

하트블리드 탐지 및 대응방안

장영진*, 한승우**, 허준범***
 고려대학교 컴퓨터학과
 issuxark11@korea.ac.kr*
 gkstmddn87@korea.ac.kr**
 jbhur@korea.ac.kr***

Heartbleed Detection and Countermeasure

Youngjin Jang, Seung-Woo Han, Junbeom Hur
 Dept. of Computer Science and Engineering, Korea University

요약

하트블리드는 OpenSSL 라이브러리 상의 심각한 취약점이며, 이 결함으로 인해 SSL/TLS encryption 을 사용해 보호되는 서버의 암호화 키나 사용자들의 민감한 정보들을 공격자가 탈취하는 것이 가능해졌다. 본 논문에서는 하트블리드 취약점에 대해 알아보고 탐지 및 대응방안을 소개한다.

1. 서론

2014년 4월 인터넷 상에서 정보 전송 시 보안을 위해 사용하는 OpenSSL에서의 취약점인 하트블리드(Heartbleed)가 공개 되었고 최근 발견된 보안 결함 중 가장 심각한 수준이라는 평가 받았다. 하트블리드 취약점으로 인해 OpenSSL을 사용하는 전 세계 많은 웹사이트들이 보안적으로 취약한 상태에 노출되었고 이로 인해 해커들은 취약한 버전의 OpenSSL을 사용하는 서버에 하트블리드 공격을 시도함으로써 서버 내 메모리상의 암호화 키, 사용자 이름, 민감한 개인 정보 등을 탈취하는 것이 가능해졌다. 캐나다의 국세청 홈페이지에서 900여 개의 사회보장번호가 유출되는 등 캐나다와 영국에서 피해사례가 발표되었지만 정보가 유출되어도 로그가 남지 않기 때문에 정확한 피해 여부를 알 수 없어 더 큰 문제가 되었다. 하트블리드 취약점이 보고되고 나서 곧바로 패치된 OpenSSL 버전이 배포되었지만 OpenSSL이 널리 사용되는 오픈 소스이고 취약성 코드는 2012년 3월 OpenSSL 1.0.1 버전 출시 이후 오랜 기간 동안 하트블리드 취약점에 노출되어 있었기 때문에 피해 규모가 더 클 것으로 예상된다[2].

보안상의 취약점이 발견된 후에 해야 할 일은 해당 취약점을 해결하는 것과 어떻게 하면 취약점을 발견할 수 있는지를 알아내는 것이다. 따라서 본 논문에서는 심각한 보안 위협을 불러일으킨 하트블리드 취약점을 조기에 발견하지 못한 원인에 대해 분석하였다. 또한 하트블리드 취약점을 효과적으로 발견하고 대응하기 위한 방법들에 대해 조사하였다.

2. OpenSSL 과 Heartbleed Bug

OpenSSL은 Secure Sockets Layer(SSL)와 Transport

Layer Security(TLS) 프로토콜을 구현한 오픈 소스 라이브러리이다. SSL/TLS는 인터넷을 통해 주고받는 정보들을 안전하게 보호하기 위한 프로토콜이다. SSL/TLS는 항상 서버에 대한 인증과정을 거치고 옵션에 따라 클라이언트에 대한 인증도 수행할 수 있다. 또한 서버와 클라이언트간 전송되는 모든 정보는 암호화되며 암호화된 연결을 확립하기 위해 연결 수립 단계와 정보전송단계를 거친다[1]. 그림.1 에서는 OpenSSL을 사용하는 서버와 클라이언트간 통신 절차를 보여준다.

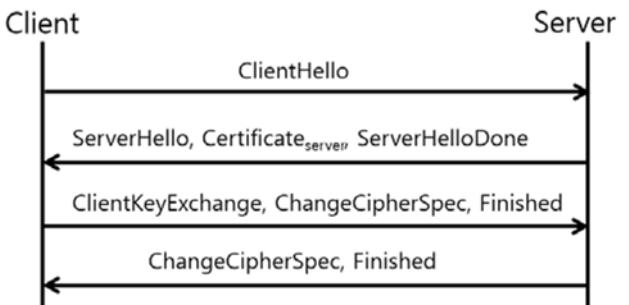


그림. 1 SSL Handshake 프로토콜 동작 순서

하트블리드 취약점은 OpenSSL의 Transport Layer Security(TLS) / Datagram Transport Layer Security (DTLS) Heartbeat Extension(RFC6520)의 구현상에 존재하는 취약점이다[1]. Heartbeat는 OpenSSL에서 사용되는 확장규격으로서 서버와 클라이언트 간에 매번 재연결을 하지 않아도 안정적이고 효율적인 연결을 유지하기 위해 도입되었다. 클라이언트는 Heartbeat Request를 서버에 보내 임의의 정보를 요청하며 Heartbeat Request Module은 타입, 페이로드 길이, 페이로드, 패딩으로 구성되어 있다. 서버는 해당 요청에 대해 응답하는 과정에서 요청한 정보를 보내며 현재 연결이

