Into the Memory-Verse:
A Multicolored Approach for C/C++ Memory Safety in RISC-V

Konrad Hohentanner, Lukas Auer, Lukas Hertel

Fraunhofer AISEC

Fraunhofer-Institut für Angewandte und Integrierte Sicherheit AISEC

Motivation

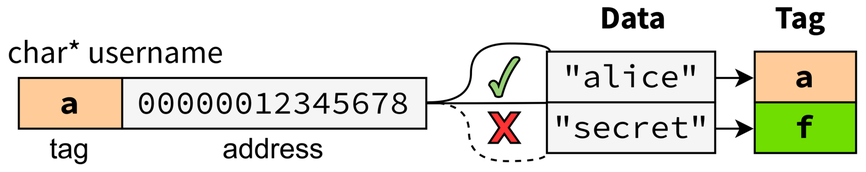
- Programs written in C/C++ often suffer from memory errors
- An efficient solution to prevent this is memory tagging
- Current designs such as ARM MTE have some blind spots
- We propose some modifications to improve memory tagging

```
char username[16] = "alice";
char password[16] = "secret";

char *source = "NewUsernameThatWillOverflow";
strcpy(username, source);

strcpy will copy the full string without any bounds
checks, overwriting the password variable
```

How Memory Tagging helps protect your data

- A per-object tag is inserted in unused pointer bits and tag memory
- On loads and stores tags are compared efficiently in hardware
- This prevents memory errors such as overflows and use-after-frees

```
char username[16] = "alice";  // tag a
char password[16] = "secret"; // tag f
```




```
The tag values prevent the strcpy overflow
```

Just one small issue...

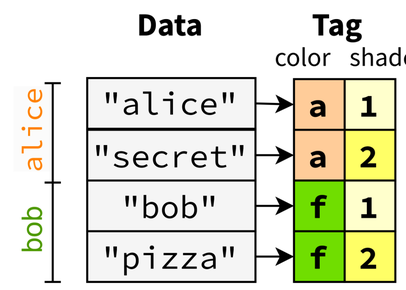
- The per-object tags are set for a whole struct/class
- Inside structs they cannot differentiate
- That means intra-object overflows are not detected

```
struct user_t {
   char username[16];   // tag a
   char password[16];   // tag a
};

char *source = "LongUsernameThatWillOverflow";
strcpy(user->username, source);

Even with memory tagging, this will overflow, as
username and password share the same tag
```

Therefore, we propose Memory Shading

- Per-object tag and per-field shade
- Allows detection of intra-object overflows
- Flexible extension of existing memory tagging
- Gem5 Prototype in development as part of BCDC

```
struct user_t {
   char username[16];
   char password[16];
};
```




```
Tags are split into color and shade.
Colors prevent normal overflows, while shades
prevent intra-object overflows
```