RaceGuard: Kernel Protection From Temporary File Race Vulnerabilities

유효곤 정수환

목차

- Introduction
- Temporary File Race Vulnerabilities
- RaceGuard: Dynamic Protection from Race Attacks
- Security Testing
- Compatibility Testing
- Performance Testing
- Related Work
- Conclusions

Introduction

Race Condition

한정된 자원을 동시에 사용하려는 여러 프로세스가 경쟁을 벌이는 현상

Introduction

정상 동작

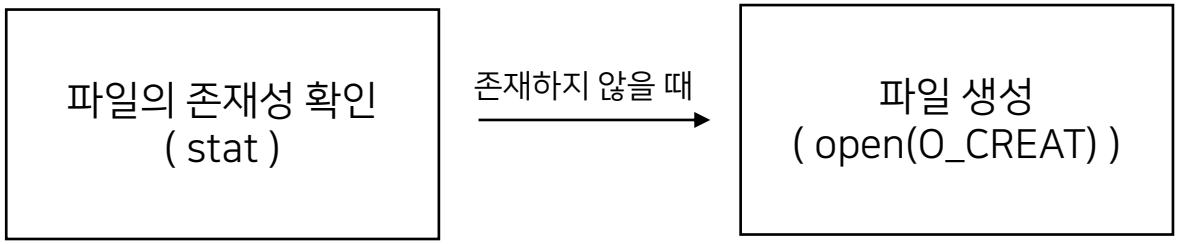
레이스 컨디션 발생

공격자가 파일 생성

O_CREAT 는 " 존재하지 않을 경우 생성하여 open한다 " 는 의미
파일이 이미 존재하므로 해당 파일 open

Introduction

mktemp



만약 " 중요한 파일 " 에 심볼릭 링크가 걸려있으면??

ex) /etc/passwd

Introduction

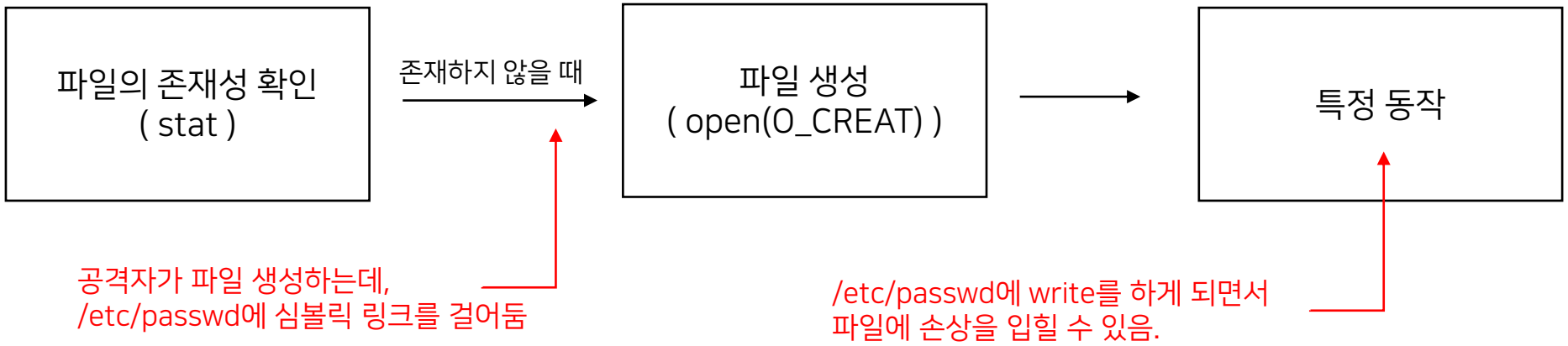
- 해결 방법 1.
  - open(O_CREAT) 대신 open(O_CREAT | O_EXCL) 사용 => mkstemp 함수
    - O_EXCL : 존재하지 않을 경우 FAIL
  - 그런데 O_EXCL을 사용하면 의도한 동작이 불가능한 경우들이 존재
    - ex) Apache
  - 그래서 mkstemp가 있음에도 mktemp를 사용함

Introduction

- 해결 방법 2.
  - RaceGuard 이용
    - stat() => open() 시, stat에서 확인한 파일의 유무랑 open()에서 확인한 파일의 유무랑 다를 시, open() 함수 도중에 abort()
    - 커널 단에서 Race condition을 이용한 exploit 시도를 탐지함

