코드 주입을 통한 OpenSSL 공유 라이브러리의 보안 취약점 공격

(Attacking OpenSSL Shared Library Using Code Injection)

안 우 현 † 김 형 수 ††
(Woo Hyun Ahn) (Hyungsu Kim)

요 약 OpenSSL은 보안 통신 프로토콜인 SSL을 구현한 공개 소스 기반의 라이브러리이다. 하지만, 이 라이브러리는 리눅스 혹은 유닉스 운영체제에서 공유 라이브러리 형식으로 사용될 때 보안 정보를 쉽게 노출할 수 있다는 취약점이 있다. 본 논문은 이런 취약점을 공격하는 기법을 제안한다. 이 기법은 실행 중인 클라이언트 프로그램에 공격 코드를 주입하여 SSL 핸드셰이크 단계에서 보안 취약점을 다음과 같이 공격한다. 첫째, 클라이언트가 서버에게 지원 가능한 암호 알고리즘의 목록을 전송할 때 그 목록의 모든 알고리즘을 임의로 지정한 알고리즘으로 교체한다. 이 교체는 암호 알고리즘의 목록을 수신한 서버로 하여금 지정한 암호 알고리즘을 선택하도록 한다. 둘째, 암복호화에 사용되는 암호 키를 생성 과정에서 가로채고, 그 암호 키를 외부 공격자에게 전송한다. 그 후 외부 공격자는 지정한 암호 알고리즘과 가로챈 암호 키를 사용하여 송수신된 암호 데이터를 복호화한다. 제안하는 기법의 실현성을 보이기 위해 본 논문은 리눅스에서 OpenSSL 공유 라이브러리를 사용하는 ftp 클라이언트가 서버로 전송하는 암호화된 로그인(login) 정보를 가로채 복호화하는 실험을 수행하였다.

키워드 : 시스템 보안, Secure Socket Layer, OpenSSL, 공유 라이브러리, 코드 주입

Abstract OpenSSL is an open-source library implementing SSL that is a secure communication protocol. However, the library has a severe vulnerability that its security information can be easily exposed to malicious software when the library is used in a form of shared library on Linux and UNIX operating systems. We propose a scheme to attack the vulnerability of the OpenSSL library. The scheme injects codes into a running client program to execute the following attacks on the vulnerability in a SSL handshake. First, when a client sends a server a list of cryptographic algorithms that the client is willing to support, our scheme replaces all algorithms in the list with a specific algorithm. Such a replacement causes the server to select the specific algorithm. Second, the scheme steals a key for data encryption and decryption when the key is generated. Then the key is sent to an outside attacker. After that, the outside attacker decrypts encrypted data that has been transmitted between the client and the server, using the specified algorithm and the key. To show that our scheme is realizable, we perform an experiment of collecting encrypted login data that an ftp client using the OpenSSL shared library sends its server and then decrypting the login data.

Key words : system security, Secure Socket Layer, OpenSSL, shared library, code injection

† 정 회 원 : 광운대학교 컴퓨터소프트웨어학과 교수
whahn@kw.ac.kr
†† 정 회 원 : 지앤비영어전문교육 소프트웨어팀 연구원
zero1140@gmail.com
논문접수 : 2010년 5월 7일
심사완료 : 2010년 6월 12일

정보과학회논문지: 시스템 및 이론 제37권 제4호(2010.8)

1. 서 론

현재 인터넷은 정보 검색부터 뉴스, 쇼핑, 주식 등에 이르기까지 생활 전반에 걸쳐 필수적인 요소로 되었다. 이런 인터넷 서비스를 이용할 때 필요한 정보들 가운데는 개인의 사생활이나 기업의 이익과 관련이 있는 보안 정보도 있다. 하지만 최근에 컴퓨터 바이러스와 웜(worm)이 개인, 기업용 컴퓨터 시스템에 침투하여 보안 정보를 불법적으로 수집되는 사례가 많이 발생하고 있다. 더욱 심각한 상황은 이런 정보가 범죄에 악용되어

개인의 사생활이 노출되거나 기업의 이익에 큰 위협이 되고 있는 실정이다. 따라서 인터넷을 구성하는 네트워크와 컴퓨터 시스템이 다루는 보안 정보를 보호하는 일은 이제 인터넷 서비스에 있어서 중요한 요소가 되었다.

인터넷에서 데이터의 보안 통신을 위한 프로토콜로 Netscape에서 개발된 Secure Socket Layer(SSL)[1]가 가장 보편적으로 사용되고 있다. SSL은 네트워크 계층과 어플리케이션 계층의 중간에 위치한 암호 프로토콜이며, 인터넷에서 일어나는 트랜잭션의 기밀을 보장하기 위해 기밀성, 무결성 및 인증 기능을 제공한다. SSL 버전 3은 국제 표준 보안 프로토콜로 표준화되었고, 이 프로토콜은 Transport Layer Secure(TLS)로 명명되었다. SSL 버전 2, 3과 TLS 버전 1을 구현한 대표적인 암호화 라이브러리로서 공개 소스 기반의 OpenSSL[2]이 있다. 이 라이브러리는 아파치 웹서버, BIND DNS 서버 등의 인터넷과 웹 보안 응용 프로그램에 널리 사용되고 있다.

기존에 OpenSSL의 보안 취약점을 공격하는 다양한 기법들이 제안되었다. OpenSSL에 구현된 RSA 공개키 암호시스템[3]의 취약점을 공격하여 복호화에 사용되는 비밀 키(private key)를 획득하는 기법들[4,5]이 소개되었다. 특히 Brumley[4]는 RSA 알고리즘의 복호화 시간이 비밀 키의 값에 따라 다르다는 사실을 발견하고, 이 복호화 시간을 측정하여 비밀 키를 유추하였다. 이런 취약점을 보완한 RSA 알고리즘의 코드가 제안되었지만, 이 코드가 하드웨어 결함이 주입된 CPU에서 실행되면 비밀 키가 추출될 수 있는 취약점이 최근에 발견되었다[5]. 아울러 OpenSSL을 사용하는 서버에 버퍼 넘침(buffer overflow)[6]을 발생시킨 후에 분산 서비스 거부(DDoS)를 일으키는 코드와 같은 악성 코드를 주입하는 기법들[7,8]이 제안되었다. 다행히 이런 버퍼 넘침을 이용한 공격들은 OpenSSL에 대한 지속적인 업데이트를 통해 무력화되었다.

기존 공격 기법은 OpenSSL의 구현 코드의 보안 취약점을 이용한 반면에, 리눅스 또는 유닉스 계열의 운영체제에서 OpenSSL를 실행할 때 노출되는 보안 취약점을 공격하지는 않았다. 본 논문에서 발견한 이러한 취약점은 다음과 같다. 첫째 OpenSSL이 공개 소스이기 때문에 보안 정보를 다루는 함수의 인터페이스와 코드의 동작이 노출되어 있다. 예를 들어 데이터의 암복호화에 사용되는 암호 키를 생성하는 함수의 구현을 소스 코드로 쉽게 확인할 수 있다. 둘째 응용 프로그램은 OpenSSL을 정적 라이브러리뿐만 아니라 공유 라이브러리 형식으로 사용한다. 실행 중인 프로그램이 공유 라이브러리를 적재하는 과정이 공개되어 있어서 그 프로그램이 사용하는 메모리에서 보안 정보를 다루는 함수의 위치를 탐색하여 접근할 수 있다. 셋째 ptrace 함수[9]를 사용하여 공격 대상 프로세스의 메모리에 저장된 코드 및 데이터를 쉽게 접근할 수 있다. 공격자가 이 함수를 사용하면 대상 프로세스에 악성 코드를 주입하거나 메모리에 적재된 코드를 변경하여 악의적인 코드를 실행하도록 할 수 있다.

본 논문은 OpenSSL 공유 라이브러리를 적재하여 실행하는 클라이언트 프로그램에서 보안 정보를 획득하고, 그 정보를 통해 송수신되는 암호 데이터를 복호화하는 기법을 제안한다. 이 기법을 코드 주입 기반의 OpenSSL 라이브러리 공격 기법(code injection-based openssl library attack, COLA)이라고 한다. 이 기법을 수행하는 공격 프로그램은 웜이나 바이러스 형태로, 관리자(root) 권한이 아닌 사용자 권한으로 설치되어 실행된다고 가정한다. 이 기법은 앞에서 소개된 보안 취약점을 이용하여 실행 중인 클라이언트 프로그램에 공격 코드를 주입하고, 클라이언트 프로그램과 서버 프로그램 간의 SSL 핸드셰이크(handshake) 단계에서 다음과 같은 공격을 가한다. 첫째, 대상 클라이언트에서 서버로 전달되는 cipher suite 목록을 임의로 지정한 목록으로 대체한다. 둘째 데이터의 암복호화에 사용되는 암호 키가 생성될 때 곧바로 가로채서 외부 공격자에게 전송한다. 마지막으로 외부 공격자는 클라이언트와 서버 간에 송수신되는 암호 데이터를 수집하고, 공격 프로그램이 지정한 cipher suite와 가로챈 암호 키를 사용하여 수집된 암호 데이터를 복호화한다.

