Внешний аудит безопасности корпоративных сетей

Лекция 7
Уязвимости памяти

Семён Новосёлов

Crypto Kantiana

БФУ имени И. Канта

2021

Уязвимости памяти

- Появляются при некорректной работе с памятью.
- Наиболее уязвимы программы на языках с прямым управлением памятью (C/C++)
- Примеры:
  - **Утечки памяти**
  - Переполнения буфера

Утечки памяти

К утечкам памяти может приводить чтение данных вне допустимого диапазона.

Наиболее известная уязвимость: OpenSSL Heartbleed.

Пример. Heartbleed (CVE-2014-0160)

- уязвимость в OpenSSL, широко распространенной библиотеке
- позволяет красть информацию кусочками по 64 килобайт из зашифрованных соединений, которые используют эту библиотеку
- были уязвимы сервисы Yandex, Google, банковские сайты

TLS. Расширение Heartbeat

- появилось в феврале 2012 года (RFC 6520)
- используется для поддержания соединения в открытом состоянии (keep-alive)
- для этого периодически посылаются специальные пакеты с полезной нагрузкой (тестовой строкой)
- сервер посылает в ответ пакет с той же самой нагрузкой

Уязвимый код

считывание размера полезной нагрузки из пришедшего пакета

```
unsigned int payload;
unsigned int padding = 16; /* Use minimum padding
*/
/* Read type and payload length first */
hbtype = *p++;
n2s(p, payload);
pl = p;
//...
/* Allocate memory for the response, size is 1 byte
 * message type, plus 2 bytes payload length, plus
 * payload, plus padding */
buffer = OPENSSL_malloc(1 + 2 + payload +
padding);
bp = buffer;
```

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

r = dtls1_write_bytes(s, TLS1_RT_HEARTBEAT,
buffer, 3 + payload + padding);
```

Уязвимый код

выделение памяти под ответный пакет.
Ошибка: нет проверки входных данных (payload).

```
unsigned int payload;
unsigned int padding = 16; /* Use minimum padding
*/
/* Read type and payload length first */
hbtype = *p++;
n2s(p, payload);
pl = p;
//...
/* Allocate memory for the response, size is 1 byte
 * message type, plus 2 bytes payload length, plus
 * payload, plus padding */
buffer = OPENSSL_malloc(1 + 2 + payload +
padding);
bp = buffer;
```

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

r = dtls1_write_bytes(s, TLS1_RT_HEARTBEAT,
buffer, 3 + payload + padding);
```

Уязвимый код

копирование присланной нагрузки Heartbeat в пакет-ответ
Ошибка: если прислать меньше данных, чем указано в payload, то скопируются данные в памяти лежащие после buffer

```
unsigned int payload;
unsigned int padding = 16; /* Use minimum padding
*/
/* Read type and payload length first */
hbtype = *p++;
n2s(p, payload);
pl = p;
//...
/* Allocate memory for the response, size is 1 byte
 * message type, plus 2 bytes payload length, plus
 * payload, plus padding */
buffer = OPENSSL_malloc(1 + 2 + payload +
padding);
bp = buffer;
```

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

r = dtls1_write_bytes(s, TLS1_RT_HEARTBEAT,
buffer, 3 + payload + padding);
```

Уязвимый код

Формирование выходного пакета для последующей отправки данных

```
unsigned int payload;
unsigned int padding = 16; /* Use minimum padding
*/
/* Read type and payload length first */
hbtype = *p++;
n2s(p, payload);
pl = p;
//...
/* Allocate memory for the response, size is 1 byte
 * message type, plus 2 bytes payload length, plus
 * payload, plus padding */
buffer = OPENSSL_malloc(1 + 2 + payload +
padding);
bp = buffer;
```

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

r = dtls1_write_bytes(s, TLS1_RT_HEARTBEAT,
buffer, 3 + payload + padding);
```

Эксплуатация

- В поле размера полезной нагрузки (payload) ставить большое значение (макс. 64 КБ)
- Фактически присылать меньше данных
- Сервер выдаст в ответ участок памяти процесса, лежащий после buffer
- Эксплойты есть в открытом доступе (www.exploit-db.com)

Что может утечь?

- секретные ключи TLS
- cookies
- логины
- пароли
- куски писем
- любые другие данные, которыми обменивается сервер и его клиенты

Уязвимость двусторонняя: сервер тоже может читать данные клиента

Исправление

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

Добавление проверки с фактически полученным размером данных

Переполнение буфера. Вводный пример

В примере производится запись за пределы буфера.

```
int main(){
    char buf[100];
    int i;
    for(i=0; i<=108; i++){
        buf[i] = 'A';
    }
    return 0;
}
```

Получается бесконечный цикл.
Почему?

Переполнение буфера. Вводный пример

В примере производится запись за пределы буфера.

```
int main(){
    char buf[100];
    int i;
    for(i=0; i<=108; i++){
        buf[i] = 'A';
    }
    return 0;
}
```

Получается бесконечный цикл.
Почему?

Ответ: бесконечно перезаписывается переменная i, лежащая в памяти после буфера

Как избежать подобных ошибок?

Очевидно: проверять корректность всех полученных извне данных.

Выполнять аудит кода с помощью инструментов статического и динамического анализа кода.

Статический анализ

Проверка исходного кода без запуска программы.

Много ложных срабатываний и много времени тратится на анализ отчётов.

Инструменты:

- cppcheck
- splint

Динамический анализ кода

Анализ запущенной программы.

Для детектирования ошибок может использоваться следующая связка:

- Valgrind (Memcheck) или Google Address/Memory Sanitizer для детектирования ошибок
- Фаззеры для генерации отправки случайных/мусорных данных на вход программы (есть в составе Metasploit)

Замечание: Valgrind ищет только ошибки динамической памяти (переполнение стека/статических буферов не детектируются)

Литература и ссылки

- Google Address/Memory Sanitizer
  - https://github.com/google/sanitizers/wiki/MemorySanitizer
  - https://github.com/google/sanitizers/wiki/AddressSanitizer
- CWE-126: Buffer Over-read
  - https://cwe.mitre.org/data/definitions/126.html
- CWE-121: Stack-based Buffer Overflow
  - https://cwe.mitre.org/data/definitions/121.html
- Heartbleed
  - https://heartbleed.com/