격 과정에 대해 다룰 것이다.

1. Heartbeat 메시지의 도입 배경

TLS는 TCP 기반 암호화 프로토콜로, 지속적인 데이터 전송 없이 연결 상태(Connection Alive)를 유지할 수 있는 기능이 반드시 제공되지 않는다.

DTLS는 UDP 기반 암호화 프로토콜로, 데이터그램(Datagram)을 안전하게 전송하기 위해 설계되었다. 그러나 TLS와 달리 세션 관리 기능이 없어, 상대방이 통신 가능한 상태인지 확인하는 유일한 방법은 재협상(renegotiation) 뿐이다. 이러한 재협상은 Handshake 과정을 다시 수행해야 하므로, 특히 단방향 트래픽을 사용하는 애플리케이션에서는 비효율적이다.

이러한 한계를 해결하기 위해 2010년 6월18일 IETF의 초안 문서에서 Heartbeat 메시지 개념이 제안되었다[3].

이후 해당 개념은 RFC 6520(Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS) Heartbeat Extension)으로 발전했으며, OpenSSL 측에서는 2012년 3월 14일에 공개된 OpenSSL 1.0.1버전에서 처음으로 도입되었다.[4]

2. HEARTBEAT PROTOCOL 주요 요구사항[5]

다음은 그림 1에서 보여주고 있는 Heartbeat Message에 대한 설명이다.

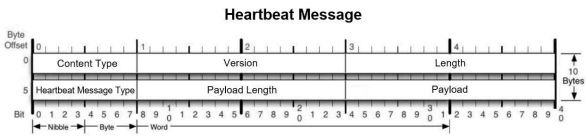

그림 1. Heartbeat Message on the TLS Record Layer[6]

- Heartbeat Protocol은 HeartbeatRequest, Heartbeat Response 두 가지 메시지 타입으로 구성된다.
- HeartbeatRequest 메시지는 연결이 유지되고 있는 동안 언제든지 도착할 수 있다.
- HeartbeatRequest 메시지를 수신하면, 상응하는 HeartbeatResponse 메시지로 응답해야 한다.
- HeartbeatRequest 수신 측은 동일한 페이로드를 포함하는 HeartbeatResponse 메시지를 반드시 전송해야 한다.
- 수신된 Heartbeat 메시지의 페이로드 길이 필드 값이 너무 큰 경우 조용히 폐기되어야 한다.
- 수신한 HeartbeatResponse 메시지가 HeartbeatRequest의 페이로드를 포함하지 않은 경우 조용히 폐기되어야 한다.
- Heartbeat Protocol 메시지는 Heartbeat 메시지 타입 필드, unsigned int16(부호 없는 16bit 정수)를 범위로 하는 페이로드 길이 필드, 페이로드 필드, 패딩 필드로 구성된다.

3. 정상적인 HEARTBEAT 메시지 처리 절차

그림 2는 정상적인 Heartbeat Message의 처리 절차(클라이언트 측 송신 가정)를 보여주고 있다.

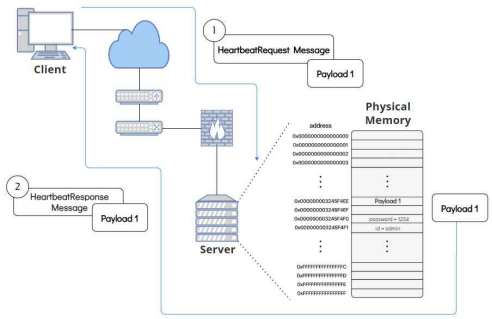

그림 2. Flow of Heartbeat Message

① 클라이언트와 서버가 암호화 통신을 위해 SSL/TLS Handshake를 한다.
② 일정 주기로 클라이언트에서 서버로 HeartbeatRequest 메시지를 송신한다.
③ 서버에서 HeartbeatRequest 메시지를 수신 후 버퍼에 저장한다.
④ 버퍼에 저장한 HeartbeatRequest 정보를 바탕으로 HeartbeatResponse 메시지를 구성하여 클라이언트에 송신한다.
⑤ 클라이언트에서 HeartbeatResponse 메시지를 수신하여 연결 상태를 확인한다.
⑥ 이 과정을 반복한다.

