REPORT

NIDS, HIDS 코드 분석

과 목 명 : 고급프로그래밍실습

담당교수 : 남재현

학　　과 : 컴퓨터공학과

학　　번 : 32191597

이　　름 : 박민규

제 출 일 : 2022-12-23

1. NIDS 코드 분석

1.0 Makefile

일반적으로 소스 파일을 단일 디렉토리에 두지 않기 때문에, Makefile 을 이용한 컴파일을 진행한다. 각각의 c 파일이 여러 디렉토리에 분산되어 있는데, vpath 를 통해 디렉토리를 명시적으로 지정하지 않아도 각 파일을 찾을 수 있도록 한다. 주요 명령어로 실행 파일을 생성하는 make all, 실행 파일과 오브젝트 파일들을 제거하는 make clean 이 있다. all 이나 clean 이라는 파일명을 가진 파일이 있을 수 있기 때문에 PHONY 를 설정해준다.

1.1 context.c

context.c 파일에서는 명령행 전달인자로 옵션들을 받아 받은 옵션에 따라 struct context_t 를 가리키고 있는 포인터 ct 에 접근해 ctx->source, string_match, regex_match, ctx->log 를 업데이트한다.

```
// function pointer configuration

if (interface_flag) {
    ctx->reception_func = pkt_live;
    ctx->reception_destroy_func = destroy_pkt_live;
} else {
    ctx->reception_func = pkt_file;
    ctx->reception_destroy_func = destroy_pkt_file;
}

ctx->decoder_func = decoder;

if (strcmp(string_match, "string1") == 0) {
    ctx->string_match_func = match_string1;
} else if (strcmp(string_match, "string2") == 0) {
    ctx->string_match_func = match_string2;
} else {
    ctx->string_match_func = match_string1;
}

if (regex_match) {
    ctx->regex_match_func = match_regex;
}

if (strcmp(ctx->log, "stdout") != 0) {
    ctx->log_func = log_file;
} else {
    ctx->log_func = log_stdout;
}
```

그 후 전달된 옵션과 모듈 선택 조건에 따라 context_t 에 정의되어 있는 함수 포인터를 사용해 함수를 연결한다.

1.2.1 pkt_live.c

컴퓨터에서 패킷을 전송하면 네트워크를 통해 라우팅과 포워딩이라는 단계를 거쳐 대상 컴퓨터의 네트워크 인터페이스로 패킷이 전당된다. 패킷 캡처는 이처럼 네트워크를 통해 패킷들이 전달되는 도중 패킷의 내용을 확인하는 것을 의미한다. 패킷 캡처는 주로 네트워크의 상태를 파악하기 위한 통계 수집, 보안을 목적으로 한 패킷 검사, 또는 네트워크를 디버깅하기 위해 많이 사용된다.

pkt_live.c 는 인터페이스 기반 패킷 캡처 엔진으로, -I 옵션을 통해 인터페이스 이름을 받은 경우 해당 인터페이스를 통해 패킷 캡처를 진행한다. -I 옵션이 없다면 인터페이스 목록 중 첫 번째 인터페이스를 통해서 패킷 캡처를 진행한다. pcap_open_live() 함수를 통해 핸들러를 생성하고, pcap_loop() 함수를 통해 패킷 캡처를 한다.

1.2.2 pkt_file.c

pkt_file.c 는 파일 기반 패킷 캡처 엔진이다. -f 옵션을 통해 파일을 받은 경우 해당 파일로부터 패킷을 읽어드린다. pcap_open_offline() 함수를 통해 ctx->source 로부터 패킷을 읽는 핸들러를 생성한다. 그리고 위와 마찬가지로 pcap_loop() 함수를 통해 패킷 캡처를 진행한다.

1.2.3 packet_pool.c

런타임에 메모리를 동적으로 할당이 필요할 경우 워크로드가 높다면 성능적인 문제가 발생한다. 이를 보완하기 위해 고정된 크기의 블록을 미리 할당해 놓은 것이 메모리 풀(memory pool)이다. 애플리케이션은 런타임에 할당된 블록을 사용함으로써 동적 할당 오버헤드를 최소화할 수 있다. 일반적으로 queue 를 이용해 메모리 풀을 관리한다. 이 코드에선 연결리스트를 이용한 큐를 사용해 메모리 풀을 관리한다. struct packet_t 를 위한 메모리 풀 관련 함수들인 push_packet(), pop_packet(), initialize_packet(), destroy_packet() 이 있다.