지속적으로 이루어지고 있다는 사실을 클라이언트에게 알려준다. 하지만 전달받은 요청에 대해 요청 받은 정보와 그 정보의 길이가 일치하는지 확인을 하지 않아 버퍼오버리드에 취약해진다. Heartbeat Request를 통해 요청할 수 있는 정보의 크기는 최대 64KB로 공격자가 Heartbeat Request를 반복적으로 보내는 과정에서 서버 내 메모리상에 올라가 있는 사용자 정보, 비밀키 등 보안상 민감한 정보들이 유출될 수 있다. 서버는 Heartbeat Request 패킷을 통해 받은 페이로드 길이를 인증 과정 없이 신뢰해서는 안되지만 실제 레코드의 길이를 체크하지 않아 버퍼오버리드를 야기하는 취약점을 가지게 되었다[4].

TLS Heartbeat Request를 다루는 코드는 다음과 같다.

```
/* Read type and payload length first */
hbtype = *p++;
n2s(p, payload);
pl = p;
```

그림. 2 취약한 OpenSSL 코드1

Fig. 2는 요청 받은 Heartbeat Request Message를 처리하는 과정을 담고 있다. 포인터 *p*는 Heartbeat 메시지의 처음을 가리킨다. *n2s()*는 16비트 크기의 페이로드 길이를 *payload* 변수에 할당하고 포인터를 2바이트만큼 증가시킨다. 따라서 포인터 *pl*은 Heartbeat Request의 페이로드에 담긴 데이터를 가리키게 된다. *payload* 변수는 공격자가 보낸 Heartbeat Request 패킷으로부터 읽어 들이므로 버퍼는 최대 64KB로 공격자가 의도한 크기만큼 할당된다. 아래의 코드에 의해 응답 메시지가 생성된다.

```
/* Enter response type, length and copy payload */
*bp++ = TLS1_HB_RESPONSE;
s2n(payload, bp);
memcpy(bp, pl, payload);
```

그림. 3 취약한 OpenSSL 코드2

Fig. 3은 Heartbeat Response 메시지를 구성하는 과정을 담고 있다. 포인터 *bp*는 Heartbeat Response 메시지를 생성하기 위한 버퍼포인터의 시작을 가리킨다. 먼저 메시지 타입을 쓰고 1바이트 만큼 *bp*를 증가시킨다. *s2n()*은 16비트의 페이로드 길이를 메모리에 쓰고 2바이트 만큼 *bp*를 증가시킨다. 이후 *memcpy*를 이용하여 *payload* 변수의 크기만큼을 *pl*로부터 읽어들여 *bp*에 복사하고 이는 heartbeat response 패킷의 페이로드가 된다. 앞에서 *payload* 변수는 공격자가 의도에 의해 정해지므로 최대 64KB 만큼의 데이터가 서버의 메모리 상에서 새어나갈 수 있게 된다.

하트블리드 취약점은 실제 페이로드의 길이가 충분히 긴지 또는 페이로드가 0인지를 체크함으로서 해결할 수 있다. Hearbeat Request 메시지는 SSL3 structure를 통해 오며 *s3->rrec.length*는 실제 레코드 길이를 의미한다. 그림. 4는 업데이트된 OpenSSL에서 하트블리드 취약점을 해결하기 위해 사용한 코드이다[6].

리드 취약점을 해결하기 위해 사용한 코드이다[6].

```
if(1 + 2 + 16 > s->s3->rrec.length)
    return 0; /* silently discard */
hbtype = *p++;
n2s(p, payload);
if(1 + 2 + payload + 16 > s->s3->rrec.length)
    return 0; /* silently discard per RFC 6520 sec. 4 */
pl = p;
```

그림. 4 수정된 OpenSSL 코드

3. 기존의 취약점 탐지 방법과 하트블리드

기존의 취약점 탐지 방법에는 정적 분석과 동적 분석 크게 2 가지 방법이 있다. 3 장에서는 기존의 취약점 탐지 방법에 대해 알아보고 해당 방법들로 하트블리드 취약점을 왜 초기에 발견되지 못했는지 살펴본다. 그리고 어떤 방법으로 하트블리드 취약점을 탐지하고 대응할 수 있는지 소개한다.

3.1 정적 분석 방법

정적 분석 방법은 프로그램을 실행시키지 않고 코드 레벨에서의 취약점을 찾는다. OpenSSL 코드는 정적 분석 툴이 취약점을 발견하기에 너무 복잡했고 기존의 툴들의 탐지 범위를 벗어났다. 정적분석 툴들은 현재까지 발견되지 않은 형태의 취약점들에 대해서는 해결할 수 없으며 새로운 휴리스틱을 적용해야만 유사한 취약점들을 발견할 수 있다[5].