4. 취약점 소스코드 분석

그림 3는 OpenSSL_1_0_1/ssl/t1_lib.c[7]의 Read Heartbeat Request Message를 보여주고 있다. 다음 표들은 이에 대한 상세한 설명이다.

```
2435    #ifndef OPENSSL_NO_HEARTBEATS
2436    int
2437    tls1_process_heartbeat(SSL *s)
2438        {
2439        unsigned char *p = &s->s3->rrec.data[0], *pl;
2440        unsigned short hbtype;
2441        unsigned int payload;
2442        unsigned int padding = 16;
2443
2444        hbtype = *p++;
2445        n2s(p, payload);
2446        pl = p;
```

그림 3 Read Heartbeat Request Message

- Line 2439 실행 후: *p=1, p 위치=Request 메시지 1번째 바이트

직역	*p에 rrec.data[0] 대입, *p, *pl의 자료형은 unsigned char (1byte)

기능	p는 Heartbeat Request 메시지가 저장된 메모리의 첫 주소 지정
맥락	포인터 p로 Heartbeat Request 메시지를 다룰 예정

- Line 2444 실행 후: hbtype=1, p 위치= Request 메시지 2번째 바이트

직역	hbtype에 *p(1byte)를 자동 형변환하여 대입 후, p를 상위주소로 1byte 이동
기능	hbtype에 요청 타입값 0x01 저장, p는 2번째 byte 지정
맥락	Request 메시지에 대한 Response 메시지를 생성하게 함 (line 2453)

- Line 2445 실행 후: payload=페이로드 길이, p 위치=Request 메시지 4번째 바이트

직역	p부터 2바이트를 읽어 payload에 대입 후, p를 상위주소로 2byte 이동
기능	payload에 Request 메시지의 Payload Length 값 저장
맥락	Response 메시지에 쓰일 Payload Length 값 저장

- Line 2446 실행 후: pl 위치=Request 메시지 4번째 바이트

직역	pl에 p를 대입
기능	pl은 Request 메시지 4번째 바이트(payload 필드 시작주소) 지정
맥락	Response 메시지 구성할 때 사용

그림 4는 OpenSSL_1_0_1/ssl/t1_lib.c[7]의 Write Heartbeat Response Message를 보여주고 있다. 다음은 이에 대한 상세한 설명이다.

```
2453    if (hbtype == TLS1_HB_REQUEST)
2454        {
2455        unsigned char *buffer, *bp;
2456        int r;
2457
2458        buffer = OPENSSL_malloc(1 + 2 + payload + padding);
2459        bp = buffer;
2460
2461        *bp++ = TLS1_HB_RESPONSE;
2462        s2n(payload, bp);
2463        memcpy(bp, pl, payload);
2464
2465        bp += payload;
2466        RAND_pseudo_bytes(bp, padding);
2467
2468        r = ssl3_write_bytes(s, TLS1_RT_HEARTBEAT, buffer,\
2469                             3 + payload + padding);
```

그림 4. Write Heartbeat Response Message

- Line 2458 실행 후: buffer=Heartbeat Response 메시지 버퍼 시작 주소

직역	buffer에 동적할당한 메모리 주소 대입
기능	Response 메시지 구성 시 필요한 저장공간 확보
맥락	Response 메시지 구성에 필요한 메모리 동적할당

- Line 2459 실행 후: bp=Heartbeat Response 메시지 버퍼 시작 주소

직역	bp에 buffer 값 대입
기능	bp는 포인터이므로 하위주소에서 상위주소로 저장 가능
맥락	bp로 Response 메시지 다룰 예정

- Line 2461 실행 후: *bp=2, bp=Heartbeat Response 메시지 2번째 바이트

직역	*bp에 TLS1_HB_RESPONSE(2) 저장 후, bp를 상위주소로 2byte 이동
기능	Response 메시지 Type 필드 값 저장 후, bp는 메시지 2번째 byte 지정
맥락	Response 메시지에 Type 필드값 저장

- Line 2462 실행 후: bp=Heartbeat Response 메시지 4번째 바이트

직역	변수 payload(2byte)를 읽어 빅 엔디안으로 변환 후, bp에 저장 후 bp를 상위주소로 2byte 이동
기능	bp에 Payload Length값 저장
맥락	Response 메시지에 Payload Length 빅 엔디안 방식으로 저장

- Line 2463 실행 후:

직역	pl부터 변수 payload 값만큼 읽어 bp에 쭉 저장
기능	pl이 가리키는 주소부터 payload(Reqeust 메시지 Payload Legth값) 값만큼 bp에 복사
맥락	조작된 경우 버퍼 Out-of-bound Read 발생

- Line 2468 실행 후:

직역	*buffer(Response 메시지 전체)를 ssl3_write_bytes() 함수에서 처리
기능	*buffer를 암호화 후 전송
맥락	Response 메시지 전송

그림 3 및 그림 4에 제시된 코드를 분석한 결과, Heartbeat Request 메시지의 Payload Length 값과 실제 payload의 길이를 비교하여 검증하는 로직이 존재하지 않으며, Payload Length 값이 매우 클 경우 메시지를 조용히 폐기(silent discard)하라는 프로토콜 요구사항을 구현한 코드 또한 확인되지 않는다.[5][7] 결국 이러한 입력값 검증 누락과 프로토콜 불완전 구현이 Heartbleed 공격의 원인이 되었다.

