Running the program

```
---- SQL injection demo ----
1:$ # Build
1:$ ./build.sh
```

Running the program

```
---- SQL injection demo ----
1:$ # Build
1:$ ./build.sh

---- SQL injection demo ----
0:$ # Prepare db
0:$ ./admin rm-db

---- SQL injection demo ----
0:$ ./admin create-db

---- SQL injection demo ----
0:$ ./admin show-db
```

Running the program

```
---- SQL injection demo ----
1:$ # Build
1:$ ./build.sh

---- SQL injection demo ----
0:$ # Prepare db
0:$ ./admin rm-db

---- SQL injection demo ----
0:$ ./admin create-db

---- SQL injection demo ----
0:$ ./admin show-db

---- SQL injection demo ----
0:$ # Add regular user interactively
0:$ ./add-user 2>> users.log
*** Welcome to sql injection ***
Please enter name: First User
```

Running the program

```
---- SQL injection demo ----
1:$ # Build
1:$ ./build.sh

---- SQL injection demo ----
0:$ # Prepare db
0:$ ./admin rm-db

---- SQL injection demo ----
0:$ ./admin create-db

---- SQL injection demo ----
0:$ ./admin show-db

---- SQL injection demo ----
0:$ # Add regular user interactively
0:$ ./add-user 2>> users.log
*** Welcome to sql injection ***
Please enter name: First User

---- SQL injection demo ----
0:$ # Check
0:$ ./admin show-db
81750|First User
```

Running the program

```
---- SQL injection demo ----
1:$ # Build
1:$ ./build.sh

---- SQL injection demo ----
0:$ # Prepare db
0:$ ./admin rm-db

---- SQL injection demo ----
0:$ ./admin create-db

---- SQL injection demo ----
0:$ ./admin show-db

---- SQL injection demo ----
0:$ # Add regular user interactively
0:$ ./add-user 2>> users.log
*** Welcome to sql injection ***
Please enter name: First User

---- SQL injection demo ----
0:$ # Check
0:$ ./admin show-db
81750|First User
```

```
---- SQL injection demo ----
0:$ # Regular user via "external" process
0:$ echo "User Outside" | ./add-user 2>> users.log
*** Welcome to sql injection ***
Please enter name:
```

Running the program

```
---- SQL injection demo ----
1:$ # Build
1:$ ./build.sh

---- SQL injection demo ----
0:$ # Prepare db
0:$ ./admin rm-db

---- SQL injection demo ----
0:$ ./admin create-db

---- SQL injection demo ----
0:$ ./admin show-db

---- SQL injection demo ----
0:$ # Add regular user interactively
0:$ ./add-user 2>> users.log
*** Welcome to sql injection ***
Please enter name: First User

---- SQL injection demo ----
0:$ # Check
0:$ ./admin show-db
81750|First User
```

```
---- SQL injection demo ----
0:$ # Regular user via "external" process
0:$ echo "User Outside" | ./add-user 2>> users.log
*** Welcome to sql injection ***
Please enter name:

---- SQL injection demo ----
0:$ ./admin show-db
81750|First User
81757|User Outside
```

Running the program

```
---- SQL injection demo ----
1:$ # Build
1:$ ./build.sh

---- SQL injection demo ----
0:$ # Prepare db
0:$ ./admin rm-db

---- SQL injection demo ----
0:$ ./admin create-db

---- SQL injection demo ----
0:$ ./admin show-db

---- SQL injection demo ----
0:$ # Add regular user interactively
0:$ ./add-user 2>> users.log
*** Welcome to sql injection ***
Please enter name: First User

---- SQL injection demo ----
0:$ # Check
0:$ ./admin show-db
81750|First User
```

```
---- SQL injection demo ----
0:$ # Regular user via "external" process
0:$ echo "User Outside" | ./add-user 2>> users.log
*** Welcome to sql injection ***
Please enter name:

---- SQL injection demo ----
0:$ ./admin show-db
81750|First User
81757|User Outside

---- SQL injection demo ----
0:$ # Add Johnny Droptable
0:$ ./add-user 2>> users.log
*** Welcome to sql injection ***
Please enter name: Johnny'); DROP TABLE users; --
```