3.2 동적 분석 방법

동적 분석 방법은 프로그램에 특정 입력 값을 주고 실행시키고 취약점을 찾는 방식이다. 이 방법이 취약점을 탐지하기 위해서는 해당 입력 값이 프로그램에 부적절한 실행을 일으켜야 한다. 퍼지(fuzzy) 테스팅은 임의의 난수 값을 생성해 프로그램에 입력하고 실행 시 의도하지 않은 일이 발생하는지를 테스트하는 방법이다. 기존의 테스트 방법은 해당 입력 값에 대해 예상되는 결과값과 비교하는 반면 퍼지 테스팅은 프로그램 충돌과 같은 어떤 비정상적인 상황이 발생하는지 여부를 체크하여 적은 리소스를 사용한다는 장점이 있다. 하지만 가능한 모든 경우에 대해 테스트를 해볼 수 없고 비정상적인 실행을 만들어 내지 않는 취약점 또한 있을 수 있기 때문에 좁은 탐지 범위를 갖는다. 이러한 이유로 기존의 퍼지 테스팅을 이용한 툴들은 버퍼 오버플로우는 탐지할 수 있었지만 버퍼 오버리드는 탐지하지 못하였다. 하트블리드와 같은 취약점을 탐지하기 위해서는 유효한 범위를 벗어나는 메모리 접근에 대해 탐지할 수 있어야 했지만 OpenSSL은 직접 메모리를 할당하고 해지하는 것이 아니라 캐시를 사용해 할당된 메모리를 재사용하는 방식을 사용했기 때문에 기존의 툴들이 하트블리드를 발견하는 데에 어려움이 있었다[3].

3.3 하트블리드 탐지 방법

동적 분석 방법을 보완하기 위해 Address Accessibility Checker를 함께 사용한다면 효과적으로 하트블리드와 같은 취약점들을 탐지할 수 있다.

Address Accessibility Checker 는 대부분의 버퍼 오버플로우, 오버리드, 메모리 해지 후의 사용, 메모리 누수와 같은 결함들을 찾아낼 수 있으므로 퍼지 테스팅의 탐지 범위를 늘릴 수 있다. 기존의 퍼지 테스팅의 경우에는 입력 값에 대한 결과에 대해 충돌이 발생하는지 아닌지 정도만 검사하기 때문에 좁은 탐지 범위를 가진다. 이를 보완하기 위해 추가적인 정보를 통해 특정 출력 값에 대해 취약점으로 판단하도록 하는 방법을 쓸 수 있다[3].

Snort 는 오픈 소스 네트워크 침입 탐지 시스템으로 실시간으로 트래픽을 분석한다[7]. Snort Rule Engine 은 사용자가 직접 작성한 규칙을 적용해 패킷을 분석하는 것을 가능하도록 한다. 이를 하트블리드를 감지하는데 사용하기 위해 Heartbeat Request 의 실제 레코드의 길이가 충분히 긴지에 대한 Snort 규칙을 작성하고 하트블리드를 감지할 수 있다[8].

4. 결론

하트블리드는 OpenSSL 을 사용하는 서버의 메모리 정보를 탈취하는 취약점이다. 본 논문에서는 하트블리드가 어떤 종류의 취약점인지에 대해 알아보고 그것을 발견하고 대응하기 위한 방법에 대해 조사하였다.

기존의 취약점 탐지 방법으로 하트블리드를 발견할 수 없었던 이유에 대해 살펴 보았고 퍼지 테스팅과 함께 Address Accessibility Checker 를 사용하거나 퍼지 테스팅의 탐지 범위를 높이기 위해 추가적인 정보를 사용함으로써 하트블리드와 같은 취약점을 효과적으로 탐지하는 방법을 소개하였다.

사사

본 연구는 2013 년도 정부(미래창조과학부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(No. 2013R1A2A2A01005559). 또한 이 논문은 2015 년도 정부(미래창조과학부)의 재원으로 정보통신기술진흥센터의 지원을 받아 수행된 연구임(No.R0190-15-2011, IoT 소프트웨어 보안 취약점 자동분석 기술 개발).

참고문헌

- [1] Weaver, A.C., Secure Sockets Layer, IEEE Computer Volume39, 2006
- [2] The Heartbleed Bug. <http://heartbleed.com>
- [3] Wheeler, D.A., Preventing Heartbleed, IEEE Computer Volume47, 2014
- [4] Carvalho, M., DeMott, J., Ford,R., Wheeler, D.A., Heartbleed 101, IEEE Computer Volume12, 2014.
- [5] James A. Kupsch and Barton P. Miller, Why Do Software Assurance Tools Have Problems Finding Bugs Like Heartbleed, SWAMP, 2014
- [6] Anatomy of OpenSSL's Heartbleed, http://www.theregister.co.uk/2014/04/09/heartbleed_explained/
- [7] Snort, <http://www.snort.org/>

- [8] Yu Zhang., A Snort-based Approach for Heartbleed Bug Detection, International Conference on Computer Science and Electronic Technology(ICCSET), 2014