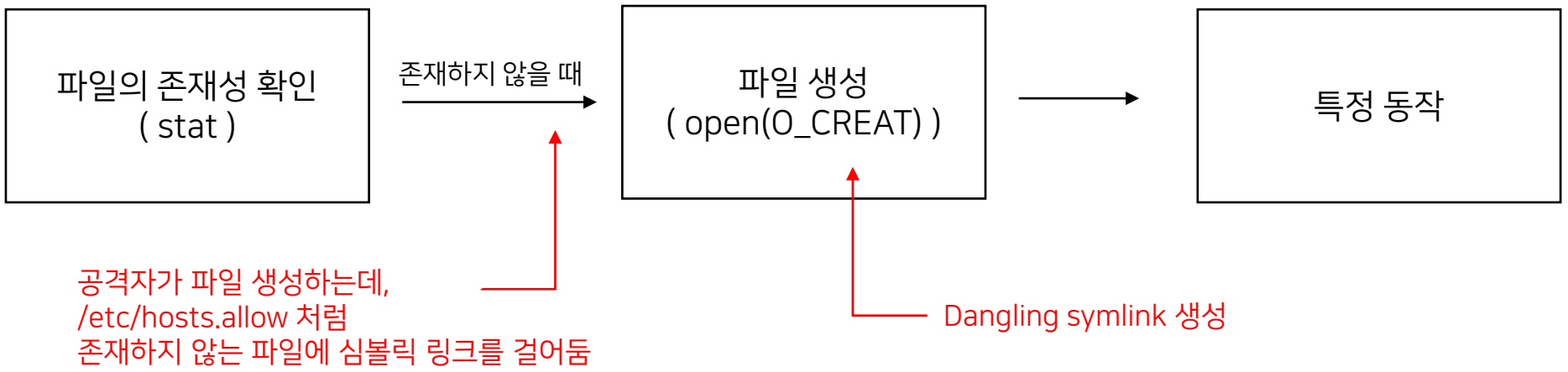
Temporary File Race Vulnerabilities

- 1. 심볼링 링크를 이용한 중요한 파일 손상



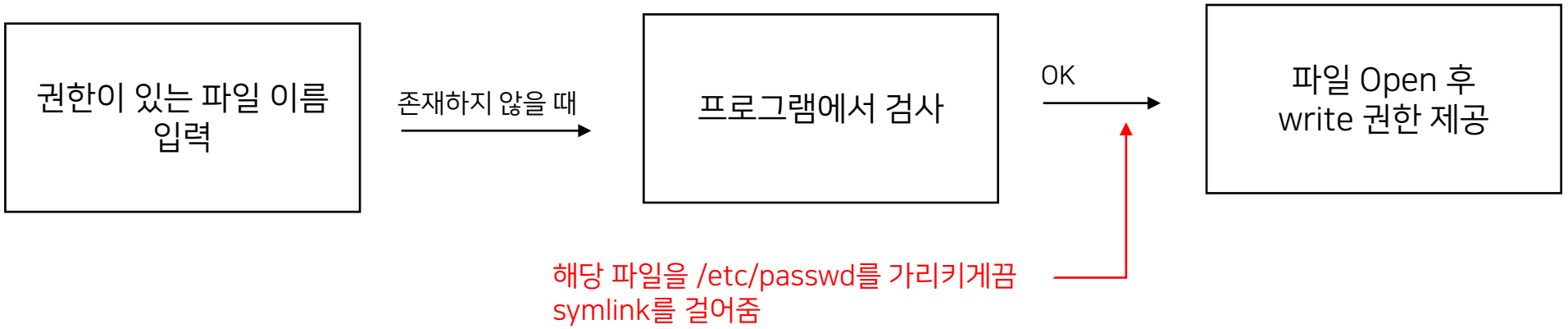
Temporary File Race Vulnerabilities

- 2. Dangling symlink



Temporary File Race Vulnerabilities

- 3. file swap
  - 공격 대상 : SUID를 가진 프로그램
    - 사용자의 권한을 확인하고 특정 파일에 데이터를 write할 수 있게 해주는 프로그램



RaceGuard
RaceGuard Design

- RaceGuard가 race condition attack을 탐지하는 방법
  - stat()
    - 파일이 존재하지 않았다면 파일 명을 캐시에 저장한다
  - open(O_CREAT)
    - 파일이 존재한다면 캐시에 해당 파일 이름이 저장되어 있는지 확인
      - 저장되어 있다면 공격임을 인식하고 abort() !!