그림 5는 OpenSSL_1_0_1/ssl/ssl.locl.h[7]의 n2s, s2n 매크로를 보여주고 있다. n2s(c, s) 매크로는 c[0]를 상위 바이트로 간주하여 왼쪽으로 8bit shift 한 후 c[1]과 OR연산으로 결합하여, s에 16bit 정수로 저장한다. 이는 네트워크 바이트를 받아 해당 시스템의 내부 저장방식에 맞게 변환하는 역할을 한다. s2n(s, c) 매크로는 정수 s의 상위 바이트를 우로 8bit shift 하여 0xff와 AND 연산으로 값을 추출하여 c[0]에 저장하고, 하위 바이트는 0xff과 AND연산으로 추출하여 c[1]에 저장한다. 이 과정은 시스템 내부 저장 방식으로 저장된 정수 값을 네트워크 바이트 형식으로 변환하여 전송 가능한 형태로 만드는 역할을 한다.

```
#define n2s(c,s) ((s=(((unsigned int)(c[0]))<< 8)| \
                  (((unsigned int)(c[1])))), c+=2)

#define s2n(s,c) ((c[0]=(unsigned char)(((s)>> 8)&0xff), \
                  c[1]=(unsigned char)(((s))&0xff)), c+=2)
```

그림 5 n2s, s2n 매크로

5. 취약점을 이용한 공격 과정

그림 5는 Heartbleed Attack의 전체 흐름을 설명하고 있다.

① 클라이언트와 서버가 암호화 통신을 위해 SSL/TLS Handshake를 한다.

② 클라이언트(공격자)는 HeartbeatRequest 메시지의 Payload Length 값을 조작하고 Payload 내용을 최대한 짧은 데이터로 조작하여 송신한다.

③ 서버에서 어떠한 인증 없이 HeartbeatRequest 메시지를 수신 후 버퍼에 저장한다.

④ 버퍼에 저장된 payload부터 Payload Length 값을 길이로 한 새로운 payload를 HeartbeatResponse 메시지에 복사하여 클라이언트(공격자)에 송신한다.

⑤ 클라이언트에서 수신한 메시지에서 원 페이로드를 제외한 나머지 데이터를 조합하여 유의미한 정보를 추출한다.

⑥ 이 과정을 수차례 반복한다.

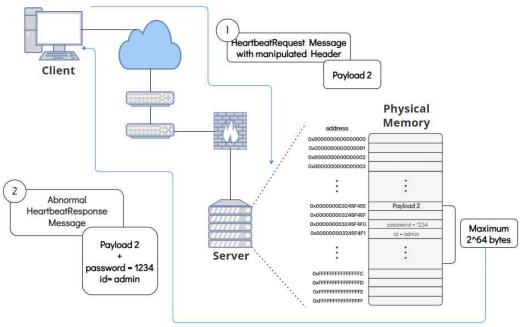

그림 6. Flow of Heartbleed Attack

Ⅳ. 대응 방안 (Countermeasures)

하트블리드 공격은 다양한 피해를 남겨는데, 대표적인 예는 다음과 같다.

- 캐나다 국세청(CRA): 900명의 사회보장번호 유출, 캐나다 국세청 5일간 온라인 서비스 중지, 사회보장 번호 재발급
- 미국: Community Health Systems(미국의 영리병원체인 중 2번째 규모)의 450만 명의 환자기록 유출 당함[8]
- 영국 Mumsnet: 150만 명 회원정보 유출 당함
- OKCupid: 사용자의 로그인 정보와 메시지 내용(채팅 내역, 성적 취향 등) 노출

위에서 기술한 사례들처럼, 해당 취약점은 광범위하게 영향을 미쳤다. 또한 취약점이 공개되기 전까지는 Heartbeat 메시지를 IDS/IPS에서 로그에 저장하는 것이 기술적으로 가능했음에도 불구하고, 대부분의 기관은 그 필요성을 느끼지 못해 기록하지 않았다. 이로 인해 특정 기관의 온라인 서비스가 취약한 지는 알 수 있었지만 공격이 실제로 발생했는지, 발생했다면 어느 범위까지 피해가 확산되었는지를 정확히 파악할 수 없게 되었다. 다음은 대응방안을 설명한다.

1. 취약점 발견 전 예방조치

- 개발 측 예방조치
  - ✓ 시큐어코딩 준수
  - ✓ 프로토콜 표준문서와 구현코드 비교검증
  - ✓ 퍼징(Fuzzing) 테스트

- 제도적 대응방안
  - ✓ 소스코드 외부 감사
  - ✓ 버그바운티 활성화