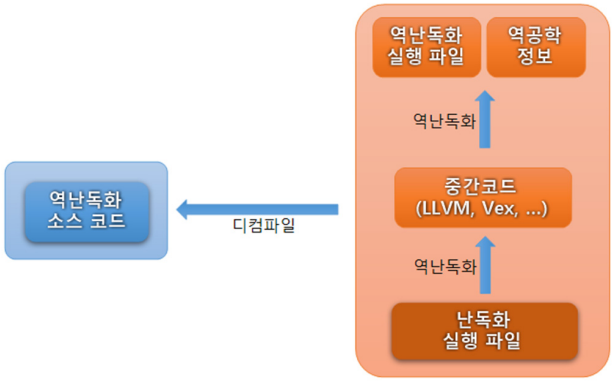

그림 3 역난독화 접근법

3.1 실행 압축 기법 대응

실행 압축 기법은 정적 분석은 방해할 수 있지만 동적 분석을 하는 경우 원래의 코드가 다 압축 해제되었을 때는 코드를 복원 할 수 있다. 실행압축을 해제하기 위해서는 원래 프로그램이 다 압축 해제 되고 원래의 실행 주소를 실행하는 시점을 찾아야 한다. 이를 위한 기법들이 제안 [22, 23] 되었으며 이 기법들은 상용 패커들에 대해 잘 적용 된다.

3.2 안티 리버싱 대응

안티 리버싱 기법은 분석 도구나 분석 환경을 탐지한다. 이를 회피하기 위해서 디버거에서는 안티안티 디버거 플러그인을 이용한다 [24, 25]. Intel Pin이나 Valgrind 같은 동적 바이너리 계측(dynamic binary instrumentation) 도구들은 프로그램을 실행할 때 다른 위치로 코드를 옮기거나 중간 언어 형태로 실행하기 때문에 디버거 탐지 기법 등에 안전하여 안티 리버싱 대응을 위해서 활용 되고 있다. 그러나 안티 DBI 기법[26]이 제안되고 있어서 이에 대한 새로운 대응 기법 개발이 필요하다.

3.3 복잡한 조건식 대응

복잡한 조건식은 프로그램 분석 도구가 활용하고 있는 SAT Solver, SMT Solver들을 무력화시킬 수 있다. 난독화 도구들에서는 제한된 개수의 복잡한 조건식을 활용하고 있다. 따라서 이를 이용해 난독화 한 경우에는 복잡한 조건식이 반복해서 나타난다. 이 성질을 이용하여 반복된 복잡한 조건식을 검색하여 조건식을 상수로 치환할 수 있다. 또한 SMT Solver에 의해 풀 수 있는 조건식들도 많이 있기 때문에[27] 기본적으로는 심볼릭 실행을 이용하고 풀리지 않는 경우에 패턴을 이용하여 풀 수 있다. 패턴을 이용할 때 코드 삽입과 복잡한 조건식이 함께 적용되기 때문에 복잡한 조건식을 바로 찾을 수 없는 경우가 생긴다. 이 때 중간 코드에 대한 간략화 또는 어셈블리 코드에 대한 코드 최적화를 적용한 후 복잡한 조건식에 대한 패턴과 일치시킬 수 있다.

3.4 가상화 난독화 대응

가상화 난독화 코드를 역난독화 하기 위하여 크게 두 가지 접근법이 있다. 하나는 가상화 난독화 구조를 완전히 이해한 후 정적 분석에 의해 역난독화 하기 위한 방법이다 [28, 29]. 이 방법은 난독화 도구의 새로운 버전이 출시 되면서 쉽게 무력화 될 수 있다는 단점이 있다. 이 방식의 장점은 프로그램을 완전히 이해할 수 있는 형태로 변환한다는 것이다. 다른 하나는 동적 분석을 중심으로 프로그램을 축약하고[7, 30] 커버리지를 넓혀 가며[27] 분석하는 방법이다. 동적 분석 방식은 잠재적인 위협을 탐지하는데 한계가 있으나 단순한 악성코드의 경우 분석 가능해지는 장점이 있다. 아직까지 동적 분석과 정적 분석 방법 모두 다른 난독화 기법과 결합하여 이용되는 가상화 난독화 코드에 대한 효과적인 대응 방안이 되지 못하고 있다. 관련 연구에서 이용한 실험 데이터는 장난감 예제 또는 벤치마크이거나 이전 버전의 난독화 도구를 이용하여 분석 가능한 악성코드이다. 역난독화에 대한 평가 기준도 CFG의 유사도나 난독화 기법을 얼마나 제