컴퓨터 시스템에 공격 프로그램이 사용자 권한으로 설치되고 실행되었다는 전제하에, 사용자 또는 보안 정보를 획득하거나 변경하기 위해 공격 프로그램이 수행할 수 있는 공격 기법이 제안되었다. Baliga[10]는 e-mail 암호화, SSL 등 대부분 보안 프로그램에서 사용되는 난수 생성기(PRNG)의 기능을 무력화하거나 firewall의 기능을 무력화하는 기법을 제안하였다. 리눅스 기반의 스마트폰에서 사용자의 정보를 획득하는 기법[11]도 소개되었다. 하지만, 이들 기법들은 커널의 코드와 데이터를 접근해야하기 때문에 구현 복잡도가 매우 높다. 또한 이들 기법은 커널을 접근하기 위해 공격 시 관리자 권한을 획득해야하나, 관리자 권한의 획득을 막는 최근의 리눅스 환경[12]에서 공격을 수행할 수가 없다. Windows 운영체제에서 난수 생성기의 구현 취약점을 이용하여 관리자 권한 없이도 난수 값을 획득하는 공격 기법[13]이 최근에 제안되었다. 하지만 이 기법은 계산 복잡도가 큰 문제점이 있다. 또한 SSL 통신에서 송수신되는 암호 데이터를 복호화할 목적으로 이 기법을 사용하여 난수를 획득하더라도 이 난수만으로는 복호화에 필요한 암호 키를 유추할 수 없다.

본 논문에서 소개된 보안 취약점과 COLA 기법을 검증하기 위해 OpenSSL의 공유 라이브러리로 최근에 배포된 OpenSSL 0.9.8k, 운영체제로 리눅스 커널 2.6을 탑재한 Fedora 8[14]를 사용하는 클라이언트 시스템을 구축하였다. 이 시스템에서 OpenSSL 0.9.8k를 공유 라이브러리로 사용하는 ftp 클라이언트 프로그램 lftp[15]를 공격하여 서버와 송수신되는 암호 데이터를 복호화하였다. 아울러 바이러스 검사 프로그램을 실행한 결과, 바이러스 또는 공격 프로그램이 없다는 진단을 확인하였다.

본 논문의 구성은 다음과 같다. 2장에서는 배경 지식을 설명한다. 3장에서는 OpenSSL의 취약점을 이용한 공격기법에 대해 구체적으로 설명하며, 4장에서는 실험 결과를 설명한다. 5장에서는 관련 연구를 설명하고, 6장에서는 결론과 COLA 공격에 대한 대응책을 기술한다.

2. 배경 지식

2.1 OpenSSL

SSL[1]은 클라이언트와 서버 간의 암호화 통신을 위해 사용되는 보안 프로토콜이다. SSL 프로토콜에서 클라이언트와 서버 간에 데이터를 송수신하기 이전에 서로 보안 정보를 공유하기 위해 핸드셰이크 과정이 수행된다. 클라이언트와 서버는 ClientHello와 ServerHello 메시지를 사용하여 각자의 SSL 버전, 섹션 번호 및 난수(random number)를 교환한다. 특히 클라이언트는 키 교환, 블록 암호화 및 해시(hash) 알고리즘으로 구성된 cipher suite의 목록을 ClientHello 메시지에 포함하여 서버로 전송한다. 서버는 이 목록에서 한 개의 cipher suite를 선택하여 그 결과를 ServerHello 메시지를 통해 클라이언트로 전송한다. 그 후 클라이언트는 pre-master secret를 생성하고, 이를 서버로 전달하여 서로 공유한다. 클라이언트와 서버는 각자 pre-master secret를 사용하여 master secret와 키 블록(key block)을 순차적으로 생성한다. 키 블록은 48 바이트의 크기이며, 송수신하는 데이터의 암호화와 복호화를 위한 암호 키들을 포함한다. 마지막으로 Finished 메시지를 서로 수신한 후에 핸드셰이크를 종료한다.

OpenSSL은 SSL 프로토콜을 구현한 공개 소스 기반의 라이브러리이며, 응용 프로그램에게 정적 또는 공유 라이브러리 형식으로 제공된다. 특히 공유 라이브러리일 경우 파일 이름은 libssl.so이다. OpenSSL을 사용하는 클라이언트가 ClientHello 메시지를 서버로 보내거나 키 블록을 생성하기 위해 그림 1과 같은 함수들을 사용한다. ssl3_client_hello 함수는 암호화 통신에 사용할 cipher suite 목록을 함수 인자인 crypto_data로 받은 후 서버로 전달한다. ssl3_setup_key_block 함수는 클라

```
int ssl3_client_hello(SSL *crypto_data)
int ssl3_setup_key_block(SSL *crypto_data)
```

그림 1 OpenSSL 주요 함수

이언트 난수, 서버 난수, master secret 등을 포함하는, 함수 인자인 crypto_data를 받아서 키 블록을 생성하고, 키 블록을 다시 그 함수 인자에 저장한다.

하지만, OpenSSL이 공개 소스이기 때문에 위 두 함수와 같이 보안 정보를 다루는 모든 함수의 인터페이스와 내부 동작이 쉽게 파악된다. 또한 함수 인자에 저장된 보안 정보가 암호화되지 않아서 쉽게 획득 및 변경될 수 있다는 취약점이 있다. 예를 들어 ssl3_client_hello 함수가 함수 인자에 포함된 cipher suite 목록을 서버로 전송하기에 앞서, 실행 중인 악의적인 프로그램이 그 목록을 임의의 cipher suite 목록으로 교체한다면 서버는 이 교체된 목록에서 한 개의 cipher suite를 선택할 수밖에 없다. 또한 ssl3_setup_key_block 함수에서 생성된 키 블록이 함수 인자에 저장된 직후에 악의적인 프로그램이 그 인자로부터 키 블록을 가로챈다면, 이 키 블록과 ssl3_client_hello 함수에서 교체된 cipher suite를 사용하여 송수신되는 암호 데이터를 쉽게 복호화할 수 있다.

2.2 공유 라이브러리

공유 라이브러리는 응용 프로그램의 실행 중에 메모리로 적재되어 링크될 수 있는 라이브러리이며, 이런 링크 과정을 동적 링킹(*dynamic linking*)이라고 한다. 공유 라이브러리가 한 번 적재되면 실행 중인 다른 프로그램들이 각자 이 라이브러리를 적재하지 않고 공유할 수 있기 때문에 메모리 공간이 절약된다. 리눅스 및 UNIX 계열의 운영체제에서 공유 라이브러리와 실행 파일은 ELF(Executable Linkable Format) 파일 형식[16]을 가진다. 아울러 ELF 파일 형식과 그 적재 과정은 이미 공개되어 있다.

공유 라이브러리에 있는 각 함수가 실행 중인 프로그램에서 처음 호출되면, 동적 링커(dynamic linker)는 함수의 이름(심벌)을 사용하여 메모리에서 함수의 주소를 탐색한 후에 이 주소부터 함수를 실행시킨다. 이름으로 함수의 주소를 탐색하는 동작을 심벌 해석(symbol resolution)이라고 하며, 컴파일 시점이 아닌 실행 중인 프로그램이 함수를 호출하는 시점에서 심벌 해석을 수행하는 동작을 지연 바인딩(lazy binding)이라고 한다. 동적 링커는 각 실행 파일과 공유 라이브러리 파일에 포함된 전역 오프셋 테이블(*Global Offset Table, GOT*)과 프로시저 연결 테이블(Procedure Linkage Table, PLT)을 사용하여 지연 바인딩을 수행한다.

GOT의 첫 번째 테이블 엔트리(entry) GOT[0]은 동적 섹션(dynamic section)의 주소, 두 번째 엔트리 GOT[1]은 동적 링크될 공유 라이브러리들의 정보, 세 번째 엔트리 GOT[2]는 지연 바인딩을 수행하는 함수의 주소를 가진다. 네 번째 엔트리 GOT[3]부터는 각 함수의 메모리 주소를 가진다. PLT는 호출되는 함수로 실행 흐름을 이동시키는 기능을 수행한다. PLT의 첫 번째 테이블 엔트리 PLT[0]은 GOT[2]에 포함된 지연 바인딩 함수를 호출하는 코드를 갖는다. 두 번째 엔트리 PLT[1]부터는 각 함수에 할당된 주소를 GOT에서 획득하고 그 함수를 실행하는 코드를 가진다. 아울러 PLT[0]을 제외한 모든 PLT 엔트리는 지연 바인딩 함수를 호출하는 PLT[0]으로 분기(branch)하는 코드를 추가로 포함한다. 이는 임의의 함수가 처음 호출될 때 지연 바인딩을 사용하여 이 함수의 주소를 획득하기 위해서이다.

컴파일 시점에 프로그램이 호출하는, 공유 라이브러리의 함수에 고유 번호가 부여되며, 각 함수마다 PLT, GOT에 엔트리가 한 개씩 할당된다. 함수 번호가 n인 함수를 위해 PLT[n+1]과 GOT[n+3]의 엔트리가 할당된다. 그림 2는 실행 중인 프로그램이 공유 라이브러리 libc에 포함된 printf 함수를 호출하는 과정을 나타낸다. 이 함수의 번호는 0이라고 가정한다. 그림 2(a)에서 printf 함수가 처음 호출되면 PLT[1]의 코드가 실행된다(①). 이 코드는 GOT[3]에 저장된 함수의 주소가 유효한지를 검사한다(②). 이 함수가 처음 호출되었기 때문에 GOT[3]은 유효한 주소를 포함하지 않는다. 따라서 PLT[1]의 코드는 PLT[0]의 코드를 호출하여 지연 바인딩 함수를 실행한다(③). 이 함수의 심벌 해석 동작으로 획득된 함수 주소 0x804864는 GOT[3]에 저장되고, 이 주소부터 함수의 실행이 시작된다. 그림 2(b)처럼 printf 함수가 다시 호출되면 PLT[1]의 코드는 지연 바인딩을 거치지 않고 곧바로 GOT[3]에 포함된 함수 주소에서 그 함수를 실행한다.

