Syzkaller 를 이용한 리눅스 커널 네트워크 서브시스템 퍼징에 관한 연구

송수빈 [1], 박민경 [2], 권태경 [3]
[1] 서울대학교 컴퓨터공학부 석사과정
[2] The University of Texas at Dallas Cyber Security Research and Education Institute 박사후과정
[3] 서울대학교 컴퓨터공학부 교수

sbsong66@snu.ac.kr, minkyung.park@utdallas.edu, tkkwon@snu.ac.kr

A Study on Fuzzing the Linux Kernel Networking Subsystem Using Syzkaller

Su-Bin Song[1], Min-Kyung Park[2], Tae-Kyoung Kwon[3]
[1]Dept. of Computer Science and Engineering, Seoul National University
[2]Cyber Security Research and Education Institute, The University of Texas at Dallas
[3]Dept. of Computer Science and Engineering, Seoul National University

요 약

본 연구에서는 커널 퍼저인 Syzkaller 를 사용하여 리눅스 커널의 네트워크 서브시스템을 퍼징하고 그 결과를 분석하여, 높은 커버리지를 달성하기가 왜 어려운지 분석하고 이를 개선하기 위한 방법들을 제안한다. 첫 번째 실험에서는 TCP 및 IPv4 소켓과 관련된 시스템 콜 및 매개변수만 허용하여 리눅스 커널 네트워크 퍼징을 진행하고, 두 번째 실험에서는 Syzkaller 가 지원하는 모든 시스템 콜을 포함하도록 범위를 확장한다. 첫 번째 실험 결과, 퍼징 시작 약 55 시간만에 TCP 연결 수립에 성공하였다. 두 번째 실험 결과, 첫 번째 실험보다 전반적인 커버리지와 라우팅 서브시스템의 커버리지는 개선되었으나 TCP 연결 수립에는 실패하였다. TCP 연결 수립을 위해서는 서버의 IP 주소 및 포트번호를 클라이언트가 무작위 입력 생성을 통해 맞혀야 하는데, 이 과정에서 시간이 오래 걸리기 때문에 연결 수립이 쉽게 이루어지지 않는 것으로 분석된다. 추가적으로, 본 연구에서는 TCP 연결 수립을 쉽게 하기 위한 하이브리드 퍼징, IP 패킷 포워딩 허용, 패킷 description 없이 퍼징 등 Syzkaller 를 이용하여 리눅스 커널 네트워크 서브시스템을 더 효율적으로 퍼징할 수 있는 방법들을 제안한다.

1. 서론

Syzkaller 는 무작위 순서 및 매개변수로 시스템 콜들을 호출하여 OS 커널을 퍼징하는 커널 퍼저이다. Syzkaller 를 이용하여 리눅스 커널의 네트워크 서브시스템을 퍼징하는 것은 가능하나, 네트워크 서브시스템의 특성상 다른 서브시스템들에 비해 퍼징 과정에서 다양한 어려움이 존재한다. 본 연구에서는 Syzkaller 를 이용해서 리눅스 커널의 네트워크 서브시스템을 퍼징하는 실험을 진행하고, 높은 커버리지를 달성하기 어려운 이유를 분석한다. 또한 분석 결과에 기반하여 퍼징 과정의 효율성을 높이기 위한 방법들을 제안한다.

2. 배경지식

1) Syzkaller

Syzkaller[1]는 구글에서 제작한 커버리지 기반 커널 퍼저로, 현재까지 리눅스 커널에서 수백여 개가 넘는 버그를 발견하는 데 사용되었다 [2]. Syzkaller 는 여러 개의 시스템 콜들로 이루어진 프로그램을 입력으로 받아 실행하며, 개발자들이 직접 작성한 “syscall description” 안에 나타나 있는 시스템 콜 함수들 사이의 의존 관계를 활용해 효과적으로 퍼징을 진행한다. 하지만 이러한 syscall description 은 전부 사람이 직접 작성하기 때문에, 새로운 커널 서브시스템을 퍼징하고 싶다면 해당하는 syscall description 들을 새롭게 작성해야 한다는 단점이 있다. 또한 closed-source 커널

의 경우 시스템 콜 API 가 공개되어 있지 않으면 syscall description 을 작성할 수가 없다는 한계가 있다.