1.3 rule_mgmt.c

rule.txt 파일에 rule 들이 있는데, 이를 parsing 하기 위한 함수를 작성한다. rule 파일을 연 후 struct rule_t 타입의 메모리 공간을 할당 받고, 초기화 한다. 그 후 strtok() 함수를 통해 룰 파싱을 진행한다. 전달받은 rule 에 따라 다른 방법으로 parsing 을 해야 한다. 각각 parsing 한 내용을 할당 받은 메모리 풀에 저장한다. 그 후 할당 받은 메모리 풀을 반환하고, 파일을 닫는다.

1.4 main.c (signal handler)

signal 은 비동기적 이벤트를 처리하기 위해 사용된다. 주로 사용되는 signal 은 Ctrl+C 를 눌렀을 때 발생하는 SIGINT, 강제로 프로세스를 종료할 때 사용하는 SIGKILL, SegFault 가 생기면 발생하는 SIGSEGV 등이 있다. 이 코드에선 sig_handler 를 통해 Ctrl+C 가 입력되었을 때 바로 종료하지 않고 reception_destroy_func()를 호출하여 패킷 캡쳐를 종료시키고 메모리 풀을 정리하고 정상적으로 프로그램이 종료될 수 있도록 하였다.

```c
void sigint_handler(int signo)
{
    // destroy packet reception
    ctx.reception_destroy_func();

    // destroy memory pool
    if (destroy_packet_pool() != 0) {
        printf("[ERROR] Failed to destroy the memory pool for raw packets\n");
    }

    // destroy rule tables
    if (destroy_rule_table(&ctx) != 0) {
        printf("[ERROR] Failed to destroy the rule tables\n");
    }

    // turn the running flag off
    ctx.RUN = 0;
}
```

1.5 rule_mgmt.c (Rule table 관리)

struct contet_t 안에 rule table 을 만들기 위한 변수가 정의되어 있다. initialize_rule_table() 함수에는 rule spec 상 protocol 관련하여 IPv4 (0), TCP (1), UDP (2), ICMP (3), HTTP (4) 이렇게 5 가지 종류의 rule 이 있는데, load_rules()에서 각각의 rule 들을 파싱한 후 프로토콜 별 Linked List 형태로 rule 을 관리한다. destroy_rule_table() 함수는 ctx->rule_table[0] ~ [4]까지 돌며 각각에 존재하는 모든 룰들에 대해서 linked list 를 따라가며 메모리를 반환한다. 기존의 load_rules() 함수에서 free(rule) 부분을 제거하고, 프로토콜에 따라 rule table 에 rule 을 추가하는 동작을 하도록 수정한다.

1.6 match_string1.c match_string2.c

일반적으로 문자열 내 서브 문자열을 찾기 위해 strstr() 함수를 사용한다. 하지만 string.h 에서 제공하는 함수들의 경우 기본적으로 Null-termination 을 사용한다. 따라서 문자열 중간에 ₩0 이 존재할 경우 ₩0 뒤는 비교가 불가하다. KMP 알고리즘과 Boyer-Moore 알고리즘을 구현해 더 빠르고 예외사항 없이 문자열 매칭을 할 수 있도록 한다. KMP 알고리즘은 O(N + M)의 시간

복잡도를 가진다. 접두사와 접미사의 개념을 사용해 문자열 비교를 한다. 보이어 무어 알고리즘은 다른 알고리즘과 달리 오른쪽에서 왼쪽으로 진행하며 비교하고 일치하지 않는 문자가 나타나면 정해진 규칙에 따라 오른쪽으로 skip 후 다시 진행한다.

1.7 match_regex.c log_stdout.c log_file.c

다음의 파일에서 문자열 매칭을 쉽게 할 수 있도록 정규 표현식을 사용한다. 정규 표현식은 특정한 규칙을 가진 문자열의 집합을 표현하기 위해 쓰이는 형식언어로, 텍스트 내 특정한 형태나 규칙을 가지는 문자열을 찾기 위한 패턴이다.

