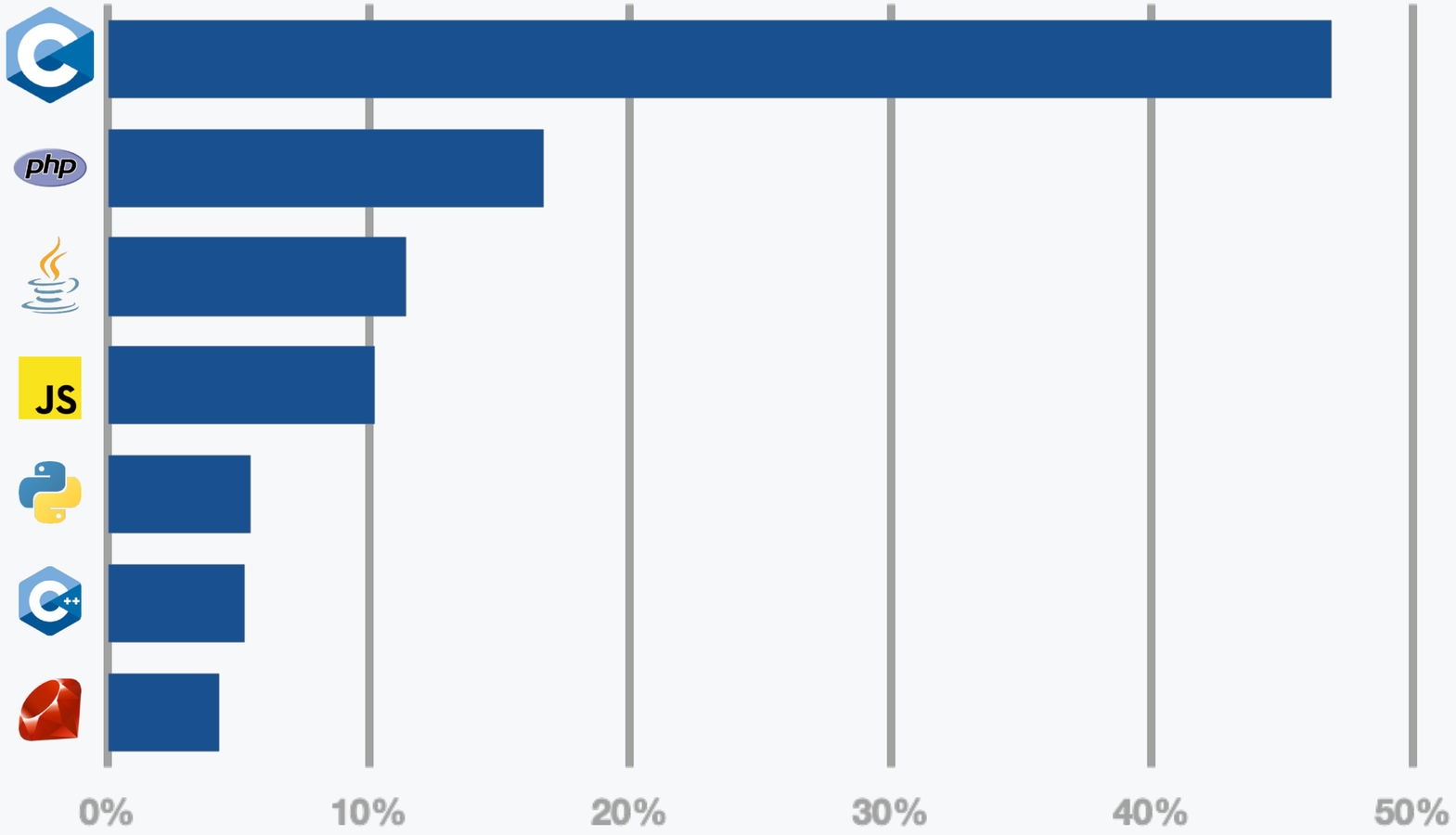
2024년 한국정보과학회 프로그래밍언어연구회 여름학교 번개발표

태그를 붙이냐 마느냐, 그것이 문제로다.
C의 유니언을 러스트의 태그 붙은 유니언으로 (ASE '24)