거했는지로 제시하고 있어 현실적으로 이 방법을 활용하여 악성 코드를 분석하는데 한계가 있다.

최근에 머신 러닝 기법을 이용하는 프로그램 합성 기법을 통해 가상화 난독화 핸들러를 역난독화 하는 기법이 제안 되었다[31]. 이 접근 방법은 기존의 난독화 구조와 가상 CPU에 대한 이해가 충분할 때 적용해 볼수 있다.

4. 결 론

난독화와 역난독화는 이론적으로 수학적으로 안전하다고 검증된 방법이 아니며 방법이 알려졌을 때는 분석이 가능한 프로그램 분석의 문제이다. 따라서 공격자들은 코드 난독화 기술을 통해 악성코드 분석 시간을 지연 시켜 시간을 확보할 수 있다. 프로그램 보호를 목적으로도 자주 업데이트가 일어난다는 가정하에 난독화 기법을 적용하는 것이 효과적이다. 분석가들은 난독화 코드에 대한 일반적이고 자동화 가능한 역난독화 기술을 적용할 수 없다. 난독화 코드 분석을 위해 난독화 기법에 대한 이해와 식별이 먼저 필요하며 각 난독화 기법에 대한 대응을 해야 한다. 분석가들은 난독화 기법에 대한 이해를 바탕으로 DBI, 디버거와 같은 다양한 도구들과 심볼릭 실행, 슬라이싱 등 프로그램 분석 기술을 이용하여 역난독화를 할 수 있으며 부분적으로 자동화 할 수 있는 역난독화 도구를 개발할 수 있다.

여러 역난독화 기법 중 아직까지 가상화 난독화에 대해 효과적으로 적용할 수 있는 연구가 존재하지 않으며 이를 위한 추가적인 연구가 필요하다.

참고문헌

[1] Nagra, Jasvir, and Christian Collberg. Surreptitious Software: Obfuscation, Watermarking, and Tamperproofing for Software Protection: Obfuscation, Watermarking, and Tamperproofing for Software Protection. Pearson Education, 2009.

[2] Barak, Boaz, et al. "On the (im) possibility of obfuscating programs." Annual International Cryptology Conference. Springer, Berlin, Heidelberg, 2001.

[3] Appel, Andrew. "Deobfuscation is in NP." Princeton University, Aug 21 (2002)

[4] Anckaert, Bertrand, et al. "Program obfuscation: a quantitative approach." Proceedings of the 2007 ACM workshop on Quality of protection. ACM, 2007.

[5] Ceccato, Mariano, et al. "The effectiveness of source code obfuscation: An experimental assessment." Program Comprehension, 2009. ICPC'09. IEEE 17th International Conference on. IEEE, 2009.

[6] Collberg, Christian S., and Clark Thomborson. "Watermarking, tamper-proofing, and obfuscation-tools for software protection." IEEE Transactions on software engineering 28.8 (2002): 735-746.

[7] Yadegari, Babak, and Saumya Debray. "Symbolic execution of obfuscated code." Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security. ACM, 2015.

[8] Banescu, Sebastian, et al. "Code obfuscation against symbolic execution attacks." Proceedings of the 32nd Annual Conference on Computer Security Applications. ACM, 2016.

[9] Junod, Pascal, et al. "Obfuscator-LLVM--software protection for the masses." Software Protection (SPRO), 2015 IEEE/ACM 1st International Workshop on. IEEE, 2015.

