Разработка модуля ответного реагирования для IDS Snort

Н.В. Егоров[1], С.А. Бурлов[1]

[1]Самарский национальный исследовательский университет им. академика С.П. Королева, Московское шоссе 34А, Самара, Россия, 443086

Аннотация. Данная работа посвящена созданию дополнительных средств защиты сети на базе IDS Snort. Разработан модуль на языке программирования C для сетевой IDS Snort, который позволяет автоматизировать некоторые действия.

1. Введение

Появление сетевых технологий на рубеже XX-XXI веков позволило людям мгновенно обмениваться информацией, осуществлять быстрый поиск информации и хранить большие объемы информации в виртуальном пространстве – облаке. Появились возможности организации распределённых вычислений и создания распределенных клиент-серверных информационных систем [1]. Несмотря на многочисленные преимущества сетевых технологий, появились и новые угрозы безопасности информации. Так по статистике российской компании, занимающейся защитой информации, на 2018 год, ежедневно в мире совершается больше миллиона сетевых преступлений [2] от кражи личных данных до отказа в обслуживании промышленных объектов.

Ответом на сетевые угрозы стало создание программно-аппаратных комплексов, способных определять вторжения на стадии их подготовки, например во время сетевой разведки. В частности, сетевые IDS являются примером таких программно-аппаратных комплексов.

Данная работа посвящена созданию дополнительных средств обеспечения безопасности сети на базе IDS Snort. Разработан модуль на языке программирования C, который позволяет автоматизировать некоторые действия.

2. Настройка IDS Snort

Для тестового примера используется веб-приложение, написанное на языке программирования Java. В веб-приложении намеренно создана уязвимость к SQL-инъекциям. Рабочий порт веб-приложения: 8080.

Для защиты веб-приложения в IDS Snort, подключается модуль NFQ, который позволяет взаимодействовать с механизмом NFQUEUE ядра Linux. Для настройки NFQUEUE используется утилита iptables. Команда, которая перенаправляет сетевой трафик в очередь NFQUEUE

```
iptables -t filter -A OUTPUT -p tcp --dport 8080 -j NFQUEUE --queue-num=2,
```

где опция -t filter указывает на таблицу сетевого фильтра, опция -A OUTPUT добавляет правило в цепочку правил OUTPUT, опции -p tcp и --dport 8080 указывают на тип сетевого пакета, который соответствует данному правилу. Опция -j NFQUEUE указывает на цель

достижения сетевого пакета, а опция --queue-num=2 указывает на номер очереди NFQUEUE, в которую будут попадать сетевые пакеты.
Для того, чтобы Snort удалял сетевые пакеты, попадающие под правило, необходимо запустить его в inline режиме для этого используется опция -Q. Команда для запуска Snort

snort -Q --daq nfq --daq-dir=/usr/local/lib/daq --daq-var queue=2
-c ./community-rules/snort.conf -l /var/log/snort,

где опция --daq определяет тип используемого модуля Snort. Опция --daq-var задает параметры модуля. Опция -c указывает на путь к конфигурационному файлу Snort, а опция -l определяет директорию, где будут появляться лог-файлы.
Правила для Snort описаны в файле local.rules. В файле описано правило для поиска SQL-инъекции с использованием слова union. Данное правило изображено на рисунке 1.

local.rules

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 8080 (msg:"SQL Injection | union select keywords";
pcre:"/((((\%55)|(u)|(\%75))((\%4e)|(n)|(\%6e))((\%69)|(i)|(\%49))((\%6f)|(o)|(\%4f))
((\%4e)|(n)|(\%6e)))[^\ n]*(((\%73)|(s)|(\%53))((\%65)|(e)|(\%45))((\%6c)|(l)|(\%4c))
((\%65)|(e)|(\%45))((\%63)|(c)|(\%43))((\%74)|(t)|(\%54))))/i";
classtype: Web-application-attack; sid:2)
```

Рисунок 1. Правило для IDS Snort.

3. Построение модуля для IDS Snort

В Snort есть возможность передать событие «alert», генерируемое во время срабатывания правила, внешней программе или внешнему процессу при помощи плагина alert_unixsock. Плагин передает данные другой программе через доменный сокет Unix. Для добавления alert_unixsock в Snort, нужно переконфигурировать IDS командой ./configure --enable-control-socket.
Для включения плагина нужно при запуске Snort задать опцию -A unsock и опцию -l /path/to/log/files. Параметр -l задает путь, где Snort будет искать файл доменного сокета Unix. Имя файла фиксировано – «snort_alert». Плагин alert_unixsock выступает в роли клиента, который подключается к доменному сокету Unix и отправляет данные. В исходных файлах Snort можно найти реализацию плагина в файлах spo_alert_unixsock.c и spo_alert_unixsock.h.
Для отправки данных по доменному сокету Unix используется структура Alertpkt. Структура изображена на рисунке 2.

spo_alert_unixsock.h