홍재민, 류석영 | {jaemin.hong, sryu.cs}@kaist.ac.kr

PLRG

C의 낮은 안전성



오픈소스 보안 취약점의 언어별 비율

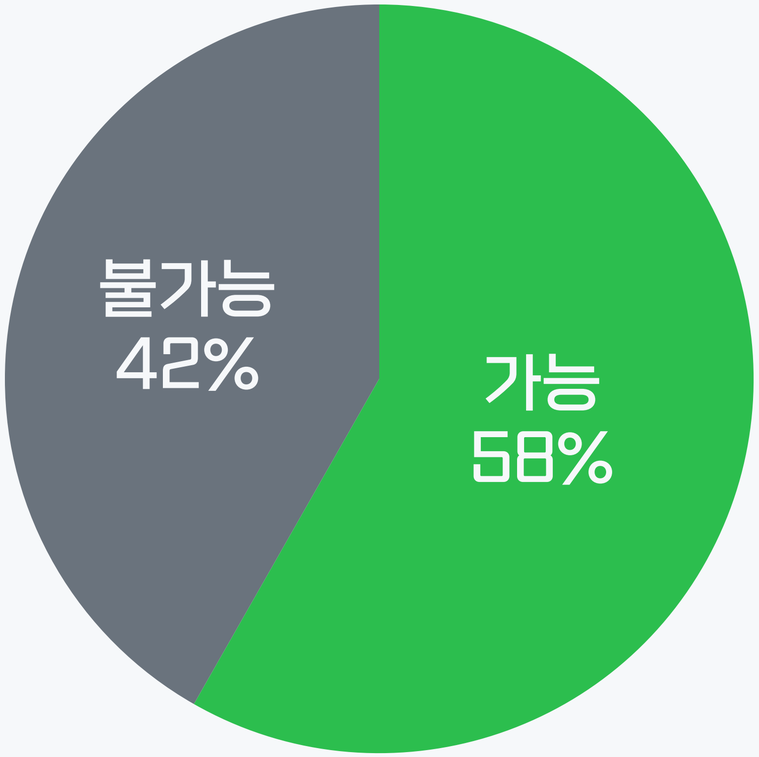
WhiteSource Report: What are the Most Secure Programming Languages

러스트의 높은 안전성

```
error[E0499]: cannot borrow `s` as mutable more than once at a time
 --> b.rs:5:14
  |
4 |     let r1 = &mut s;
  |              ------ first mutable borrow occurs here
5 |     let r2 = &mut s;
  |              ^^^^^^ second mutable borrow occurs here
6 |
7 |     println!("{}, {}", r1, r2);
  |                        -- first borrow later used here

error: aborting due to previous error

For more information about this error, try `rustc --explain E0499`.
```



cURL의 버그를 러스트가 방지할 수 있었는가?

Tim Hutt, Would Rust secure cURL?

러스트가 기존 시스템에 도입되는 중

InfoQ Homepage > News > Linux 6.1 Officially Adds Support For Rust In The Kernel

DEVELOPMENT

Linux 6.1 Officially Adds Support for Rust in the Kernel

LIKE DISCUSS

DEC 20, 2022 • 1 MIN READ

by

Sergio De Simone

After over two years in development, support for using Rust for kerne
which became available a couple of weeks ago.

Previous to its official release, Rust support has been available in linu
developers and maintainers trees, for over a year. With the stable rel
accepted for Linux kernel development, along with C.

Initial Rust support is just the absolute minimum to get Rust code bui
possibly means that Rust support is not ready yet for prime-time dev
level are to be expected in coming releases. Still, there has been quit
should become available in the next future. These include a Rust nvm

DARPA DEFENSE ADVANCED RESEARCH PROJECTS AGENCY

EXPLORE BY TAG

ABOUT US / OUR RESEARCH / NEWS / EVENTS / WORK WITH US /

> Defense Advanced Research Projects Agency > Our Research > Translating All C to Rust

Translating All C to Rust (TRACTOR)

Dr. Dan Wallach

After more than two decades of grappling with memory safety issues in C and C++, the software engineering community has reached a consensus. It's not enough to rely on bug-finding tools. The preferred approach is to use "safe" programming languages that can reject unsafe programs at compile time, thereby preventing the emergence of memory safety issues.

The TRACTOR program aims to automate the translation of legacy C code to Rust. The goal is to achieve the same quality and style that a skilled Rust developer would produce, thereby eliminating the entire class of memory safety security vulnerabilities present in C programs. This program may involve novel combinations of software analysis, such as static analysis and dynamic analysis, and machine learning techniques like large language models.

Additional information is available in the TRACTOR Special Notice on SAM.Gov.

https://www.infoq.com/news/2022/12/linux-6-1-rust/

유니언을 태그 붙은 유니언으로

$$e ::= 1 \mid -e \mid e + e \mid e \times e$$

C

```c
struct Expr {
    int kind;
    union {
        struct Expr *e;
        struct BExpr b;
    } v;
};
struct BExpr {
    struct Expr *l;
    struct Expr *r;
};
```

러스트

```rust
enum Expr {
    One,
    Neg(Box<Expr>),
    Add(Box<Expr>, Box<Expr>),
    Mul(Box<Expr>, Box<Expr>),
}
```

유니언을 태그 붙

C

```
struct Expr {
    int kind;
    union {
        struct
        struct
    } v;
};
struct BExpr {
    struct Exp
    struct Exp
};
```

러스트

```
{
ox<Expr>),
ox<Expr>, Box<Expr>),
ox<Expr>, Box<Expr>),
```

태그를 붙이냐 마느냐, 그것이 문제로다.

