Внешний аудит безопасности корпоративных сетей

Лекция 6
Утечки данных из памяти

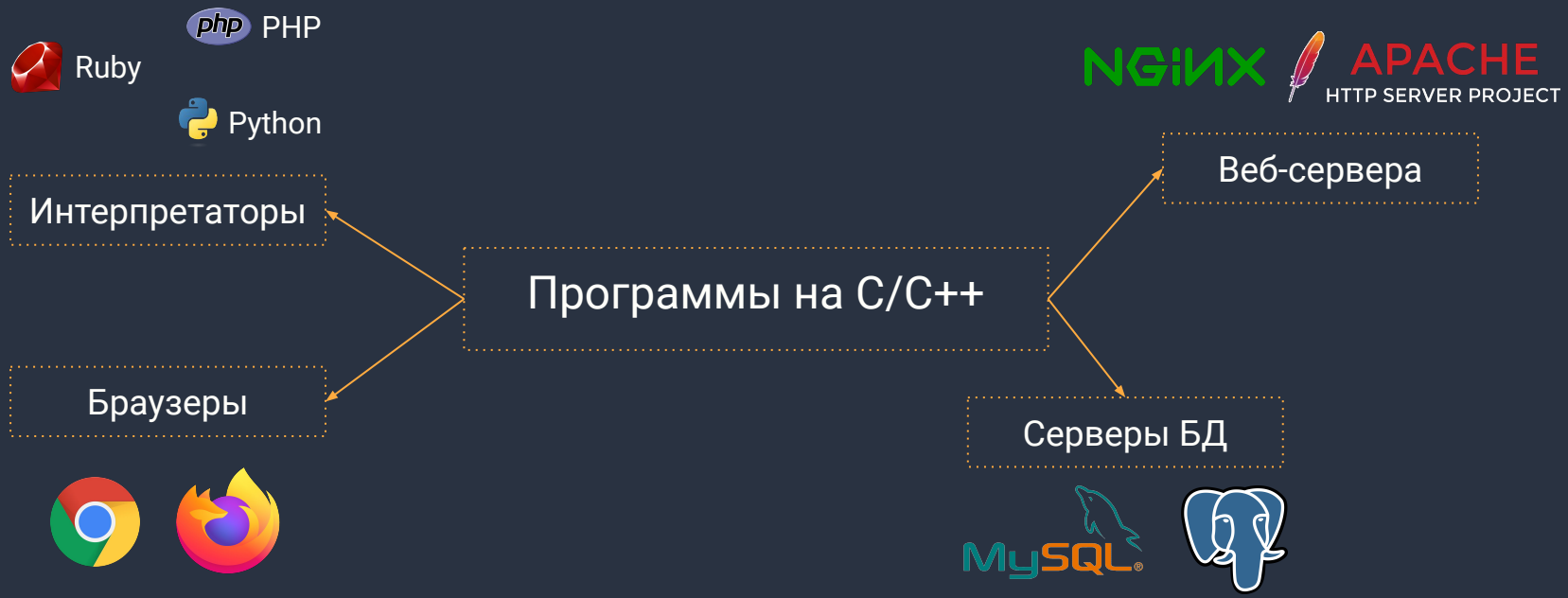
Семён Новосёлов

2022

Утечки данных из памяти

- К утечкам памяти может приводить чтение данных вне допустимого диапазона.
- Наиболее уязвимы программы на языках с прямым управлением памятью (C/C++)
- Самая известная уязвимость: OpenSSL Heartbleed.

В каких программах может встречаться?



Пример. Уязвимость Heartbleed

Где?

OpenSSL: широко распространенная библиотека

Кого затронуло?

- Yandex
- Google
- банковские сайты
- (всех)

Что утекает?