2. Simple Host Intrusion Detection System (Hids) 코드 분석

2.0 Introduction

Host Intrusion Detection System (HIDS)는 시스템 내부에 대해서 악의적인 동작이나 상태 변화를 탐지하는 시스템이다. 악의적인 사용자가 시스템에서 어떤 일을 하고 어떤 프로그램을 실행하고 어떤 리소스에 접근하는지 동작 감시를 한다. 또한 악의적인 사용자가 시스템 내부에 어떤 파일을 추가하고 변조하고 삭제하는지 상태 감시를 한다. Simple HID 의 주요 기능은 다음과 같다. 첫째, 관리자가 감시하고자 하는 디렉토리 위치를 지정하면, 해당 디렉토리 및 서브 디렉토리 내 파일들의 상태 변화를 감시한다. 둘째, 이를 위해 디렉토리와 파일들에 대한 구조와 각 오브젝트의 속성 (권한, 크기, 변경된 날짜 등)을 저장한다. 셋째, 오브젝트에 내용이 있다면 특정 형식 (MD5 해시 또는 유사한 해시)의 체크섬도 저장한다. 넷째, 이후 새로운 파일이 추가되면 해당 오브젝트의 정보를 업데이트한다. 다섯째, 기존 파일 (특히, 실행파일)의 변경이 탐지되었을 때 체크섬 값을 통해 무결성이 회손 되었는지 확인한다. 무결성이 회손되었다면 이에 대한 경고 로그를 생성한다. 여섯째, 외부 클라이언트가 시스템에 연결되어 있다면 로그를 클라이언트로 전달하여 사용자가 실시간으로 로그를 확인할 수 있도록 한다. 일곱째, 외부 클라이언트를 통해서 사용자를 시스템 재시작 없이 새로운 디렉토리 위치를 감시할 수 있다.

2.1 entry.c (File Management)

특정 디렉토리가 주어지면 하위 디렉토리와 파일들에 대한 정보를 저장하는 포인터 배열을 생성한다. 정보에는 경로, 크기, 최근 접근 시간, 최근 수정 시간의 정보가 적혀 있다. main.c 함수에서 load_entries() 함수를 호출하고, update_entry_info() 함수와 update_entries() 함수를 이용해 하위 디렉토리, 파일들에 대한 정보를 저장한다. 파일이나 디렉토리가 삭제되면

release_entries() 함수가 호출되어 할당 받은 저장공간을 해제한다. 반대로 추가되면 add_entry() 함수가 호출되어 저장공간을 추가로 할당 받아 저장한다.

2.2 hash.c

해시는 임의의 길이를 가지는 데이터를 고정된 길이의 데이터로 변환하는 방법이다. 해시 함수를 통해 임의의 길이를 가지는 데이터를 고정된 길이의 데이터로 변환한다. 해시는 해시 테이블을 기반으로 빠른 탐색을 위해 주로 사용된다. update_entry_info() 함수에서 hash_func() 함수를 호출해 디렉토리와 파일 이름을 해시 함수에 적용시켜 struct entry 에 hash[] 배열에 저장된다.

2.3 watch.c (File Event Monitoring)

Inotify API 는 리눅스 파일 시스템에서 발생되는 다양한 이벤트들을 모니터링하기 위한 API 다. 보안, 성능 등 다양한 목적으로 사용자 공간을 모니터링할 수 있다. Inotify 를 이용한 모니터링 절차는 첫째, inotify_init 을 사용하여 파일 디스크립터를 연다. 둘째, 하나 이상의 감시를 추가한다. 셋째, 이벤트를 대기한다. 넷째, 이벤트를 처리한 후 대기 상태로 돌아간다. 다섯째, 더 이상 활성화된 감시가 없거나 어떤 신호를 수신하는 경우 파일 디스크립터를 닫고 정리한 후 종료한다. Watch.c 에 파일 추가/변경/삭제하는 이벤트를 모니터링하는 코드를 작성한다. Entry 정보의 변경을 출력하고, 컨텐츠가 바뀌었다면 MD5 해시에 업데이트 한다. Signal handler 도 추가해 Ctrl + C 를 누르면 모니터링을 종료하도록 한다.

2.4 thpool.c

프로세스는 실행되고 있는(memory 에 load 된) 프로그램이다. 스레드는 프로세스 내에서 실행되는 흐름의 단위로, 프로세스의 자원을 공유하며 할당 받은 자원을 이용한다. 동시성은 싱글 코어에서 멀티 쓰레드를 동작 시키기 위해 여러 쓰레드가 번갈아 가며 실행되는 방식이고, 병렬성은 멀티 코어에서 멀티 쓰레드를 동작 시키기 위한 방식으로 한 개 이상의 쓰레드가 각 코어들에서 동시에 실행되는 것을 의미한다. 이 프로그램에서 여러 디렉토리를 한 번에 모니터링하기 위해 멀티 쓰레드를 사용한다.