C의 유니언을 러스트의 태그 붙은 유니언으로 (ASE '24)

홍재민, 류석영 | {jaemin.hong, sryu.cs}@kaist.ac.kr

1. 동기 및 목표

C의 유니언은 서로 다른 타입의 필드를 같은 공간에 저장.
어떤 필드에 마지막으로 값을 썼는지 언어 수준에서 기록하지 않음.
잘못된 필드에서 값을 읽지 않기 위해 태그 값을 개발자가 직접 기록.
다음 문법의 식을 유니언을 사용해 구현 가능. $e ::= n \mid Succ\ e$

```
struct Expr {
    int kind;
    union { int n; struct Expr *e; } v; };
```

태그 0은 정수, 1은 다음 수를 나타내며, kind는 이를 저장하는 태그 필드.

```
int eval(struct Expr *e) {
 switch (e->kind) { case 0: return e->v.n;
                    case 1: return eval(e->v.e) + 1; }}
```

잘못된 필드를 읽거나 태그 값을 잘못 설정하는 실수를 막을 수 없음.

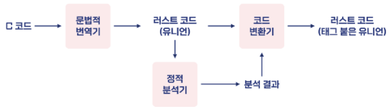
```
case 0: return eval(e->v.e) + 1;
```

```
e->v.n = 0; e->v.kind = 1;
```

러스트는 태그와 유니언을 결합한 태그 붙은 유니언tagged union을 제공.

```
enum Expr { Int(i32), Succ(*mut Expr) }
fn eval(e: *mut Expr) {
    match *e { Int(n) => n, Succ(e) => eval(e) + 1, }}
```

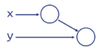
패턴 대조pattern matching를 사용하므로 컴파일러가 개발자의 실수를 방지.
번역 시 유니언과 태그 필드를 태그 붙은 유니언으로 대체해야 함.
분석을 통해 태그 필드와 각 태그 값의 의미를 찾은 뒤 코드를 변환.



2. 정적 분석

조건문 안에서 구조체struct의 필드 값을 알아내야 함.
반드시 가리키는 곳 분석must-points-to analysis 진행.
프로그램의 실행 상태를 그래프로 표현.

```
x = &y;
```

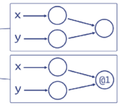


정수 값 정보를 그래프에 함께 표현.

```
x = 1;
```

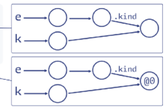
@n은 주소 n에 있는 가상의 값을 의미함.

```
x = y;
match x {
    1 =>
```



같은 값을 가진 메모리 지점으로 새로운 정보를 효율적으로 전파 가능.
여러 메모리 지점의 값을 한번에 바꾸려면 "반드시 같다"라는 정보 필요.

```
let k = (*e).kind;
match k {
    0 => (*e).v.n
```



유니언 필드 n에 접근할 때
구조체 필드 kind의 값이 0이다.
구조체의 필드가 다음 조건을 만족하면 태그 필드로 간주.
- 해당 값을 가진 채 접근하는 유니언 필드가 최대 하나.

예시 1. n에 접근할 때 0, e에 접근할 때 1이면, 태그 필드.
예시 2. n에 접근할 때 0, e에 접근할 때 0 또는 1이면, 태그 필드 아님.
예시 3. n에 접근할 때 0, e에 접근할 때 1 또는 2이면, 태그 필드.

3. 코드 변환

태그 필드를 없애고 유니언을 태그 붙은 유니언으로 대체.

```
struct Expr { kind: i32, v: V }
union V { n: i32, e: *mut Expr }
```

```
struct Expr { v: V }
enum V { n(i32), e(*mut Expr) }
```

단순한 변환naive transformation과 자연스러운 변환idiomatic transformation을 설계.
단순한 변환은 모든 코드에 적용 가능하나 자연스럽지 않은 코드를 생성.
태그 필드 및 유니언 필드의 역할을 대체하는 메서드를 정의 후 사용.

```
match (*e).kind { 1 => eval((*e).v.e),
```

```
match (*e).kind() { 1 => eval((*e).v.get_e()),
```

```
(*e).kind = 0; (*e).v.n = 1;
```

```
(*e).set_kind(0); *(*e).v.deref_n_mut() = 1;
```

자연스러운 변환은 자연스러운 코드를 생성하나 적용 불가능할 수 있음.

```
match (*e).kind { 1 => eval((*e).v.e),
```

```
match (*e).v { V::e(ref x) => eval(*x),
```

```
(*e).kind = 0; (*e).v.n = 1;
```

```
(*e).v = V::n(1);
```

4. 평가

35개의 C로 작성된 프로그램을 사용하여 실험 진행.
RQ1: 분석 정확도
- 74개의 태그 필드를 발견, 0개의 거짓 양성, 5개의 거짓 음성 확인.

RQ2: 올바름
- 테스트 케이스가 있는 23개 중 17개가 변환 후에도 테스트 통과.

RQ3: 효율성
- 19만 줄의 러스트 코드를 4,910초 안에 분석 완료.

RQ4: 성능에의 영향
- 테스트 실행에 0.1초 이상 걸리는 20개가 변환 후에 0.1% 성능 저하.

KAIST

PLRG