кусочки по 64 Кб из зашифрованных соединений

TLS. Расширение Heartbeat

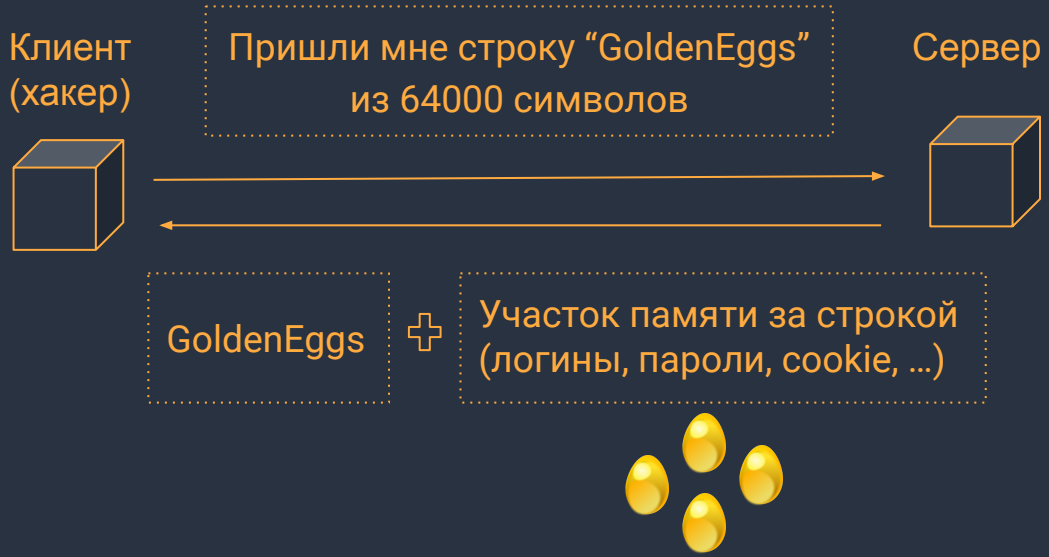
Появление: февраль 2012 года

Функция: поддержание соединения в открытом состоянии (keep-alive)

Принцип работы:

- периодически посылаются пакеты с текстовой строкой
- сервер посылает в ответ пакет с той же самой строкой

Схема уязвимости



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 копируются 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

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.

Недостатки: Много ложных срабатываний и много времени тратится на анализ отчётов.

Инструменты:

cppcheck

Использование инструментов статического анализа кода

CppCheck

```
cppcheck --bug-hunting  main.c
```

Splint

```
splint -strict main.c
```

Динамический анализ кода

Анализ запущенной программы.

Для нахождения ошибок может использоваться следующая связка:

Фаззеры

Valgrind (Memcheck)

Google Address/Memory Sanitizer

генерация и отправка мусорных данных на вход программы

детектирование ошибок

Фаззеры

Есть в составе Metasploit:

```
search auxiliary/fuzzers/
```

Утилиты из входящие в Kali Linux:

https://kali.tools/all/?category=fuzzer/

Использование динамических инструментов поиска ошибок памяти

Google Address Sanitizer

```
gcc -fsanitize=address -o main main.c
```

Valgrind

```
valgrind --tool=memcheck ./main
```

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

Пример уязвимой программы. Hexer

- Сервис, который висит на заданном порту
- При подключении запрашивает данные в формате:

[размер строки]
[строка]

- После получения данных выдаёт строку в формате hexdump.

Код функции main. Часть I

Занимаем за процессом, заданный в консоли порт:

```
int main(int argc, char *argv[]) {
    int sockfd, newsockfd, portno, clilen;
    char buffer[256];
    struct sockaddr_in serv_addr, cli_addr;
    int n;

    if(argc < 2) {
        printf("Enter port number\n");
        return 1;
    }
    portno = atoi(argv[1]);

    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd < 0) {
        perror("ERROR opening socket");
        return 1;
    }
```

```
    /* Initialize socket structure */
    bzero((char *) &serv_addr, sizeof(serv_addr));
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = INADDR_ANY;
    serv_addr.sin_port = htons(portno);

    int timeout = 5 * 1000;
    setsockopt(sockfd, SOL_SOCKET, SO_RCVTIMEO,
               (const char*)&timeout, sizeof timeout);

    /* Now bind the host address using bind() call.*/
    if (bind(sockfd, (struct sockaddr *) &serv_addr,
             sizeof(serv_addr)) < 0) {
       perror("ERROR on binding");
       return 1;
    }
```

