Safe Extensions (II)

Dawn Song

dawnsong@cs.berkeley.edu

Enforcing Isolation Using Type Safety

- **XFI's protection is still not fine-grained**
- **Safe languages provide type safety, but cannot handle legacy code**
- **Retrofit legacy code for type safety**
  - **E.g., CCured, Cyclone**
    - » **Issues: fat pointer, change data layout**

Enforcing Safety

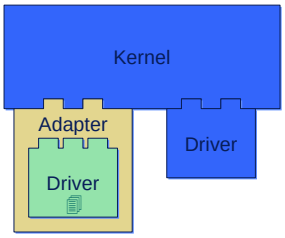
Previous Approach (Cyclone, CCured, SafeC)

```
struct buffer {
    int *data;
    int len;
} b;
```

```
int *data_b;  // lower bound (base)
int *data_e;  // upper bound (end)
```

```
for (i = 0; i < b.len; i++) {
    // verify that b.data[i] is safe
    assert(data_b <= b.data + i < data_e);
}
```

Isolating Extensions with CCured



Problems:

- ✓ **Driver bug can't corrupt kernel**
- ✓ **Driver can't corrupt itself**
- • **Adapter is complicated!**

CCured [PLDI 03], Cyclone [Jim et al., USENIX 02]

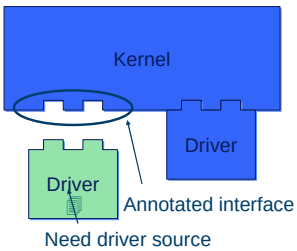
Enforcing Safety with Deputy

```
struct buffer {
    int * count(len) data;
    int len;
} b;

for (i = 0; i < b.len; i++) {
    assert(0 <= i < b.len);
    ... b.data[i] ...
}
```

Advantages:
1. No change in data layout
2. Easier to optimize
3. Contract is in the code!

Isolating Extensions with Deputy



Problems:

- ✓ **Driver bug can't corrupt kernel**
- ✓ **Driver can't corrupt itself**
- ✓ **No adapter required**

Deputy [ESOP 07, OSDI 06]

Deputy

```
struct buffer {
    int * count(len) data;
    int len;
} b;
```

Key Insight:
Most pointers' bounds information is already present in the program in some form--just not in a form the compiler understands!

Deputy

```
struct buffer {
    int * count(len) data;
    int len;
} b;
```

Dependent Types:
Types whose meaning depends on the *run-time value* of a program expression.

Dependent types enable modular checking!

Modularity

Alternative to whole-program analysis and instrumentation

- **Source code unavailable**
- **Source code cannot be recompiled**

Incremental improvements

- **Improve program module by module**
- **Improve overall code quality gradually**

Why Dependent Types?

Used by many common idioms in C code

```
struct buffer {
    char * data;
    int len;
};
```

```
int strlcpy(char * dst,
            char * src,
            int n);
```

```
struct message {
    int tag;
    union {
        int num;
        char *str;
    } u;
};
```


Why Dependent Types?

Used by many common idioms in C code

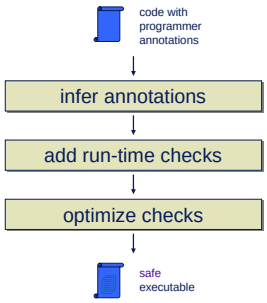
If we annotate these idioms, we can check for correct use!

```
struct buffer {
    char * count(len) data;
    int len;
};
```

```
int strlcpy(char * nt count(n) dst,
            char * nt count(0) src,
            int n);
```

```
struct message {
    int tag;
    union {
        int num   when(tag == 1);
        char *str when(tag == 2);
    } u;
};
```


Compiler Overview



Discussion (I)

- **Do annotations need to be trusted?**
  - **What happens when annotations are conservative?**
    - **E.g., a COUNT(3) pointer actually points to a buffer of length 6?**
- **How well does the "Deputy assumption" hold?**
  - **"Pretty good for array bounds. Breaks down a bit for more complicated cases such as OO-style inheritance..." [Condit]**

Discussion (II)

- **How can attackers circumvent SafeDrive?**
  - **SafeDrive/Deputy assumptions:**
    - **Trusted casts are safe**
    - **Deallocation is safe**
    - **Concurrency is correct (TOCTTOU)**
- **What do you think about "incremental improvement" property from security point of view?**

Discussion (III)

- **So far, most work has focused on isolation. Is this the whole picture? What's missing?**
- **What properties should interface design consider?**
- **What security measures do you need to take for shared data structure?**

Discussion (IV)

- **How may attacker get around XFI?**
- **What would you do to solve the safe extension problem if you were Bill Gates?**
  - **SDV (Static Driver Verifier) shipped with WDK**

Summary

- **Safe extension**
  - **Challenging & important problem**
- **Next class: Virtual Machines & Security**
- **Mid-semester questionnaire**