RaceGuard
RaceGuard Design

- RaceGuard Algorithm
  - cache : list of the file name

  1. stat 에서 Non-exist임을 확인
     - cache에 파일 명 저장
  2. open시 이미 존재한다고 확인되면
     - RaceGuard cache에 파일 명이 저장되어있는지 확인
  3. 만약 정상적으로 동작하여 파일이 생성되었다면
     - RaceGuard cache에 파일 명이 있으면 cache에서 데이터 제거

RaceGuard
RaceGuard Design

- “dangling – symlink”
  - symlink가 있으면 파일명과, 심볼링 링크로 가리키는 파일명 모두 확인
    - 만약 두 동작이 다른데, 파일 명이 cache에 있다면 -> Abort() !!
- “file swap”
  - RaceGuard로 해결 못함.

RaceGuard
Implementation & Cache Management Policy

- RaceGuard가 관찰하는 함수
    - 1. 파일의 존재성 확인 → stat, lstat, access, newstat, newlstat
    - 2. 파일 생성 → open , create, mkdir, mknod, link, symlink rename, bind
    - 3. create, remove process → fork , exit

RaceGuard
Implementation & Cache Management Policy

- cache management policy
    - RaceGuard에서 상당히 흥미로운 부분
  - cache에 데이터를 정확히 관리해야만 정상적인 과정임에도 race condition을 탐지했다고 abort하는 일이 벌어지지 않음!!

RaceGuard
Implementation & Cache Management Policy

- cache management policy
  - cache의 크기 : 7
    - stat -> open 같은 간단한 과정에서 발생
    - 굳이 entry가 많은 필요가 없음
  - FIFO scheduling 사용
    - 1. cache : circular buffer
    - 빈공간을 찾아서 데이터 저장
    - 만약, 공간이 없다면 가장 최근에 넣은 위치 이전의 공간의 데이터를 꺼냄
      - 이렇게 해야 빠르고, 가장 최근의 데이터를 꺼내는 것을 피할 수 있음

RaceGuard
Implementation & Cache Management Policy

- cache management policy
  - 부모 process의 cache 정보도 자식 process에게 공유되어야 함.
  - 만약 부모 process에서 evict 했다면 자식 process에서도 evict 해줘야 함.

RaceGuard
Implementation & Cache Management Policy

- 한계점
  - Race Guard의 대상 함수의 범위 지정
    - mkdir, link 등의 명령어는 내부에서 자체적으로 존재성 검증이 있음
    - 만약 chmod를 이용해서 존재성을 판단하는 프로그램이라면?? (검증 불가)
  - entry가 적음
    - 만약 $PATH에 있는 모든 파일의 존재성을 확인한 후에, open하려고 한다면?
      - 검사하는 개수가 7개가 넘어가면 정상적인 작동 불가

Security Testing

- Test 방법
  - Race를 발생시켜서 attacker가 공격하는 상황을 만들고, 이를 RaceGuard가 잘 감지하고, 방지하는 지 확인
- Mktemp 변형
  - 1. Pause the Program : mktemp 중 30초간 pause
  - 2. Print the Created File Name: tmp file의 이름을 출력
  - Attacker의 공격을 수월하게 하기 위함

Security Testing

- Mktemp를 변형시킨 것이 security robustness에 영향을 주지 않기에 raceguard를 타당하다 할 수 있음
- we attacked four programs:
  - RCS version 5.7 [15],
  - rdist Version 6.1.5 [14],
  - sdiff - GNU diffutils version 2.7 [2],
  - shadowutils-19990827

Security Testing

```
 <<< set up our target to overwrite >>>
[steve@reddwarf .elm]$ echo "please dont hurt me" > ~/dont_hurt_me

        <<< run our vulnerable program >>>
[steve@reddwarf .elm]$ LD_PRELOAD=~/libmktemp.so rcsdiff -u elmrc > /dev/null
===================================================================
RCS file: RCS/elmrc,v
retrieving revision 1.3
unsafe_mktemp[20038]: ImmunixOS unsafe mktemp - about to pass back /tmp/T0LZ388D
        <<< In another shell, do "ln -s ~/dont_hurt_me /tmp/T0LZ388D" >>>
diff -u -r1.3 elmrc

        <<< and what does our target now contain? >>>
[steve@reddwarf .elm]$ cat ~/dont_hurt_me | head -5
#
# .elm/elmrc - options file for the ELM mail system
#
# Saved automatically by ELM 2.5 PL1 for Steve Beattie
#
```