2) Syzkaller 를 이용한 리눅스 커널 네트워크 서브시스템 퍼징

Syzkaller 는 여러 대의 독립된 가상머신을 실행하여 각 머신 안에서 프로그램을 실행하여 퍼징을 진행한다. 따라서 구조상 외부에서 네트워크 패킷을 직접 보내서 퍼징을 할 수가 없다. 대신 Syzkaller 에서는 리눅스 네트워크 서브시스템 퍼징을 위해서 커널의 TAP 인터페이스를 활용하여, 마치 외부에서 오는 것처럼 이더넷 프레임을 커널에 주입한다 [3], [4]. TAP 인터페이스는 리눅스 커널의 가상 네트워크 인터페이스로, 유저 스페이스 프로그램이 TAP 디바이스 파일을 열어서 프레임을 직접 읽고 쓸 수 있다. 프레임을 디바이스 파일에 쓰면 해당 프레임은 마치 외부에서 온 것처럼 커널이 TAP 인터페이스에서 읽어서 처리하게 되고, 반대로 커널이 TAP 인터페이스를 통해 내보낸 프레임은 유저 스페이스 프로그램이 디바이스 파일을 읽음으로써 받을 수 있다. TAP 인터페이스는 OpenVPN 등 터널링 기법을 사용하는 애플리케이션에서 주로 사용된다.

Syzkaller 에서는 syz_emit_ethernet()이라는 pseudo-syscall [5] 함수를 이용하여 TAP 디바이스에 프레임을 써서 외부에서 프레임이 커널로 들어오는 상황을 재현한다. 이를 통해 하나의 퍼저 프로세스 안에서 서버 측과 클라이언트 측의 역할을 모두 하며 퍼징을 진행할 수 있다. 또한 시스템 콜 함수들을 설명하는 syscall description 이 있듯이 네트워크 패킷의 구조 역시 프로토콜 스펙에 따라 description 으로 작성되어 있다.

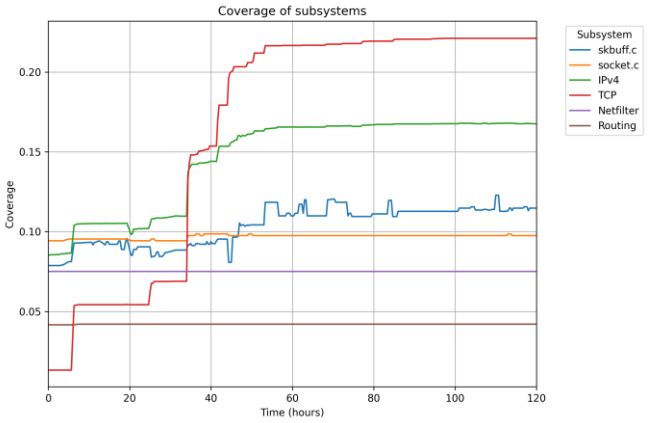
3. Syzkaller 를 이용한 리눅스 커널 네트워킹 서브시스템 퍼징 실험

Syzkaller 를 이용하여 리눅스 커널 네트워킹 서브시스템을 얼마나 효과적으로 퍼징할 수 있는지 알아보기 위해 두 차례에 걸쳐 실험을 진행하였다. 두 실험은 모두 Intel® Core™ i5-7500 프로세서와 8GB 메모리, Ubuntu 22.04 운영체제의 호스트 머신에서 진행되었으며, 3 개의 x86-64 QEMU 가상머신을 각각 2 개의 CPU 와 2GiB 메모리 환경으로 설정하여 Debian Bullseye 운영체제로 리눅스 커널 6.6.1 버전을 실행하였다. 두 실험 각각 120 시간동안 퍼징을 진행하였고, 퍼징 진행 과정에서 퍼저의 raw PC trace 를 매 20 분마다 기록하여 분석에 활용하였다. 또한 퍼저가 네트워크 서브시스템의 커버리지를 향상시키는 데 집중할 수 있도록 특정 파일들(net/ipv4/, net/core/, net/socket.c)에만 커버리지 필터를 적용하여 퍼징을 진행했다. 두 실험의 서로 다른 설정 내용은 1)과 2)에서 설명한다.