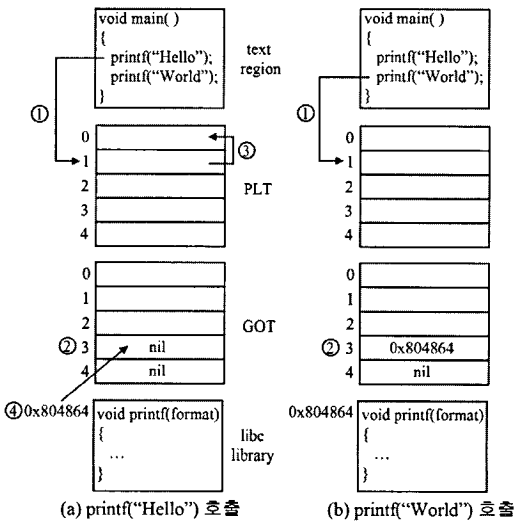


그림 2 지연 바인딩 과정

하지만 지연 바인딩 함수가 수행하는 심벌 해석 동작은 악의적인 프로그램이 OpenSSL과 같이 보안 정보를 다루는 공유 라이브러리에 있는 함수의 주소를 획득하기 위해 악용될 수 있다는 문제점이 있다. 이것은 공개된 ELF 파일 형식을 가지는 실행 파일과 공유 라이브러리 파일에 포함된 PLT와 GOT을 쉽게 접근할 수 있기 때문이다. 따라서 악의적인 프로그램은 보안 정보를 다루는 함수의 주소를 탐색하여 그 함수의 동작을 바꾸거나 원하는 정보를 가로챌 수 있다. 앞으로 본 논문에서는 공유 라이브러리를 라이브러리로 언급한다.

2.3 ptrace 시스템 함수

ptrace[9] 함수는 리눅스와 기존 유닉스 기반의 운영체제에서 지원하는 시스템 함수(system call)로 한 프로세스가 다른 프로세스의 실행을 관찰하거나 제어하는 수단을 제공한다. ptrace는 주로 프로세스의 디버깅에 사용되며, 그 함수 형식은 그림 3과 같다.

```
ptrace(enum request, pid_t pid, void* addr, void* data)
```

그림 3 ptrace 함수

ptrace 함수는 4개의 인자를 가진다. request는 함수의 동작을 지정하고, pid는 대상 프로세스의 아이디이다. addr는 접근할 대상 메모리의 위치를 나타내며, data는 대상 메모리로부터 읽을 데이터를 저장할 메모리 공간 또는 대상 메모리 주소로 저장할 데이터를 포함하는 메모리 공간의 주소를 나타낸다. 한 프로세스가 대상 프로세스와 동일한 사용자 권한을 가지면 관리자 권한 없이도 대상 프로세스의 메모리에 있는 코드 및 데이터를 읽거나 변경할 수 있다.

실행 중인 악의적인 프로그램이 ptrace 함수를 사용하면 공격 대상 프로세스의 메모리에 있는 보안 정보를 가로채거나 프로그램의 실행 흐름을 변경할 수 있는 문제점이 있다. 하지만 이런 문제점에도 불구하고 개인용 컴퓨터에 탑재된 리눅스는 소프트웨어 개발에 필요한 디버깅 기능, 실시간 소프트웨어 패치[17] 및 메모리 오류(memory corruption) 탐지[18] 등을 위해 ptrace 함수를 지속적으로 지원하고 있다.

3. OpenSSL 공유 라이브러리 공격 기법

3.1 기본 개념

본 논문은 OpenSSL 라이브러리 기반의 클라이언트

프로그램을 실행하는 프로세스를 공격하여 보안 정보를 획득하거나 변경하는 COLA 기법을 제안한다. 이 기법을 수행하는 공격 프로그램은 Intel CPU 기반의 리눅스를 탑재한 클라이언트 시스템에 웜이나 바이러스 형태로 주입되어 실행된다고 가정한다. 이 프로그램을 실행하는 공격 프로세스는 다음과 같은 시스템의 보안 취약점을 이용한다. 첫째, ptrace 함수를 사용하여 공격 대상 프로세스의 메모리에 저장된 코드와 보안 정보를 접근할 수 있다. 둘째, OpenSSL에서 보안 정보를 다루는 함수의 인터페이스와 구현이 공개되었다. 셋째, OpenSSL 라이브러리가 ELF 파일 형식이여서 대상 프로세스에 적재된 후 그 라이브러리에 있는 함수의 주소가 쉽게 탐색될 수 있다. 이런 취약점을 이용하여 공격 프로세스는 공격 코드를 가지는 공유 라이브러리를 대상 프로세스에 주입한다. 이 공격 코드는 SSL 핸드셰이크 단계에서 대상 프로세스가 서버로 전송하는 cipher suite 목록에 포함된 모든 cipher suite를 지정된 cipher suite로 교체한다. 이 공격을 cipher suite 교체라고 칭한다. 또한 대상 프로세스가 생성한 키 블록을 가로채서 외부 공격자에게 전송한다. 외부 공격자는 키 블록과 지정된 cipher suite를 사용하여 송수신되는 암호 데이터를 복호화한다.

그림 4는 COLA을 수행하는 공격 프로세스가 대상 프로세스에 공격 라이브러리를 적재하고 그 내부에 있는 공격 코드를 실행하는 기본 동작을 보여준다. 그림 4(a)처럼 공격 전에 대상 프로세스의 메모리 공간에 GOT, 코드(text), libssl이 적재되며, GOT[1]에 libssl 라이브러리의 정보가 포함된다고 가정한다. COLA는 libssl에 있는 다음 두 개의 함수를 공격할 대상 함수로 지정한다. 첫 번째 대상 함수인 ssl3_client_hello는 SSL 버전, 섹션 번호, 난수와 cipher suite 목록을 포함한 보안 정보를 ClientHello 메시지에 넣어 서버로 전송한다. 두 번째 대상 함수인 ssl3_setup_key_block은 클라이언트와 서버 측의 보안 정보를 사용하여 키 블록을 생성한다.

그림 4(b)처럼 공격 프로세스가 대상 프로세스에 공격 코드를 주입하여 대상 함수를 공격하는 과정은 다음과 같은 세 단계로 구성된다. 첫째, 대상 프로세스에 있는 GOT의 두 번째 엔트리에 저장된 라이브러리 정보를 사용하여 이미 적재된 libssl를 검색하고, 그 내부에

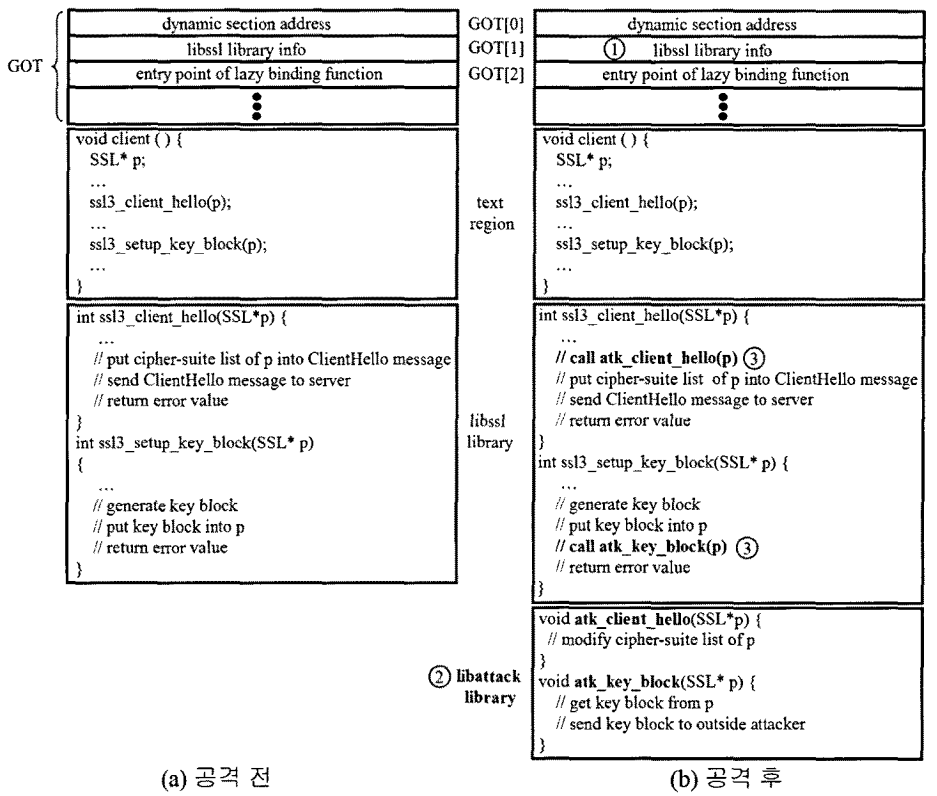


그림 4 COLA 기법의 기본 공격 과정

서 대상 함수의 주소를 획득한다(①). 둘째, 대상 프로세스의 메모리에 공격 함수를 포함하는 공격 라이브러리인 libattack을 적재시키고, 그 내부에서 공격 함수의 주소를 획득한다(②). 셋째, 대상 함수의 코드 일부분을 수정하여 공격 함수를 호출하도록 한다(③). 이 공격 함수들 중에 atk_client_hello는 ssl3_client_hello에서 호출되어 cipher suite 교체를 수행하며, 또 다른 공격 함수인 atk_key_block은 ssl3_setup_key_block에서 호출되어 대상 프로세스가 생성하는 키 블록을 가로채서 외부 공격자로 전달하는 역할을 담당한다. 본 논문은 공격 코드의 주입에 필요한 세 단계의 과정을 구현하기 위해 기존에 소개된 기법을 응용하거나 새로운 기법을 제안한다. 이 기법들을 다음 절부터 설명한다.