Figure 1 Successful Attack Against RCS Without RaceGuard

Security Testing

```
 <<< set up our target to overwrite >>>
[steve@kryten .elm]$ echo "please dont hurt me" > ~/dont_hurt_me

        <<< run our vulnerable program >>>
[steve@kryten .elm]$  LD_PRELOAD=~/libmktemp.so rcsdiff -u elmrc > /dev/null
===================================================================
RCS file: RCS/elmrc,v
retrieving revision 1.3
unsafe_mktemp[1456]: ImmunixOS unsafe mktemp - about to pass back /tmp/T0POjIdZ
        <<< In another shell, do "ln -s ~/dont_hurt_me /tmp/T0POjIdZ" >>>
/usr/bin/co: Killed
rcsdiff aborted

        <<< and what does our target now contain? >>>
[steve@kryten .elm]$ cat ~/dont_hurt_me
please dont hurt me

        <<< RaceGuard intrusion alert in syslog >>>
[steve@kryten .elm]$ dmesg | tail -1
Immunix: RaceGuard: rcsdiff (pid 1458) killing before opening /tmp/T0POjIdZ!
```

Figure 2 Failed Attack Against RCS With RaceGuard

Compatibility Testing

- 일반적인 software에 적용하여 test를 진행하였음
- 그 과정에서 다음과 같은 소프트웨어에서 몇가지 문제점을 발견하였음.
- Mozilla web/mail client.
  - Mozilla는 tmp파일을 자주 사용하는 데, re-use 과정중에 false positive(race가 아닌 데 race라 보고하는 경우)가 발생함
  - File creation 완료시 Cache clearing 과정을 추가하여 해결
- Red Hat Linux
  - re-boots시 /dev/random의 entropy pool을 보존하는 스크립트에서 발생
  - Parent process에서 probe과정을 거치고 child process에서create하는 과정이 존재

Compatibility Testing

- CVS Checkout
  - probe("foo") ⇒ chdir("bar") ⇒ creat("foo")
  - 위처럼 director가 변경이 되었는 데, 파일명이 동일한 경우에 대해서 raceguard false positive…
  - 절대경로로 변경하여 해결
- VMWare virtual Machine Emulation system
  - Vmware 동작중에 stat(NULL)를 하는 데, raceguard에서 오류 발생
  - 따로 처리하여 해결

Performance Testing

- Raceguard를 적용했을 때와 적용하지 않았을 때를 비교
- Latency를 측정하였고, Overhead가 작으면 작을 수록 기존의 성능에 영향을 주지 않는 것이기에 긍정적으로 평가 가능
- Microbenchmark
  - 각 기능 별로 얼마나 overhead가 발생하는 지 확인
- Macrobenchmark
  - 일반적인 프로그램을 실행했을 때, overhead가 얼마나 발생하는 지 확인

Performance Testing
Microbenchmarks

- Stat() non existent file : 기존에는 단순 stat, raceguard 적용시 cache작업
- Open non-existent file : 기존에는 단순 open, raceguard 적용시 cache작업
- Fork : cache push/clear 가 아닌 다른 동작
- 10000회 loop를 통해 overhead 측정

Table 1: RaceGuard Microbenchmark Results

System Call	Without RaceGuard	With RaceGuard	% Overhead
Stat non-existent file	4.3 microseconds	8.8 microseconds	104%
Open non-existent file	1.5 milliseconds	1.44 milliseconds	-4%
Fork	161 microseconds	183 microseconds	13%

Performance Testing
Macrobenchmarks

- Khernel-stone
  - single-processor 700 MHz Athlon machine with 128 MB of RAM.
- Apache web-server
  - case was a dual processor 700 MHz Pentium III, 256 KB of cache, 256 MB of RAM, and a 100 Mbit network to the client.

Performance Testing
Macrobenchmarks

Table 2: Khernelstone Macrobenchmark, in Seconds

	Real Time	User Time	System Time
Without RaceGuard	10,700	8838	901
With RaceGuard	10,742	8858	904
%Overhead	0.4%	0.2%	0.3%

Related Work

- The study of temporary file race vulnerabilities is old
  - But, temporary file race vulnerabilities were found in core Internet infrastructure tools such as Apache in 2001
- Bishop's seminal paper [6], TOCTTOU에 대한 개념을 제시
  - 정적으로 TOCTTOU vuln을 찾는 것을 불가능
  - Raceguard와 비슷한 race condition guard를 제시하였으나 overhead가 너무 커서 사용되지 않음
- "Solar Designer" [10] takes a different approach to combating temporary file race vulnerabilities.
  - 파일에 권한 설정을 달리해줘서 race를 해결하고자 하는 방법을 제시
  - 하지만, 권한이 꼬이는 결함이 생김..