[10] Collberg, Christian, Clark Thomborson, and Douglas Low. A taxonomy of obfuscating transformations. Department of Computer Science, The University of Auckland, New Zealand, 1997.

[11] Collberg, Christian, Clark Thomborson, and Douglas Low. "Manufacturing cheap, resilient, and stealthy opaque constructs." Proceedings of the 25th ACM SIGPLAN-SIGACT symposium on Principles of programming languages. ACM, 1998.

[12] Robin David & Sébastien Bardin. “Code Deobfuscation: Intertwining Dynamic, Static and Symbolic Approaches.” Black Hat Europe, 2016.

[13] Obfuscator-LLVM. https://github.com/obfuscator-llvm/obfuscator/wiki

[14] Themida. https://www.oreans.com/themida.php

[15] VMProtect. http://vmpsoft.com

[16] Chow, Stanley, et al. "An approach to the obfuscation of control-flow of sequential computer programs." International Conference on Information Security. Springer, Berlin, Heidelberg, 2001.

[17] 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. ACM, 2015.

[18] Collberg, Christian, G. R. Myles, and Andrew Huntwork. "Sandmark-a tool for software protection research." IEEE security & privacy 99.4 (2003): 40-49.

[19] Z3Prover/z3. https://github.com/Z3Prover/z3
[20] Linn, Cullen, and Saumya Debray. "Obfuscation of executable code to improve resistance to static disassembly." Proceedings of the 10th ACM conference on Computer and communications security. ACM, 2003.
[21] Madou, Matias, et al. "Software protection through dynamic code mutation." International Workshop on Information Security Applications. Springer, Berlin, Heidelberg, 2005.
[22] Kang, Min Gyung, Pongsin Poosankam, and Heng Yin. "Renovo: A hidden code extractor for packed executables." Proceedings of the 2007 ACM workshop on Recurring malcode. ACM, 2007.
[23] Oberheide, Jon, Michael Bailey, and Farnam Jahanian. "PolyPack: an automated online packing service for optimal antivirus evasion." Proceedings of the 3rd USENIX conference on Offensive technologies. USENIX Association, 2009.
[24] ScyllaHide. https://bitbucket.org/NtQuery/scyllahide
[25] StrongOD. https://tuts4you.com/e107_plugins/download/download.php?view.2028
[26] Ke Sun and Xiaoning Li. “Break out of the Truman show: Active detection and escape of dynamic binary instrumentation”. Black Hat Asia 2016.
[27] Bardin, Sébastien, Robin David, and Jean-Yves Marion. "Backward-Bounded DSE: Targeting Infeasibility Questions on Obfuscated Codes." Security and Privacy (SP), 2017 IEEE Symposium on. IEEE, 2017.
[28] Rolles, Rolf. "Unpacking virtualization obfuscators." 3rd USENIX Workshop on Offensive Technologies. (WOOT). 2009.
[29] Kalysch, Anatoli, Johannes Götzfried, and Tilo Müller. "VMAttack: Deobfuscating Virtualization-Based Packed Binaries." Proceedings of the 12th International Conference on Availability, Reliability and Security. ACM, 2017.
[30] Coogan, Kevin, Gen Lu, and Saumya Debray. "Deobfuscation of virtualization-obfuscated software: a semantics-based approach." Proceedings of the 18th ACM conference on Computer and communications security. ACM, 2011.
[31] Aschermann, Tim Blazytko Moritz Contag Cornelius, and Thorsten Holz. "Syntia: Synthesizing the Semantics of Obfuscated Code." (2017).

약 력

최 석 우

1998 KAIST 전산학과 졸업(학사)
2000 KAIST 전산학과 졸업(석사)
2009 KAIST 전산학과 졸업(박사)
2009～2010 KT 네트워크 연구소 연구원
2010~ 국가보안기술연구소 선임연구원

관심분야 : 난독화 코드 분석, 바이너리 코드 검색
Email : seogu.choi@gmail.com