공격 프로그램은 관리자 권한이 아닌 사용자 권한으로 COLA 기법을 실행한다. 이런 사용자 권한 기반의 공격 기법은 기존 기법들[10,11,13]에 비해 다음과 같은 장점을 가진다. 첫째, 기존 기법들은 공격하고자 하는 코드와 데이터를 접근하기 위해 커널 메모리를 스캐닝(scanning)하거나 하드웨어 이벤트인 인터럽트(interrupt)를 처리하는 코드를 접근해야하기 때문에 복잡도가 매우 크다. 반면에, COLA 기법은 응용 프로그램 수준에서 공격을 수행할 수 있기 때문에 그 구현 복잡도가 낮다. 둘째, COLA 기법은 관리자 권한의 획득을 막는 보안 프로그램[12]이 설치된 최근의 리눅스에서도 공격을 수행할 수가 있다.

3.2 공격 대상 함수의 검색

대상 프로세스의 메모리에 적재된 libssl에서 대상 함수의 주소를 탐색하는 기법을 구현하였으며, 이를 함수 탐색 기법이라고 한다. 이 기법의 동작은 기존 동적 링커의 지연 바인딩 함수가 수행하는 심벌 해석 동작과 유사하며, 다음과 같은 세 단계로 구성된다. 첫째, 대상 프로세스의 동적 섹션에 포함된 GOT를 탐색한다. 둘째, GOT에서 프로그램에 동적 링크된 라이브러리의 정보를 관리하는 GOT[1]을 접근하여 libssl의 정보를 획득한다. 셋째, libssl의 내부에 있는 대상 함수의 주소를 탐색한다.

GOT를 접근하기 위해 함수 탐색 기법은 우선 대상 프로세스의 ELF 헤더(header)를 접근한다. 리눅스는 실행 파일의 ELF 헤더를 각 프로세스의 메모리 공간에서 0x08048000 주소에 위치시킨다. 함수 탐색 기법은 대상 프로세스의 0x08048000에 위치한 ELF 헤더를 접근하고, 이 헤더에 저장된 프로그램 헤더 테이블(Program Header Table, PHT)의 주소를 사용하여 PHT를 접근한다. 실행 파일은 여러 개의 섹션으로 구성되며, PHT는 각 섹션에 대한 이름, 종류, 메모리 주소 등의 정보를 가지는 배열로 구성된다. 함수 탐색 기법은 이 배열을 검사하여 동적 섹션을 탐색하고, 동적 섹션에 있는 GOT의 메모리 주소를 획득한다.

탐색된 GOT의 두 번째 엔트리 GOT[1]을 사용하여 대상 함수를 포함하는 libssl 라이브러리의 정보를 획득한다. GOT[1]은 공유 라이브러리의 정보를 관리하는 연결 리스트(linked list)의 시작 주소를 저장한다. 이 리스트는 link_map 구조체의 객체들로 구성되며, 각 link_map은 프로그램에 동적 링크된 공유 라이브러리의 이름, 주소와 그 라이브러리가 가지는 동적 섹션의 주소 등의 정보를 포함한다. 결국 함수 탐색 기법은 GOT[1]이 가리키는 연결 리스트의 시작부터 libssl의 이름을 가지는 link_map 객체를 탐색하고, 이 객체에서 libssl의 메모리 주소와 동적 섹션의 주소 등의 정보를 획득한다.

획득된 libssl의 정보 중에 동적 섹션에 포함된 심벌 테이블(symbol table)과 문자열 테이블(string table)을 사용하여 대상 함수의 주소를 탐색한다. 문자열 테이블의 각 엔트리(string table entry, STRE로 칭함)는 함수 혹은 전역 변수의 이름을 저장한다. 심벌 테이블의 각 엔트리는 해당 라이브러리에 포함된 함수나 전역 변수의 정보를 포함하며, 이 정보에는 시작 주소, 심벌 종류 등이 있다. 아울러 심벌 테이블의 각 엔트리는 함수 혹은 전역 변수의 이름을 직접 저장하지 않고, 이 이름을 가지는 STRE를 가리키는 포인터를 가지고 있다. 함수 탐색 기법은 대상 함수의 시작 주소를 획득하기 위해 심벌 테이블의 엔트리가 가리키는 STRE에 저장된 함수 이름과 대상 함수의 이름을 순차적으로 비교한다. 대상 함수의 이름을 가지는 STRE를 가리키는 심벌 테이블 엔트리가 탐색되면 이 엔트리에서 대상 함수의 주소를 획득한다.

3.3 공격 라이브러리 적재

탐색된 대상 함수의 코드를 수정하여 그 내부에서 공격 함수가 호출되도록 하기 위해서는 우선 대상 프로세스에 공격 라이브러리를 적재되어야 한다. 이를 위해 기존에 연구된 기법[19]은 GNU C 라이브러리인 libc의 심벌 테이블에서 _dl_open 함수를 탐색하여 호출하였다. 하지만, 이 함수가 libc 2.7 버전부터 심벌 테이블에서 삭제되었기 때문에 이 기법을 더 이상 libc 2.8을 탑재한 최근 리눅스에서 사용할 수 없다.

본 논문은 최근 리눅스에서 공격 라이브러리를 프로세스에 적재하는 기법을 제안한다. 이 기법은 공유 라이브러리를 사용하는 프로세스에 일반적으로 적재된 libdl 라이브러리의 dlopen 함수를 사용한다. 이 기법의 구체적인 동작 과정은 다음과 같다. 첫째, 대상 프로세스의 메모리에서 dlopen의 주소를 탐색한다. 이 탐색을 위해 3.2절에 나타난 함수 탐색 기법이 사용된다. 둘째, 대상 프로세스의 실행 흐름을 변경하여 대상 프로세스에서 dlopen이 호출되도록 하며, 함수 인자로 공격 라이브러

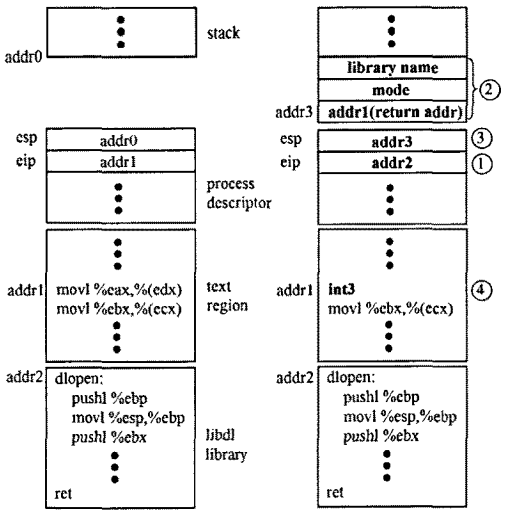


그림 5 공격 라이브러리의 적재 과정

리의 이름과 함수 동작을 지정하는 모드(mode)가 입력된다. 하지만 대상 프로세스는 정상적인 실행에서 공격 라이브러리를 적재하기 위해 dlopen을 의도적으로 호출하지 않는다. 따라서 이런 호출이 대상 프로세스에서 수행될 수 있도록 공격 프로세스는 대상 프로세스의 메모리 내용을 변경한다. 셋째, 대상 프로세스의 실행 흐름을 dlopen 호출 이전으로 복귀시킨다.

그림 5는 대상 프로세스가 스스로 dlopen을 호출하여 공격 라이브러리를 적재하도록 공격 프로세스가 대상 프로세스의 메모리 내용을 변경하는 과정을 보여준다. 그림 5(a)는 공격 전에 실행 중인 대상 프로세스의 메모리 공간을 보여준다. Intel x86 기반 CPU의 eip 레지스터(register)는 현재 명령어의 실행 후에 다음 실행할 명령어의 주소를 저장한다. 아울러 스택(stack)은 호출된 함수의 인자, 지역 변수 및 복귀 주소(return address) 값을 저장하는 메모리 공간이며, 스택의 최상부(top) 주소는 esp 레지스터에 저장된다. 프로세스 스케줄링 또는 디바이스 I/O 등의 이유로 프로세스가 일시적으로 정지하면 리눅스는 eip와 esp를 포함한 모든 레지스터들의 값을 커널 메모리에 있는 프로세스 디스크립터(process descriptor, 디스크립터로 칭함)[20]에 임시적으로 저장한다. 프로세스가 재실행되면 디스크립터에 저장된 레지스터의 값을 CPU에 다시 적재하여 정지된 다음 명령어부터 실행된다.