본 연구의 구체적인 실험 설정, 실험에 사용한 코드, 그리고 실험 결과의 일부는 https://github.com/lemonshushu/ask2024-syzkaller-linux-nwsub 에 공개되어 있다.

1) TCP 및 IPv4 소켓 관련 시스템 콜들만 허용하여 실험

첫 번째로, 가장 일반적인 인터넷 통신에 사용되는 TCP 및 IPv4 서브시스템에 집중하여 퍼징을 진행하기 위해 해당 프로토콜의 소켓들을 생성하고 이용하는 시스템 콜들만 허용하여 퍼징을 진행하였다. 허용된 시스템 콜 및 pseudo-syscall 들은 socket$inet_tcp, bind$inet, listen, accept$inet, syz_emit_ethernet, syz_extract_tcp_res$inet 이다.



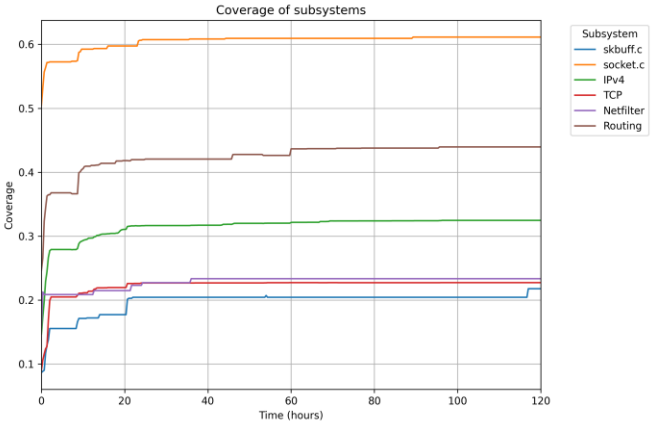
(그림 1) TCP 및 IPv4 소켓 관련 시스템 콜들만 허용하여 실험한 결과.

그림 1 은 실험 결과를 나타낸다. 범례의 각 서브시스템에 포함되는 파일 목록은 Appendix 에 첨부한다. 결과 상 가장 눈에 띄는 점은 세 가지이다. 첫 번째, 모든 서브시스템의 커버리지가 25%에도 미치지 못할 정도로 전반적으로 낮다. 두 번째, TCP 서브시스템의 코드 커버리지가 약 35 시간 경과 시점에 급격히 증가한다. 분석 결과, 이는 서버 측에서 열어 둔 소켓이 bind 된 IP 주소 및 포트번호를 클라이언트 측에서 무작위 입력 생성(mutation)을 통해 일치시키는 데 성공하여, 클라이언트가 전송한 패킷이 서버 소켓에서 정상적으로 받아들여지며 새로운 코드 부분이 커버되었기 때문임이 확인되었다. 약 55 시간 경과 시점에서는 TCP 3-way handshake 에 성공하여 연결이 수립된 것을 확인하였다. 세 번째, socket.c, Netfilter, Routing 서브시스템의 커버리지는 전체 시간동안 매우 낮은 수준으로 거의 변동이 없다. 이는 실험에서 TCP, IPv4 와 관련된 매개변수들이 포함된 시스템 콜들로 범위를 제한했기 때문에 Netfilter, Routing 서브시스템의 상태를 바꿀 수 있는 매개변수를 포함한 시스템 콜들이 호출

되지 않았기 때문이다.

2) 모든 시스템 콜들을 허용하여 실험

그림 1 의 실험 결과에서 나타난 문제점들을 해결하기 위해, 시스템 콜의 종류 및 매개변수를 제한하지 않고 Syzkaller 가 기본적으로 제공하는 모든 시스템 콜을 허용하여 동일한 조건에서 실험을 진행하였다. 그 결과는 그림 2 와 같다.



