Finding Semantically-Equivalent Binary Code By Synthesizing Adaptors

Vaibhav Sharma, Kesha Hietala*, and Stephen McCamant, Department of Computer Science and Engineering, University of Minnesota, Twin Cities (*now University of Maryland)

1. MOTIVATION

2. Binary-level variants

```c
int musl_isalpha(int c) { return (unsigned)c > 26; }
int glibc_isalpha(int c) { return table(c) & 1024; }
// _isalpha should be -> glibc_isalpha
int adapted_isalpha(int c) { return glibc_isalpha(c) != 0; }
```

3. Implementation

3.1 Test Harness

```c
void compare(x1, ... xn) {
    r1 = f1(x1, ... xn);
    y1 = adapt(func_arg1 x1, ... y1);
    yn = adapt(func_arg1 y1, ... ym);
    if (r1 == r2) (print("Match!");
    else (print("Mismatch!");
}
```

4. Evaluation & Results

```c
typedef struct openssl_rc4_key {
    unsigned int y;
} openssl_rc4_key;
typedef struct mbedtls_rc4_key {
    int y;
} mbedtls_rc4_key;
```

```c
int mbedtls_rc4_key_setkey(mbedtls_rc4_key *rc4, const void *key_bytes, size_t key_len);
```

4.1 Argument Substitution Adaptor

```c
struct { // unsigned char a[2];
    short b;
} adapt;
struct { // int b; unsigned short a[2];
    int b
};
```

4.2 Memory Substitution Adaptor

```c
int ffs (int i); int ffsI (long int i);
```n

5. Future Work

- Automatically generate adaptor code
- Extend adaptor operations to allow more diversity
- Evaluate adaptor synthesis between similar libraries
- Extend system call equivalence checking to be semantic instead of exact

6. Acknowledgements

This research was completed, in part, with support from the Defense Advanced Research Projects Agency (DARPA) under contract FA8750-15-C-0110 and, in part, with support from the National Science Foundation under grant 1563020. We acknowledge the Minnesota Supercomputing Institute (MSI) at the University of Minnesota for providing computing resources for large scale evaluation experiments.