그림 5(b)는 공격 프로세스가 일시적으로 정지된 대상 프로세스의 메모리 내용을 변경하는 다섯 단계의 동작을 나타낸다. 첫째, 대상 프로세스의 디스크립터에 있는 eip 값을 dlopen 함수의 주소인 addr2로 변경하고, 원본 eip 값인 addr1을 임시적으로 백업한다(①). 이 원본 eip 값은 대상 프로세스가 dlopen을 실행 후에 실행 흐름이 이 함수를 호출한 명령어의 다음으로 복귀하기 위해 사용된다. 둘째, dlopen으로 전달될 두 개의 인자, 즉 공격 라이브러리 이름인 libattack과 이 함수 동작을 지정하는 모드 값인 RTLD_LAZY를 스택의 최상부에 순차적으로 저장한다(②). 아울러 대상 프로세스가 이 함수를 실행한 후에 복귀할 주소를 추가로 스택에 저장한다. 이 복귀 주소는 단계 ①에서 백업된 원본 eip 값인 addr1로 설정된다. 셋째, 스택의 크기가 증가하였기 때문에 대상 프로세스의 디스크립터에 포함된 esp 값을 복귀 주소가 저장된 메모리 위치로 변경한다(③). 이 esp 값은 dlopen이 실행될 때 필요한 함수 인자와 복귀 주소를 스택에서 접근하기 위해 사용된다. 아울러 원본 esp 값을 임시적으로 백업한다. 넷째, 단계 ①에서 백업된 원본 eip 값이 지정하는 주소 addr1에 있는 명령어를 트랩 인터럽트(trap interrupt)를 발생시키는 int3 명령어로 교체하고, 교체된 원본 명령어를 백업한다(④). 이 인터럽트는 dlopen의 호출이 완료되는 시점을 공격 프로세스에게 알리기 위해 사용된다. 다섯째, 공격 프로세스는 대상 프로세스에서 트랩 인터럽트가 발생할 때까지 정지한다.

그림 5(b)와 같이 변경된 메모리 내용을 가지는 대상 프로세스가 실행되면 우선 디스크립터에 저장된 eip와 esp 값은 CPU의 레지스터로 적재된다. eip가 지정하는 dlopen의 주소인 addr2에 있는 명령어부터 실행되며, 이 함수는 esp가 지정하는 스택 최상부 주소 addr3을 사용하여 실행 시 필요한 함수 인자를 접근한다. 대상 프로세스는 dlopen의 실행을 완료 후에 복귀 주소가 지정하는 주소인 addr1로 복귀하고, 이 복귀 주소에 있는 int3 명령어를 실행하여 트랩 인터럽트를 발생시킨다. 이 트랩 인터럽트의 발생은 일시적으로 정지된 공격 프로세스를 재실행시킨다. 재실행된 공격 프로세스는 단계 ①, ③에서 백업한 원본 eip, esp 값과 단계 ④에서 int3 명령어로 교체된 원본 명령어를 대상 프로세스의 해당 메모리 위치로 저장하여 대상 프로세스의 메모리 상태를 그림 5(a)로 복원시킨다. 이 후에 대상 프로세스가 실행되면 이 프로세스는 메모리 상태가 변경되기 전의 실행 흐름으로 복귀한다.

대상 프로세스의 디스크립터에 포함된 eip과 esp 값을 변경하거나 임의의 메모리 주소에 int3 명령어를 삽입하기 위해 기존에 소개된 기법[21]을 사용하였다. 이 기법은 user_regs_struct 구조체를 입력 인자로 ptrace 함수를 호출하여 대상 프로세스의 디스크립터에서 모든 레지스터들의 값을 읽고, 이 구조체의 eip와 esp 멤버를 통해 해당 레지스터들의 값을 획득한다. user_regs_

struct의 eip와 esp 멤버를 임의의 값으로 설정한 후에 이 구조체를 입력 인자로 ptrace를 호출하면 대상 프로세스의 디스크립터에 있는 해당 레지스터들의 값이 설정된 값으로 변경된다. 또한 메모리에 적재된 코드에 int3 명령어를 삽입하기 위해 이 명령어의 opcode 값(0xcc)과 삽입할 메모리 주소를 입력 인자로 ptrace를 호출한다.

3.4 대상 함수의 공격 함수 호출

메모리에 적재된 대상 함수 내에서 공격 함수가 호출되도록 대상 함수의 코드 일부분을 실행 시간에 수정한다. 이런 수정을 위해 우회 패치(detour patch)[22] 기법을 사용한다. 이 기법은 바이러스 검사 프로그램에 의해 쉽게 탐지되지 않는 가장 진보된 코드 해킹 기법이다. 이 기법은 메모리에 적재된 대상 함수 내의 특정 명령어를 call 및 jmp와 같은 분기 명령어로 교체하여 공격 함수를 호출하는 우회 함수로 분기하도록 한다. 우회 함수의 실행이 완료되면 실행 흐름이 대상 함수에 삽입된 분기 명령어의 다음으로 복귀한다. 하지만 대상 함수에서 교체된 원본 명령어의 실행 없이 분기 명령어의 다음부터 실행되면 비정상적인 동작이 발생할 수 있다. 이런 문제를 막기 위해 교체된 원본 명령어는 우회 함수에 포함되며, 우회 함수에서 공격 함수의 실행 후에 대상 함수로 복귀하기 이전에 실행된다.

그림 6은 대상 함수 ssl3_client_hello가 우회 패치 후에 공격 함수를 호출하는 과정을 보여준다. 그림 6(a)는 우회 패치 전의 대상 함수의 코드를 나타내며, 5 바이트의 시작 코드는 이 함수의 스택 프레임(stack frame)을 할당하는 초기화 코드이다. 그림 6(b)는 우회 패치 후의 대상 함수와 우회 함수 det_client_hello를 나타낸다. 우회 함수는 어셈블리 언어로 구현되어 공격 라이브러리에 포함된다. 대상 함수가 함수 인자에 저장된 cipher suite 목록을 서버로 전송하기 전에 이 목록을 교체하는 공격이 수행되어야 서버로 하여금 교체된 cipher suite 목록 중에서 하나를 선택하도록 할 수 있다. 이를 위해 대상 함수의 시작 부분에 있는 5 바이트의 원본 코드를 우회 함수로 실행 흐름을 이동시키는 분기 코드 "call det_addr1"로 교체하고, 교체된 원본 코드는 우회 함수의 끝부분에 포함된다. 우회 함수의 주소 det_addr1은 대상 프로세스에 적재된 공격 라이브러리에서 3.2절에 나타난 함수 탐색 기법을 통해 획득된다.

그림 6(b)에서 분기 코드가 호출하는 우회 함수의 코드는 세 부분으로 구성된다. 첫째, 우회 함수가 사용할 스택 프레임을 할당하고 그 내부에 대상 함수의 인자를 저장한다(①). 둘째, cipher suite 교체를 수행하는 공격 함수 atk_client_hello를 실행한다(②). 이 함수는 우회 함수의 스택 프레임에 있는, 대상 함수의 인자를 접근하

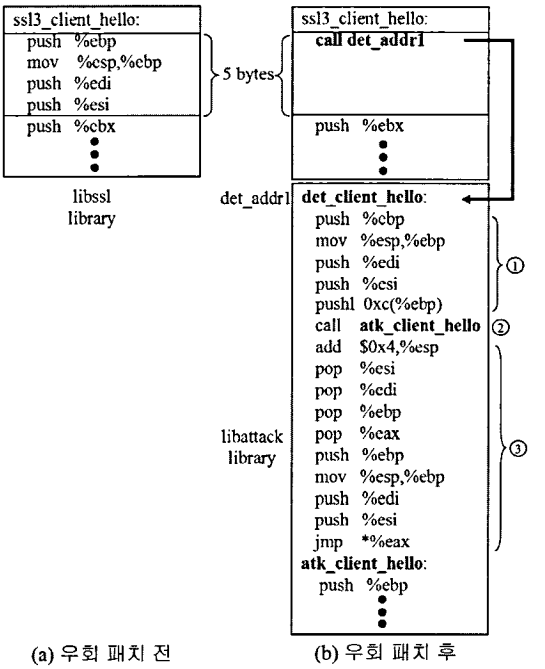


그림 6 ssl3_client_hello에 대한 우회 패치

여 cipher suite 목록의 모든 cipher suite를 임의로 선정한 cipher suite로 교체한다. cipher suite는 키 교환, 블록 암호화 및 해시 알고리즘으로 구성되지만, 공격 함수는 데이터의 암복호화에 사용되는 블록 암호화와 해시 알고리즘들만 각각 DES_CBC와 SHA로 변경한다. 셋째, 우회 함수에 할당된 스택 프레임을 제거하고, 대상 함수에서 분기 코드로 교체된 원본 코드를 실행한다(③). 이 원본 코드는 실행 흐름이 대상 함수로 복귀하기 이전에 대상 함수가 사용할 스택 프레임을 할당한다. 그 후에 우회 함수의 실행 흐름에서 복귀한 대상 함수는 함수 인자에 포함된, 교체된 cipher suite 목록을 서버로 전송한다.