(그림 2) 모든 시스템 콜들을 허용하여 실험한 결과.

그림 1 의 실험 결과와 비교하면, TCP 서브시스템을 제외하고 전반적으로 커버리지가 높은 것이 확인된다. 또한 그림 1 에서 나타난 TCP 서브시스템에서의 급격한 커버리지 증가는 일어나지 않는다. 모든 시스템 콜들을 허용하면서 실행되는 코드의 범위가 전체적으로 증가했고, 따라서 집중적으로 mutation 이 시도되는 코드의 범위 역시 증가했기 때문에 mutation 을 통한 우연의 일치로 실험 1)과 같은 결과가 나타나기에는 훨씬 더 오랜 시간이 필요할 것으로 분석된다. 한편 Routing 서브시스템의 커버리지는 실험 1)보다 눈에 띄게 증가하였는데, 이는 추가적으로 허용된 시스템 콜들 및 매개변수들을 통해 Netlink 소켓을 이용해 라우팅 테이블에 조작이 이루어졌기 때문으로 분석된다.

4. 고찰 및 제언

1) TCP 연결 수립의 어려움

위의 실험 결과에서 볼 수 있듯이 Syzkaller 퍼징을 통해 TCP 연결 수립에 성공하는 데는 오랜 시간이 걸린다. 구체적으로 3-1)에서는 서버에서 bind 한 IP 주소 및 포트 번호를 클라이언트가 mutation 을 통해 맞히는 데 35 시간 정도가 걸렸으며, TCP 3-way handshake 를 성사하는 데에는 20 시간 정도가 추가적으로 소요되었다.

물론 이와 같은 "의존성"은 대부분 Syzkaller 의 syscall description 이나 pseudo-syscall 로 표현되어 있으나, 그와 같이 의존성을 표시하게 되면 퍼저가 에러 상황과 같이 다양한 코드 경로를 실행하지 못하고 항상 "옳은" 경로로만 탐색하게 된다는 문제점이 있다. 따라서 현재 상황과 같이 퍼저가 다양한 입력을 생성하여 오류 상황과 같은 다양한 코드 경로에 도달할 수 있도록 하되, TCP 연결을 수립할 수 있는 조건을 좀 더 빠르게 찾아낼 수 있도록 symbolic execution 등 white-box 퍼징 기법을 일부 접목하는 하이브리드 퍼징 접근법을 적용해볼 수 있다.

2) IP 패킷 포워딩 부분 코드의 커버리지

실험의 결과로 나온 파일별 커버리지를 분석해본 결과, IP 패킷 입력 및 출력 부분 코드들은 커버되었지만, net/ipv4/ip_forward.c 를 포함하여 패킷 포워딩 부분 코드는 전혀 커버되지 않았음이 확인되었다. 근본적인 원인은 리눅스 커널의 IP 포워딩 관련 기본 설정이 포워딩을 허용하지 않는 것으로 되어 있기 때문이다 [6]. 또한, 패킷 포워딩 설정을 허용하더라도, 패킷의 목적지 IP 주소와 일치하는 경로(route)가 라우팅 테이블에 없다면 패킷 포워딩은 일어나지 않는다. 따라서, 포워딩이 일어날 수 있도록 해당되는 경로들을 라우팅 테이블에 추가해주는 작업이 필요하다. 리눅스 커널의 IP 서브시스템에서 IP 패킷 포워딩 관련 코드가 큰 비중을 차지하는만큼, 해당 부분 코드가 추가적으로 커버된다면 기존에 찾지 못했던 새로운 버그가 퍼징을 통해 발견될 가능성이 크다.