```
typedef struct _Alertpkt
{
    uint8_t alertmsg[ALERTMSG_LENGTH];

    uint32_t dlthdr;
    uint32_t nethdr;
    uint32_t transhdr;
    uint32_t data;
    uint32_t val;

    struct pcap_pkthdr32 pkth;
    uint8_t pkt[65535];
    Event event;
} Alertpkt;
```

Рисунок 2. Структура Alertpkt.

В массив alertmsg структуры Alertpkt копируется сообщение правила, которое стало причиной генерации события «alert». Соответствующая команда изображена на рисунке 3.
В массив pkt копируется сетевой пакет, который вызвал генерацию события. Соответствующая команда изображена на рисунке 4.
Переменные dlthdr, nethdr и transhdr отвечают за смещение заголовков сетевого пакета в массиве pkt. Переменная val содержит флаги состояния сетевого пакета. Переменная data

отвечает за смещение полезной нагрузки сетевого пакета в массиве pkt. Все данные о событии содержатся в структуре Event. Структура изображена на рисунке 5.

spo_alert_unixsock.c

```
173 static void AlertUnixSock(Packet *p, const char *msg, void *arg, Event *event) {
174     static Alertpkt alertpkt;
202     if (msg) {
203         memmove( (void *)alertpkt.alertmsg, (const void *)msg, strlen(msg));
204     }
261 }
```

Рисунок 3. Копирование сообщения правила в массив alertmsg.

spo_alert_unixsock.c

```
173 static void AlertUnixSock(Packet *p, const char *msg, void *arg, Event *event) {
174     static Alertpkt alertpkt;
188     if(p && p->pkt) {
196         memmove( alertpkt.pkt, (const void *)p->pkt, p->pkth->caplen);
197     }
260 }
```

Рисунок 4. Копирование сетевого пакета в массив pkt.

event.h

```
38 typedef struct _Event
39 {
40     uint32_t sig_generator;
41     uint32_t sig_id;
42     uint32_t sig_rev;
43     uint32_t classification;
44     uint32_t priority;
45     uint32_t event_id;
46     uint32_t event_reference;
47
48     struct sf_timeval32 ref_time;
49 } Event;
```

Рисунок 5. Событие генерируемое IDS Snort.

Переменная sig_generator определяет, какой модуль IDS Snort сгенерировал событие. Переменная sig_id определяет идентификатор генератора события. Переменные classification, priority, event_id и event_reference определяют данные о событии.

Для взаимодействия с плагином alert_unixsock необходимо реализовать модуль, выступающий в роли сервера, который будет получать информацию о генерируемом событии по доменному сокету Unix.

Перед запуском модуля необходимо передать ему путь к файлам журналирования Snort. В соответствующей директории появится файл доменного сокета Unix. В программе доменный сокет Unix создается командой sockfd = socket (AF_UNIX, SOCK_DGRAM, 0). Для получения данных используется команда

recv = recvfrom (sockfd, (void *) &alert, sizeof (alert), 0, 0, 0),

где переменная alert – получаемые данные.

Сетевой адрес источника, который стал причиной генерации события, берется из массива pkt структуры Alertpkt, соответствующая команда

ip[i] = alert.pkt[ip_offset+i].

Для определения сгенерированного события, используется массив alertmsg из структуры Alertpkt. Массив сравнивается со строкой, определенной в файле с правилами для IDS Snort. Например, для правила приведенного выше на рисунке 1, соответствующая команда

strcmp((const char*) alert.alertmsg, "SQL Injection | union select keywords").

Для уведомления пользователя о сгенерированном событии используется команда

printf ("msg: %s, [ip: %d.%d.%d.%d]\n", alert.alertmsg, ip[0], ip[1], ip[2], ip[3]).

При идентификации события, как SQL-инъекция, происходит блокирование источника атаки. Для этого используется утилита сетевого фильтра iptables, соответствующая команда

sprintf (cmd, "iptables -A INPUT -s %d.%d.%d.%d -j DROP", ip[0],ip[1],ip[2],ip[3]).

Работа программы изображена на рисунке ниже.

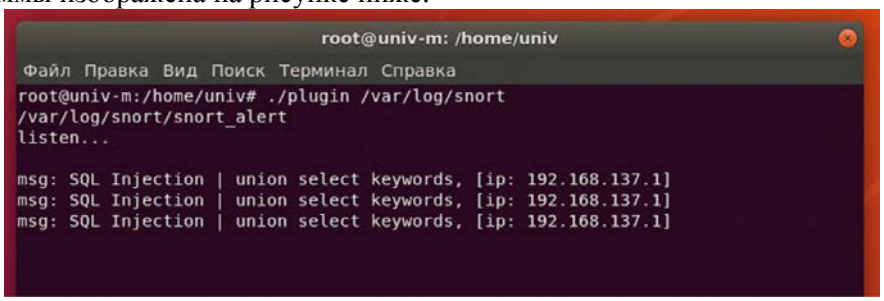

Рисунок 6. Работа программы.

4. Выводы

Написанный модуль позволяет автоматизировать некоторые действия администратора, например, заблокировать злоумышленника, пытающегося ввести SQL-инъекцию. Стоит отметить, что в IDS Snort есть возможность ограничить число событий за определенный промежуток времени, что позволит снизить нагрузку на ЦПУ во время работы IDS Snort. Полный текст модуля приведен в приложении А.

5. Литература

[1] Олифер, В.Г. Сетевые операционные системы: учебник для вузов / В.Г. Олифер, Н.А. Олифер. – СПб.: Питер, 2009. – 669 с.

[2] Kaspersky Cyberstat. Киберстатистика от "Лаборатории Касперского" в реальном времени [Электронный ресурс]. – Режим доступа: https://cyberstat.kaspersky.com (24.11.2018).

[3] Snort - Network Intrusion Detection & Prevention System [Electronic resource]. – Access mode: https://www.snort.org (24.11.2018).

[4] SNORT Users Manual 2.9.12 [Electronic resource]. – Access mode: http://manual-snort-org.s3-website-us-east-1.amazonaws.com (24.11.2018).

[5] Стивенс, У.Р. UNIX: разработка сетевых приложений / У.Р. Стивенс, Б. Феннер, Э.М. Рудофф. – СПб.: Питер, 2007. – 1039 с.

Приложение А

Текст программы.