그림 7은 대상 함수 ssl3_setup_key_block이 우회 패치 후에 공격 함수를 호출하는 과정을 보여준다. 그림 7(a)는 우회 패치 전의 대상 함수의 코드이다. 이 함수는 키 블록을 생성하여 함수 인자에 저장한다. 아울러 끝 부분에 있는 5 바이트를 실행하여 대상 함수에 할당된 스택 프레임을 제거하고 오류 코드를 반환한다. 그림 7(b)는 우회 패치 후의 대상 함수와 우회 함수 det_key_block을 나타낸다. 대상 함수에서 키 블록의 생성 후에 실행되는 5 바이트의 오류 반환 코드를 우회 함수를 호출하는 분기 코드 "jmp det_addr2"로 교체하고, 교체된 원본 코드는 대신 우회 함수의 끝부분에 포함된다. 이 우회 함수가 실행되면 먼저 대상 함수의 인자를

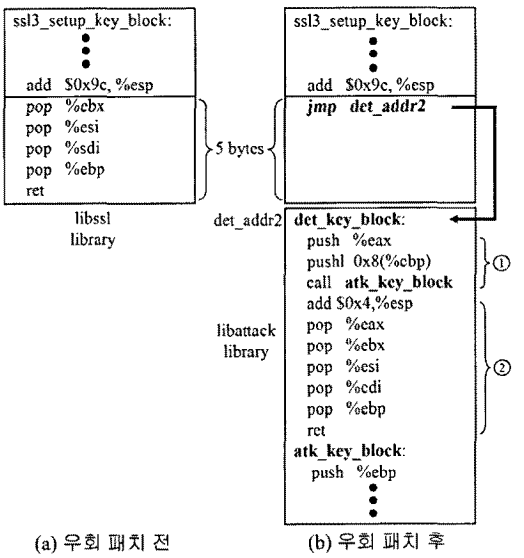


그림 7 ssl3_setup_key_block에 대한 우회 패치

입력으로 공격 함수인 atk_key_block이 호출된다(①). 이 함수는 입력된 인자로부터 키 블록을 가로채고, 이를 외부 공격자로 전송한다. 그 다음에 우회 함수는 분기 코드로 교체된 대상 함수의 원본 코드를 실행한다(②).

4. 실 험

4.1 실험 환경

본 논문은 COLA 기법에 대한 검증을 위해 3개의 컴퓨터 시스템, 즉 클라이언트, 서버 및 외부 공격자 시스템으로 이루어진 실험 환경을 구성하였다. 이 실험 환경에서 클라이언트와 서버 시스템은 OpenSSL 공유 라이브러리를 사용하는 ftp 클라이언트와 서버 프로그램을 각각 실행한다. 클라이언트 시스템에서 COLA 기법을 수행하는 공격 프로그램은 관리자 권한이 아닌 ftp 클라이언트 프로그램을 실행하는 사용자의 권한(또는 계정)으로 설치되어 실행된다. 공격 프로그램은 ftp 클라이언트 프로그램의 실행이 시작될 때 공격 라이브러리를 클라이언트에 적재하고 공격 코드를 대상 함수에 주입한다. 공격 시스템은 공격 프로그램이 획득한 키 블록을 수신하고, 클라이언트와 서버 간에 송수신되는 암호 데이터를 수집한다. 아울러 수신한 키 블록을 사용하여 수집된 암호 데이터를 복호화한다.

표 1은 각 시스템의 구성, ftp 클라이언트와 서버 프로그램, OpenSSL 공유 라이브러리의 버전 정보를 나타낸다. OpenSSL 공유 라이브러리를 동적 링크로 실행하는 ftp 클라이언트 프로그램으로 lftp[11], ftp 서버 프로그램으로 vsftpd[23]를 사용하였다. 공격 시스템은 wireshark[24] 프로그램을 사용하여 서버와 클라이언트가 송수신하는 패킷(packet)을 수집한다. 또한 수집된 패킷에서 암호 데이터를 선별하기 위해 ssldump[25] 프로그램을 사용한다. 클라이언트, 서버 및 공격 시스템은 최근에 배포된 OpenSSL 0.9.8k 공유 라이브러리를 사용하였다.

표 1 실험 환경

구성 \ 시스템	클라이언트	서버	공격
CPU (Intel)	Core2Duo 2.6GHz	Pentium4 2.1GHz	Core2Duo 2.6GHz
메모리	DDR2 1GB	DDR 512MB	DDR2 4GB
운영체제	Fedora 8 리눅스	Fedora 8 리눅스	Fedora 8 리눅스
프로그램	lftp 4.0.3	vsftpd 2.2.1	wireshark 1.0, ssldump
OpenSSL	OpenSSL 0.9.8k		

4.2 실험 결과

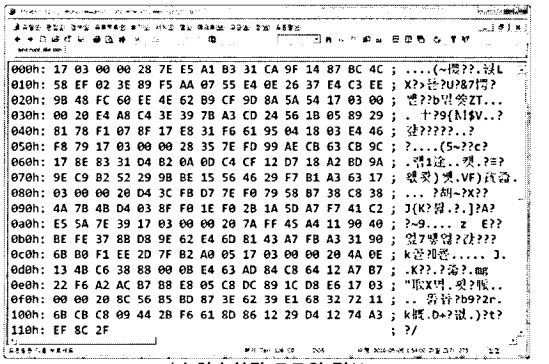
COLA 기법의 공격 성공 여부를 확인하기 위해 ftp 클라이언트가 서버로 전달하는 암호화된 로그인(login) 정보를 복호화하는 실험을 수행하였다. 이 복호화 과정은 공격 시스템에서 수행되며, 공격 시스템은 ftp 클라이언트에서 서버로 전송되는 모든 패킷을 수집한다. 공격 시스템은 클라이언트 시스템의 공격 프로그램이 ftp 클라이언트를 공격하는 과정에서 지정한 cipher suite와 가로챈 키 블록을 사용하여 수집된 패킷으로부터 로그인 정보를 복호화한다.

공격 시스템은 암호 데이터를 복호화하기에 앞서, wireshark 프로그램을 실행하여 클라이언트와 서버 간에 송수신되는 모든 패킷들을 파일로 수집하고, 수집된 패킷 중에서 암호화된 로그인 정보를 포함하는 패킷을 선별한다. 이런 선별을 위해 SSL 버전 3과 TLS 프로토콜로 송수신되는 패킷을 분석하는 프로그램인 ssldump를 사용하였다. 이 프로그램은 송수신된 패킷을 핸드셰이크, 암호 데이터 전송, 암호 규격 변경(change cipher spec) 동작 등의 트랜잭션 단위로 분류한다.

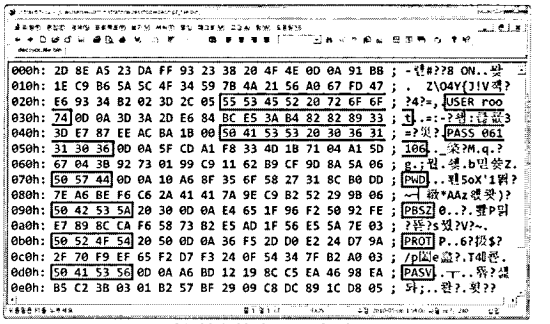
그림 8은 수집된 패킷들을 저장하는 파일을 입력으로 ssldump 프로그램을 실행한 결과를 나타낸다. 패킷 종류에서 "Application Data"는 암호 데이터를 포함하는 패킷을 나타내며, 그 외의 "Handshake"는 핸드셰이크 단계에서 전송되는 패킷, "Change Cipher Spec"은 암호 규격 변경 단계에서 전송되는 패킷, "ClientHello"는 클라이언트에서 서버로 ClientHello 메시지를 전송하는 단계의 패킷을 의미한다. ftp 클라이언트는 사용자가 첫 번째 명령어로서 서버와의 연결을 요청하면 핸드셰이크 과정을 거친 후에 암호화된 로그인 정보를 서버에 전송

패킷 번호	패킷 전송 방향	패킷 종류
1	Client -> Server	Client Hello
2	Server -> Client	Handshake
3	Server -> Client	Handshake
4	Server -> Client	Handshake
5	Client -> Server	Handshake
6	Client -> Server	Handshake
7	Client -> Server	Change Cipher Spec
8	Client -> Server	Handshake
9	Server -> Client	Change Cipher Spec
10	Server -> Client	Handshake
11	Client -> Server	Application Data
12	Server -> Client	Application Data
13	Client -> Server	Application Data
14	Server -> Client	Application Data
15	Client -> Server	Application Data
16	Server -> Client	Application Data
17	Client -> Server	Application Data
18	Server -> Client	Application Data
19	Client -> Server	Application Data
20	Server -> Client	Application Data
21	Client -> Server	Application Data
22	Server -> Client	Application Data
23	Client -> Server	Application Data
24	Server -> Client	Application Data
25	Client -> Server	Application Data
26	Server -> Client	Application Data
27	Client -> Server	Client Hello
28	Server -> Client	Handshake
⋮	⋮	⋮

그림 8 패킷 분석 결과



(a) 암호화된 로그인 정보



(b) 복호화된 로그인 정보

그림 9 암호 데이터의 복호화 결과

한다. 이다음부터 사용자가 입력하는 ftp 명령어를 실행할 때마다 서버와의 핸드세이크 과정을 재실행한다. 따라서 로그인 정보는 첫 번째 핸드세이크의 다음부터 수집된 11번부터 26번까지의 패킷들에 포함되며, 이들 중에서도 서버로 전송되는 패킷들에 포함된다. 이 로그인 정보를 포함하는 패킷들로부터 암호 데이터를 추출하기 위해 wireshark 프로그램의 메뉴에서 "Follow TCP Stream"을 실행하였다. 이 추출된 암호 데이터는 복호화에 앞서 미리 임시 파일에 저장된다.

추출된 암호화된 데이터를 복호화하여 로그인 정보를 획득하기 위해 복호화 프로그램을 구현하였다. 이 구현을 위해 OpenSSL이 제공하는 EVP 라이브러리[2]를 사용하였다. EVP 라이브러리는 주어진 임의의 데이터를 암호화하거나 암호 데이터를 복호화할 때 필요한 암호 함수들을 제공한다. 복호화 프로그램은 블록 암호화 알고리즘과 해시 알고리즘으로서 클라이언트 시스템에서 공격 프로그램이 지정한 DES_CBC와 SHA을 각각 사용하며, 복호화 알고리즘의 키로서 공격 과정에서 획득한 키 블록에 포함된 암호 키를 사용한다. 복호화 프로그램의 실행으로 획득한 로그인 정보를 포함한 데이터를 임시 파일에 저장한다.