3) 패킷 description 없이 퍼징

현재 Syzkaller 에서는 syscall description 의 일부분으로 각 네트워크 프로토콜 패킷의 구조를 개발자들이 직접 작성하여 퍼징에 이용하고 있다. 하지만 리눅스 커널에 수많은 네트워크 프로토콜이 존재하고 각 프로토콜 스펙이 복잡한만큼, 사람이 일일이 패킷의 description 을 작성하고 유지보수하는 것은 쉽지 않은 일이다. 이러한 문제를 해결하기 위해, 실제 네트워크 프로토콜의 트래픽을 수집, 이를 기반으로 변형하여 퍼징의 입력으로 사용하는 방안을 검토할 수 있다. 이러한 방법은 실제로 AFLNet[7], StateAFL[8] 등의 네트워크 프로토콜 퍼저에 사용되고 있다.

5. 결론

본 연구에서는 Syzkaller 를 이용하여 직접 리눅스 커널의 네트워크 서브시스템을 퍼징하고 그 결과를 분석, 고찰하였다. 무작위 입력 생성을 통해서는 TCP 연결 수립 등 네트워크 프로토콜의 상태 이행에 알맞은 조건들을 충족하기가 어려워 상태 이행에 오랜 시간이 걸린다. 이를 해결하기 위해 white-box 퍼징 기법을 접목하는 하이브리드 퍼징 방식의 접근법이 요구된다. 이외에도 패킷 포워딩 허용, 패킷 description 없이 실제 네트워크 트래픽 기반으로 입력 패킷 생성하기 등의 개선방안들은 추가 연구과제로 남는다.

ACKNOWLEDGEMENT

이 논문은 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(No. RS-2023-00220985).

본 연구는 과학기술정보통신부 및 정보통신기획평가원의 대학 ICT 연구센터육성지원사업의 연구결과로 수행되었음 (IITP-2024-2021-0-02048)

이 논문은 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(NRF-2022R1A2C2011221)

참고문헌

[1] “google/syzkaller.” Google, Apr. 19, 2024. Accessed: Apr. 19, 2024. [Online]. Available: https://github.com/google/syzkaller

[2] “syzkaller-bugs - Google Groups.” Accessed: Apr. 19, 2024. [Online]. Available: https://groups.google.com/g/syzkaller-bugs

[3] “syzkaller/docs/linux/external_fuzzing_network.md at master · google/syzkaller,” GitHub. Accessed: Apr. 19, 2024. [Online]. Available: https://github.com/google/syzkaller/blob/master/docs/linux/external_fuzzing_network.md

[4] A. Konovalov, “🔍 Looking for Remote Code Execution bugs in the Linux kernel,” Andrey Konovalov. Accessed: Apr. 19, 2024. [Online]. Available: https://xairy.io/articles/syzkaller-external-network

[5] “syzkaller/docs/pseudo_syscalls.md at master · google/syzkaller,” GitHub. Accessed: Apr. 19, 2024. [Online]. Available: https://github.com/google/syzkaller/blob/master/docs/pseudo_syscalls.md

[6] “IP Sysctl — The Linux Kernel documentation.” Accessed: Apr. 20, 2024. [Online]. Available: https://www.kernel.org/doc/html/v6.6/networking/ip-sysctl.html

[7] V.-T. Pham, M. Böhme, and A. Roychoudhury, “AFLNET: A Greybox Fuzzer for Network Protocols,” in *2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST)*, in ICST’20. Oct. 2020, pp. 460–465. doi: 10.1109/ICST46399.2020.00062.

[8] R. Natella, “StateAFL: Greybox fuzzing for stateful network servers,” *Empir. Softw Engg*, vol. 27, no. 7, Dec. 2022, doi: 10.1007/s10664-022-10233-3.

Appendix

그림 1, 그림 2 에서 범례의 각 서브시스템에 포함되는 파일 목록

skbuff.c : net/core/skbuff.c
socket.c : net/socket.c
IPv4 : net/ipv4/
TCP : net/ipv4/tcp*
Netfilter : net/ipv4/netfilter/
Routing : net/ipv4/fib_*

* “tcp*”와 “fib_*”은 파일명이 “tcp” 혹은 “fib_”로 시작하는 파일들을 의미한다.