Код функции main. Часть II

Принимаем в основном потоке подключения клиентов:

```
/* Now start listening for the clients, here process will go in sleep mode
and will wait for the incoming connection */
    listen(sockfd,5);
    clilen = sizeof(cli_addr);
    while (1) {
        newsockfd = accept(sockfd, (struct sockaddr *)&cli_addr, &clilen);
        if (newsockfd < 0) {
            perror("ERROR on accept");
            return 1;
        }
        /* Create child process */
        pid_t pid = fork();
        if (pid < 0) {
            perror("ERROR on fork");
            return 1;
        }
```

создаём форк процесса для обработки запроса в отдельном потоке

Код функции main. Часть III

Если мы находимся не в основном потоке, то передаём обработку запроса функции doproccesisng().

```
if (pid == 0) {
   /* This is the client process */
            close(sockfd);
            int r = doprocessing(newsockfd, &cli_addr);
            printf("client process: done with code %d", r);
            return r;
        }
        else
        {
            close(newsockfd);
        }
    } /* end of while */
    printf("main: done");
    return 0;
}
```

Функция обработки запроса

```
int doprocessing(int sock, struct sockaddr_in *addr) {
    banner(sock);
    //char hex[3000];
    char buf[128] = "";
    unsigned int buf_size;
    read_message(sock, addr, buf, &buf_size);

    int n = write(sock, "Your hexed string:\n", 19);
    if (n < 0) {
        perror("ERROR writing to socket\n");
        return 1;
    }
    write_hexdump(sock, buf, buf_size);

    printf("doprocessing: done\n");
    return 0;
}
```

Чтение данных от клиента

```
int read_message(int sock, struct sockaddr_in *addr, char *buf, int *buf_size) {
    char client[16] = "";
    inet_ntop(AF_INET, &addr->sin_addr, client, sizeof client);
    printf("Reading message from client %s ... \n", client);
    int n = read(sock, buf, 4);
    if (n < 0) {
        perror("ERROR reading from socket");
        return 1;
    }
    buf[n] = 0; *buf_size = atoi(buf);
    printf("-> Got message size: %i\n", *buf_size);

    int flag = 0; char *p = buf;
    while(1) {
      n = read(sock, p, 1);
      if (n < 0) break;
      if (flag && p[0] == '\n') break; // '\n\n' is end of message
      flag = (p[0] == '\n');
      p+=n;
    }
    p[-1]=0;
    return 0;
}
```

нет проверки размера входных данных

Отправка преобразованной строки клиенту

```
void write_hexdump(int sock, void *mem, unsigned int len) {
    unsigned int i, j, cols = 16;
    char str[64] = "";
    for(i = 0; i < len + ((len % cols) ? (cols - len % cols) : 0); i++) {
        if(i < len) {
          sprintf(str, "%02x ", 0xFF & ((char*)mem)[i]);
          write(sock, str, strlen(str));
        }
        else write(sock, "   ", 3);
        if(i % cols == (cols - 1)) {
            for(j = i - (cols - 1); j <= i; j++) {
                if(j >= len)
                  write(sock, " ", 1);
                else if(isprint(((char*)mem)[j]))
                  write(sock, ((char*)mem)+j, 1);
                else
                  write(sock, ".", 1);
            }
            write(sock, "\n", 1);
        }
    }
}
```

так как нет проверки корректности размера данных, клиенту отправится кусок памяти процесса

Эксплуатация уязвимости

- Достаточно отправить строку меньше размера, чем указано
- В ответ отправится кусок памяти процесса

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/