그림 9는 공격 시스템의 복호화 프로그램이 로그인 정보를 포함하는 암호 데이터를 복호화한 결과를 보여준다. 그림 9(a)는 wireshark을 통해 수집된 암호 데이터를 저장하는 파일을 보여주며, 이 파일의 데이터를 복호화하여 획득한 로그인 정보를 저장한 파일이 그림 9(b)에 나타난다. 로그인 정보를 가지는 파일을 통해 로그인 과정에서 입력된 사용자 아이디와 패스워드가 각각 "root"와 "061106"임을 확인할 수 있다. 아울러 ftp 클라이언트가 로그인 과정에서 사용자가 입력하지 않은 pwd, pbsz, prot, pasv 명령어도 함께 전송하였음을 확인할 수 있다. 결국 이러한 로그인 정보의 복호화는 COLA 기법이 리눅스에서 OpenSSL 공유 라이브러리의 취약점을 성공적으로 공격하였음을 보여준다.

클라이언트 시스템의 공격 프로그램이 바이러스 검사 프로그램을 통해 탐지되는지를 실험하였다. 이 실험에서 사용된 바이러스 검사 프로그램은 avast linux home edition[26]과 klamav[27]이며, 이 프로그램 모두 2009년 11월 10일자 악성 코드 데이터베이스를 사용하였다. 실험 결과, 두 프로그램 모두 클라이언트 시스템에 바이러스 또는 공격 프로그램이 없다는 진단을 내렸다.

5. 관련 연구

OpenSSL의 보안 취약점을 공격한 여러 방법이 제안되었다. 이러한 연구는 크게 RSA 공개 키 암호시스템을 공격하는 기법과 버퍼 넘침[6]을 통해 악성 코드를

주입하는 기법으로 나눌 수 있다. 한편, 리눅스 또는 유닉스 운영체제에서는 프로세스에 악성 코드를 주입하거나 커널의 보안 취약점을 공격하는 기법들이 제안되었다. 또한 컴퓨터 시스템에서 사용자 또는 보안 정보를 획득하거나 변경하기 위해, 바이러스 및 웜의 형태로 설치되어 실행되는 공격 프로그램이 수행할 수 있는 공격 기법들이 소개되었다.

OpenSSL에 구현된 RSA 공개 키 암호시스템의 보안 취약점을 공격하는 기법들이 소개되었다. Brumley[4]는 시간차 공격(timing attack)[28]을 사용하여 OpenSSL을 탑재한 서버의 비밀 키를 획득하는 기법을 제안하였다. 이 기법은 RSA 복호화에 걸리는 시간이 비밀 키의 값에 따라 다르다는 사실을 발견하였고, 키 교환 과정에서 복호화 시간을 측정하여 서버가 사용하는 비밀 키를 유추한다. 이 취약점은 RSA 알고리즘의 코드를 보완함으로써 해결되었다. 하지만 이 보완된 코드를 실행하는 CPU에 전압(voltage)의 크기를 조절하여 하드웨어 결함을 주입하면 비밀 키가 추출될 수 있는 취약점이 최근에 발견되었다[5]. 또한 OpenSSL 0.9.8c에서 RSA 비밀 키를 특정 범위내의 값으로 설정하는 문제점이 있었다[29]. 만일 공격자가 이 범위의 값을 비밀 키로 선택하여 복호화하는 작업을 반복적으로 시도한다면 비밀 키를 쉽게 발견할 수 있다. 다행히 이 취약점은 지속적인 업그레이드를 통해 보완되었다.

서버에서 실행되는 OpenSSL의 코드에 버퍼 넘침을 발생시켜 악성 코드를 주입하는 기법들이 제안되었다. 악의적인 클라이언트는 서버에서 버퍼 넘침이 발생하도록 서버와의 핸드셰이크 과정에서 비정상적인 키 또는 매우 큰 번호의 섹션 ID를 전송한다[7,8]. 버퍼 넘침이 일어난 서버의 코드에 DDoS와 같은 공격 코드를 원격으로 주입하기도 하였다. 버퍼 넘침은 strcpy와 같은 문자열 처리 함수가 버퍼의 공간에 데이터를 저장할 때 그 공간을 넘어서 주변의 다른 메모리 공간을 덮어쓰는 현상이다. 만일 버퍼의 공간을 넘어서 저장되는 데이터가 스택에 저장된, 함수의 복귀 주소를 덮어쓰고 새로운 복귀 주소가 악성 코드가 있는 메모리 주소를 지시한다면, 현재 함수가 실행된 후에 악성 코드가 실행된다. OpenSSL에서 버퍼 넘침에 취약한 코드는 지속적인 패치로 보완되었다. 아울러 최근에 GNU gcc 컴파일러에 적용된, 버퍼 넘침 탐지 기법[30,31]을 활성화하여 OpenSSL 라이브러리를 생성한다면 이 라이브러리에서 발생하는 버퍼 넘침을 막을 수 있다.

리눅스 또는 유닉스 운영체제의 프로세스가 악성 코드를 호출하게 하는 공유 라이브러리 호출 재지정(shared library call redirection) 기법[32]이 소개되었다. 이 기법은 오프라인으로 실행 파일에 악성 코드를 주입하고, 이 파일을 실행하는 프로세스가 공유 라이브러리의 특정 대상 함수를 호출하면 이 함수 대신 주입된 코드를 호출하도록 한다. 이 호출을 위해 대상 함수에 할당된 GOT 엔트리의 함수 주소를 주입된 코드의 주소로 실행 시간에 변경한다. 이 GOT 변경 방식은 Windows 운영체제의 프로세스에 코드를 주입하는 IAT 후킹(hooking) 기법[22]과 유사하다. 다행히 IAT 후킹을 탐지하는 기법[22]이 리눅스에 적용된다면 공유 라이브러리 호출 재지정 기법도 쉽게 탐지될 수 있다.

실시간 프로세스 감염(runtime process infection) 기법[19]은 리눅스에서 실행 중인 프로세스에 공유 라이브러리를 적재하고 그 내부의 코드를 실행시킨다. 이 기법은 라이브러리를 적재하기 위해 libc 라이브러리에 포함된 심벌 테이블에서 _dl_open을 탐색하여 호출한다. 적재된 라이브러리의 함수가 프로세스에서 호출될 수 있도록 공유 라이브러리 호출 재지정 기법에 나타난 GOT 변경 방식을 사용한다. 하지만, 이 기법을 막기 위해 2007년 10월에 나온 libc 2.7버전부터 심벌 테이블에서 _dl_open이 삭제되었다. 하지만 본 논문에서 제안된 공유 라이브러리 적재 기법은 _dl_open 함수 대신 dlopen 함수를 호출한다.

리눅스 또는 유닉스 운영체제에서 커널의 보안 취약점을 공격하는 기법들도 연구되었다[33,34]. 이 기법은 실시간에 적재 가능한 커널 모듈(Loadable Kernel Module, LKM)[35]을 사용하여 시스템 함수의 호출을 가로챈다. 예를 들어, read, write, close와 같은 시스템 함수가 호출될 때 악성 코드를 실행하여 커널이 관리하는 자원을 접근한다. 또한 LKM의 도움 없이 커널의 메모리에 저장된 코드나 데이터를 접근하는 기법[36]도 소개되었다. 이런 기법들과 달리 본 논문은 커널 수준의 공격 기법이 아니라 응용 프로그램 수준의 공격 기법을 제안하였다.

리눅스에 바이러스 또는 웜과 같은 악성 코드를 설치하는 방법들이 다음과 같이 소개되었다. 첫째, 사용자로 하여금 e-mail에 첨부된 프로그램을 실행하여 악성 코드를 설치하도록 한다. 둘째, 최근에는 스크린세이버(screen saver) 프로그램에 악성 코드를 탑재하고, 그 프로그램을 사용자들이 많이 방문하는 웹페이지(gnome-look.org)에 올려놓았다[37]. 사용자가 그 프로그램을 다운로드하여 설치하면 악성 코드가 리눅스에 쉽게 설치될 수 있다. 셋째, 리눅스, 윈도우 및 유닉스의 문서 작성용 프로그램인 OpenOffice로 작성된 문서를 통해 웜 코드가 설치된 사례도 있다. 이 웜은 Badbunny[38]라고 하며, 감염된 문서에 포함된 매크로(macro)가 실행될 때 웜이 생성되어 설치된다.

리눅스 및 Windows 운영체제 기반의 컴퓨터 시스템

에 공격 프로그램이 사용자 권한으로 설치되고 실행되었다고 전제하에, 중요한 사용자 정보 또는 보안 정보를 획득하거나 변경하는 공격 기법들이 소개되었다. Baliga [10]는 e-mail 암호화, SSL 등 대부분 보안 프로그램에서 사용되는 난수 생성기(PRNG)가 항상 0의 난수 값을 생성하도록 난수 생성에 필요한 정보를 변경하거나 firewall에 전달되는 패킷을 변경하여 그 기능을 무력화하는 기법을 제안하였다. 리눅스 기반의 스마트폰에서 사용자의 정보를 획득하는 기법[11]도 소개되었다. 이 기법은 특정 시스템 함수가 호출될 때 통화 내용을 가로채거나 사용자의 위치 정보를 획득하였다. 하지만, 이들 리눅스에서의 공격은 커널의 코드와 데이터를 접근해야하기 때문에 구현 복잡도가 매우 높다. 또한 이런 커널 접근을 위해 이 기법은 공격 시 사용자 권한에서 관리자 권한으로 권한 상승(privilege escalation)을 수행해야한다. 하지만, 이 기법은 권한 상승을 막는 보안 프로그램[12]이 설치된 최근의 리눅스에서 공격을 할 수가 없다.