Conclusions

mktemp 등 여러 임시파일 생성 프로그램에서 발생한 race condition은 보안에서 주목되는 문제였다.

Raceguard를 이용하게 되면, binary가 안전하지 않은 방법을 사용하거나(i.e. mktemp) 안전하지 않은 library를 사용해도 이와 상관없이 race condition attack을 방지한다.

또한, overhead가 굉장히 적어 linux kernel 등에 적용 가능하다

Demo – race condition

race1.c > main(int, char * [])

```c
#include <stdio.h>
#include <pthread.h>
#include <time.h>

int count = 0;


void print_count(char* data){
    for (int i = 0; i < 10; i++)
    {
        printf("%s : %d\n", data, count);
        count++;
    }

}

int main(int argc, char *argv[])
{
    pthread_t thread1, thread2;
    char threadname[2][20]={
        {"thread1"},
        {"thread2"}
    };

    pthread_create(&thread1, NULL, print_count, threadname[0]);
    pthread_create(&thread2, NULL, print_count, threadname[1]);

    sleep(1);
    printf("count : %d\n", count);
    return 0;
}
```

```
ugonfor@DESKTOP-72IQ4C9:/windir/c/Users/ryuhyogon/Desktop/raceguard_demo$ ./race1
thread1 : 0
thread1 : 1
thread2 : 1
thread2 : 3
thread2 : 4
thread2 : 5
thread2 : 6
thread2 : 7
thread2 : 8
thread2 : 9
thread2 : 10
thread1 : 2
thread1 : 12
thread1 : 13
thread1 : 14
thread1 : 15
thread1 : 16
thread1 : 17
thread1 : 18
thread2 : 11
count : 20
```

Source code : https://bit.ly/3hz3DfN

test1.c

test1.c > ...

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <fcntl.h>
#include <string.h>

#define MAX 0x100

int main(void){
    char* name = NULL;
    char template1[MAX] = "./raceXXXXXX";
    char text[100] = "This is text";
    name = mktemp(template1);
    struct stat buf;

    int check = stat(name, &buf);
    int fd = 0;
    if( check == -1 ){
        fd = open(name, O_CREAT | O_WRONLY, S_IRWXU|S_IRWXG|S_IRWXO);
        write(fd, text, strlen(text));
        close(fd);
    }
    else{
        printf("already exist file name %s\n", name);
    }

    return 0;
}
```

Source code : https://bit.ly/3hz3DfN

Demo – raceguard - off

- Echo “don’t touch this file” > donttouch
- export LD_PRELOAD=$PWD/libhook.so
- ./test1
- 다른 터미널 : ln donttouch –s ./race??????

```
ugonfor@DESKTOP-72IQ4C9:/windir/c/Users/ryuhyogon/Desktop/raceguard_demo$ ./test1
[!] mktemp(./raceXXXXXX) = ./raceUdEwLy
[!] open(./raceUdEwLy, 65, 511) =
[!] sleep(10)
3
```

Source code : https://bit.ly/3hz3DfN

Demo – raceguard - on

- Echo “don’t touch this file” > donttouch
- export LD_PRELOAD=$PWD/libraceguard.so
- ./test1
- 다른 터미널 : ln donttouch –s ./race??????

```
ugonfor@DESKTOP-72IQ4C9:/windir/c/Users/ryuhyogon/Desktop/raceguard_demo$ LD_PRELOAD=$PWD/libraceguard.so
ugonfor@DESKTOP-72IQ4C9:/windir/c/Users/ryuhyogon/Desktop/raceguard_demo$ ./test1
[!] mktemp(./raceXXXXXX) = ./racec9iN5c
[!] stat(./racec9iN5c, struct stat statbuf) = -1
[+] cache = ./racec9iN5c
[!] open(./racec9iN5c, 65, 511) =
[!] sleep(10)
=======raceguard START=======
[?] cache check (./racec9iN5c == ./racec9iN5c)
[X] ./racec9iN5c exist
===== race attack occurred! =====
========= exit process ==========
```

Source code : https://bit.ly/3hz3DfN

감사합니다

Source code : https://bit.ly/3hz3DfN