```
#include <sys/socket.h>
#include <string.h>
#include <sys/un.h>
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <unistd.h>

#include <pcap_pkthdr32.h>
#include <event.h>

#define  UNSOCK_FILE path

typedef struct _Alertpkt {
   struct pcap_pkthdr32 pkth;
```

```
    uint8_t     alertmsg [108];
    uint8_t     pkt [65535];
    uint32_t    dlthdr;
    uint32_t    nethdr;
    uint32_t    transhdr;
    uint32_t    data;
    uint32_t    val;
    Event       event;
} Alertpkt;

sockaddr_un snortaddr;
Alertpkt    alert;
char*       path;
int         sockfd;
int         recv;

typedef unsigned char uint8_t;
typedef unsigned int  uint32_t;

void sig_term (int sig) {
    printf  ("Exiting!\n");
    close   (sockfd);
    unlink  (UNSOCK_FILE);
    exit    (1);
}

int main (int argc, char* argv[]) {
    signal (SIGINT, sig_term);

    if (argc < 2) {
        printf ("%d, Enter path to log...\n", argc);
        return 0;
    }

    path = argv[1];
    if (path[strlen(path)-1] != '/') {
        path = strcat (path, "/snort_alert");
    } else {
        path = strcat (path, "snort_alert");
    }
    printf ("%s\n", path);

    if ((sockfd = socket (AF_UNIX, SOCK_DGRAM, 0)) < 0) {
        perror  ("socket");
        exit    (1);
    }

    bzero   (&snortaddr, sizeof (snortaddr));
    snortaddr.sun_family = AF_UNIX;
    strcpy  (snortaddr.sun_path, UNSOCK_FILE);

    if (bind (sockfd, (struct sockaddr *) &snortaddr, sizeof (snortaddr)) < 0) {
        perror  ("bind");
```

```
    exit    (1);
  }
  printf("listen...\n\n");

  while ((recv = recvfrom (sockfd, (void *) &alert, sizeof (alert), 0, 0, 0)) > 0) {
    int ip_offset = alert.nethdr + 160;
    int ip[4];

    for(int i = 0; i < 4; i++) {
      ip[i] = alert.pkt[ip_offset + i];
    }

    printf ("msg: %s, [ip: %d.%d.%d.%d]\n", alert.alertmsg,
          ip[0],ip[1],ip[2],ip[3]);

    if (strcmp ((const char*) alert.alertmsg,
              "SQL Injection | union select keywords") == 0) {
      char*   cmd = new char[100];
      sprintf (cmd, "iptables -A INPUT -s %d.%d.%d.%d -j DROP",
            ip[0],ip[1],ip[2],ip[3]);
      system  (cmd);
    }
  }
  perror  ("recvfrom");
  close   (sockfd);
  unlink  (UNSOCK_FILE);
  return 0;
}
```

Development of active response module for Snort IDS

N.V. Egorov[1], S.A. Burlov[1]

[1]Samara National Research University, Moskovskoe Shosse 34A, Samara, Russia, 443086

Abstract. This work is dedicated to the creation of additional network security tools based on IDS Snort. A module was developed in the C programming language for the network IDS Snort, which allows you to automate some actions.