최근에는 Windows 운영체제에서 난수 생성기의 구현 취약점을 이용하여 관리자 권한 없이도 난수 값을 획득하는 공격 기법[13]이 제안되었다. 이 기법은 버퍼 넘침이나 ptrace와 유사한 ReadMemory 함수를 사용하여 대상 프로세스의 스택에 저장된 난수 생성 정보로 획득하고, 이를 이용하여 다음에 생성될 난수 값을 예측한다. 하지만 이 기법은 계산 복잡도가 큰 문제점이 있다. 또한 공격 프로그램이 SSL 통신에서 송수신되는 암호 데이터를 복호화하기 위해 이 기법을 사용하여 난수 값을 획득하더라도 이 값만으로는 암호 키를 생성할 수 없다. 반면에 본 논문은 암호화된 데이터를 복호화할 때 필요한 암호 키를 획득할 수 있다.

6. 결론 및 대응책

본 논문은 리눅스에서 응용 프로그램이 공개 소스 기반의 OpenSSL 공유 라이브러리를 사용할 때 악성 프로그램에 의해서 보안 정보가 노출될 수 있다는 취약점을 발견하였다. 이런 보안 취약점을 이용하여 실행 시간에 보안 정보를 접근하는 COLA 기법을 제안하였다. 이 기법은 실행 중인 클라이언트 프로그램에 공격 코드를 포함하는 공격 라이브러리를 적재하였다. 이 공격 코드는 클라이언트와 서버 프로그램 간의 핸드셰이크 단계에서 cipher suite 목록을 변경하고 데이터의 암복호화에 사용되는 암호 키를 포함하는 키 블록을 가로챘다. ftp 프로그램을 대상으로 한 실험에서 COLA 기법은 클라이언트 프로그램에서 서버로 전송하는 암호 데이터를 복호화하는 데 성공하였다. 이 결과를 통해 리눅스 계열의 운영체제와 OpenSSL 모두 본 논문에서 제안된 취약점을 그동안 간과하고 있었으며 이런 취약점에 대한 대응책의 마련이 시급하다는 것을 확인할 수 있다.

COLA 기법에 노출된 보안 취약점에 대한 대응책으로 다음과 같은 방법들을 고려할 필요가 있다. 첫째, 응용 프로그램이 ptrace를 악의적으로 사용하지 못하도록 제어하는 방법이 리눅스 내에서 지원되어야 한다. 둘째, 응용 프로그램이 SSL 공유 라이브러리와 같은 특정 보안 라이브러리의 코드와 데이터를 변경할 때 이런 변경을 바이러스 검사 프로그램이 탐지하거나 차단해야 한다. 셋째, 프로그램의 가독성을 떨어뜨리기 위해 난독화(obfuscation)를 사용한다. 넷째, 개발자는 OpenSSL 기반의 응용 프로그램을 개발할 때 비록 소프트웨어 관리의 불편함이 있더라도 공유 라이브러리 대신 정적 라이브러리를 사용하여 응용 프로그램의 실행 파일을 생성한다.

참 고 문 헌

[1] S. A. Thomas, *SSL and TLS Essentials: Securing the Web*, John Wiley & Sons, 2000.

[2] OpenSSL: The Open Source Toolkit for SSL/TLS. Web Site: http://www.openssl.org.

[3] R. Rivest, A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public-key Cryptosystems," *Communications of the ACM*, vol.21, pp.120-126, February 1978.

[4] D. Brumley and D. Boneh, "Remote Timing Attacks are Practical," In *Proc. of the 12th USENIX Security Symposium*, pp.1-14, August 2003.

[5] A. Pellegrini, V. Bertacco, and T. Austin, "Fault-Based Attack of RSA Authentication," In *Proc. of the Conference on Design Automation and Test in Europe (DATE)*, March 2010.

[6] N. P. Smith, "Stack Smashing Vulnerabilities in the UNIX Operating System," http://destroy.net/machines/security/nate-buffer.pdf, 1997.

[7] Cert Vulnerability Note VU#102795, "OpenSSL Servers Contain a Buffer Overflow during the SSL2 Handshake Process," http://www.kb.cert.org/vuls/id/102795.

[8] Cert Vulnerability Note VU#561275, "OpenSSL Servers Contain a Remotely Exploitable Buffer Overflow Vulnerability during the SSL3 Handshake Process, http://www.kb.cert.org/vuls/id/561275.

[9] ptrace(2) - Linux man page. Web site: http://linux.die.net/man/2/ptrace.

[10] A. Baliga, P. Kamat, and L. Iftode, "Lurking in the Shadows: Identifying Systemic Threats to Kernel Data," In *Proc. of the 2007 IEEE Symposium on Security and Privacy*, pp.246-251, May 2007.

[11] B. Jeffrey, R. O'Hare, A. Baliga, Arati, V. Ganapathy, and L. Iftode, "Rootkits on smart phones:

attacks, implications and opportunities," In *Proc. of the 11th ACM HotMobile*, pp.49-54, February 2010.
[12] Ninja - Privilege escalation detection system for GNU/Linux, http://www.ubuntugeek.com/ninja-privilege-escalation-detection-system-for-gnulinux.html.
[13] L. Dorrendorf, Z. Gutterman, and B. Pinkas, "Cryptanalysis of the random number generator of the Windows operating system," *ACM Transactions on Information and System Security*, vol.13, no.1, pp.1-32, 2009.
[14] Fedora home page. Web Site: http://fedoraproject.org.
[15] lftp program home page. Web Site: http://lftp.yar.ru or http://en.wikipedia.org/wiki/Lftp
[16] J. R. Levine, *Linkers and Loaders*, Morgan Kaufmann, 2000.
[17] G. Altekar, I. Bagrak, P. Burstein, and A. Schultz. "OPUS: Online Patches and Updates for Security," In *Proc. of the 14th USENIX Security Symposium*, pp.287-302, August 2005.
[18] J. Xu, P. Ning, C. Kil, Y. Zhai, and C. Bookholt, "Automatic Diagnosis and Response to Memory Corruption Vulnerabilities," In *Proc. of the 12th ACM Conference on Computer and Communications Security*, pp.223-234, October 2007.
[19] "Runtime Process Infection," *Phrack Magazine*, vol.0x0b, no.0x3b, July 2002.
[20] R. Love, *Linux Kernel Development*, 2nd Ed., Novell, 2005.
[21] P. Padala, "Playing with ptrace, Part III," *Linux Journal*, vol.2002 no.104, p.5, December 2002.
[22] G. Hoglund and J. Butler, *Rootkits: Subverting the Windows Kernel*, Addison-Wesley, 2005.
[23] vsftpd program home page. Web Site: http://vsftpd.beasts.org.
[24] wireshark program home page. Web Site: http://-www.wireshark.org.
[25] ssldump program home page. Web Site: http://www.rtfm.com/ssldump.
[26] avast anti-virus home page. Web Site: http://www.avast.com/eng/avast-for-linux-work station.html
[27] klamav anti-virus home page. Web Site: http://klamav.sourceforge.net/klamavwiki/index.php/Main_Page.
[28] P. Kocher, "Timing Attacks on Implementations of Diffie-hellman, RSA, DSS, and Other Systems," *Advances in Cryptology*, pp.104-113, 1996.
[29] Debian OpenSSL Predictable PRNG Toys. Web Site: http://metasploit.com/users/hdm/tools/debian-openssl.
[30] C. Cowan, C. Pu, D. Maier, H. Hinton, P. Bakke, S. Beattie, A. Grier, P. Wagle, and Q. Zhang, "StackGuard: Automatic Adaptive Detection and Prevention of Buffer-Overflow Attacks," In *Proc. of the 7th USENIX Security Conference*, pp.63-78, January 1998.
[31] StackShield home page. Website: http://www.angelfire.com/sk/stackshield.
[32] S. Cesare, "Shared Library Call Redirection via ELF PLT Infection," *Phrack Magazine*, vol.0x0a, no.0x38, May 2000.
[33] A. Chuvakin, "An Overview of UNIX Rootkits," *iALERT White Paper*, iDefense Labs, http://www.megasecurity.org/papers/Rootkits.pdf, February 2003.
[34] Plaguez, "Weakening the Linux Kernel," *Phrack Magazine*, vol.8, no.52, January 1998.
[35] K. Jones, "Loadable Kernel Modules," *USENIX login: Magazine*, http://www.usenix.org/publications/login/2001-11/pdfs/jones2.pdf, November 2001.
[36] Sd and Devik, "Linux On-The-Fly Kernel Patching without LKM," *Phrack Magazine*, vol.0x0b, no.0x3a, December 2001.
[37] Linux malware: an incident and some solutions. Web site: https://lwn.net/Articles/367874.
[38] Badbunny (computer worm). Web site: http://en.wikipedia.org/wiki/Badbunny.

안 우 현
정보과학회논문지 : 시스템 및 이론
제 37 권 제 3 호 참조

김 형 수
2010년 광운대학교 컴퓨터소프트웨어학과(학사). 2010년~현재 (주)지앤비영어전문교육 소프트웨어팀 연구원. 관심분야는 시스템보안, 임베디드시스템