Running the program

```
---- SQL injection demo ----
1:$ # Build
1:$ ./build.sh

---- SQL injection demo ----
0:$ # Prepare db
0:$ ./admin rm-db

---- SQL injection demo ----
0:$ ./admin create-db

---- SQL injection demo ----
0:$ ./admin show-db

---- SQL injection demo ----
0:$ # Add regular user interactively
0:$ ./add-user 2>> users.log
*** Welcome to sql injection ***
Please enter name: First User

---- SQL injection demo ----
0:$ # Check
0:$ ./admin show-db
81750|First User
```

```
---- SQL injection demo ----
0:$ # Regular user via "external" process
0:$ echo "User Outside" | ./add-user 2>> users.log
*** Welcome to sql injection ***
Please enter name:

---- SQL injection demo ----
0:$ ./admin show-db
81750|First User
81757|User Outside

---- SQL injection demo ----
0:$ # Add Johnny Droptable
0:$ ./add-user 2>> users.log
*** Welcome to sql injection ***
Please enter name: Johnny'); DROP TABLE users; --

---- SQL injection demo ----
0:$ # And the problem:
0:$ ./admin show-db
Error: near line 2: no such table: users
```

Oops! A command disguised as data

```
---- SQL injection demo ----
0:$ # Add Johnny Droptable
0:$ ./add-user 2>> users.log
*** Welcome to sql injection ***
Please enter name: Johnny'); DROP TABLE users; --

---- SQL injection demo ----
0:$ # And the problem:
0:$ ./admin show-db
Error: near line 2: no such table: users
```

Flow in get_new_id

```
int get_new_id() {
    int id = getpid();
    return id;
}
```

```
int id = getpid();

return id;

int get_new_id() {
```

Flow in `get_new_id`

```
int get_new_id() {
    int id = getpid();
    return id;
}
```

```
int id = getpid();

return id;

int get_new_id() {
```

Flow in `get_new_id`

```
int get_new_id() {
    int id = getpid();
    return id;
}
```

```
int id = getpid();

return id;
```

```
int get_new_id() {
```

Flow in `get_new_id`

```
int get_new_id() {
    int id = getpid();
    return id;
}
```

```
int id = getpid();

return id;

int get_new_id() {
```

Flow in get_user_info

```c
char* get_user_info() {
#define BUFSIZE 1024
    char* buf = (char*) malloc(BUFSIZE * sizeof(char));
    int count;
    // Disable buffering to avoid need for fflush
    // after printf().
    setbuf( stdout, NULL );
    printf("*** Welcome to sql injection ***\n");
    printf("Please enter name: ");
    count = read(STDIN_FILENO, buf, BUFSIZE);
    if (count <= 0) abort();
    /* strip trailing whitespace */
    while (count && isspace(buf[count-1])) {
        buf[count-1] = 0; --count;
    }
    return buf;
}
```

Agent Smith

```c
count = read(STDIN_FILENO, buf, BUFSIZE);

return buf;

char* get_user_info() {
```

Flow in get_user_info

```c
char* get_user_info() {
#define BUFSIZE 1024
    char* buf = (char*) malloc(BUFSIZE * sizeof(char));
    int count;
    // Disable buffering to avoid need for fflush
    // after printf().
    setbuf( stdout, NULL );
    printf("*** Welcome to sql injection ***\n");
    printf("Please enter name: ");
    count = read(STDIN_FILENO, buf, BUFSIZE);
    if (count <= 0) abort();
    /* strip trailing whitespace */
    while (count && isspace(buf[count-1])) {
        buf[count-1] = 0; --count;
    }
    return buf;
}
```

Agent Smith

```c
count = read(STDIN_FILENO, buf, BUFSIZE);

return buf;

char* get_user_info() {
```

Flow in `get_user_info`

```c
char* get_user_info() {
#define BUFSIZE 1024
    char* buf = (char*) malloc(BUFSIZE * sizeof(char));
    int count;
    // Disable buffering to avoid need for fflush
    // after printf().
    setbuf( stdout, NULL );
    printf("*** Welcome to sql injection ***\n");
    printf("Please enter name: ");
    count = read(STDIN_FILENO, buf, BUFSIZE);
    if (count <= 0) abort();
    /* strip trailing whitespace */
    while (count && isspace(buf[count-1])) {
        buf[count-1] = 0; --count;
    }
    return buf;
}
```

Agent Smith

```c
count = read(STDIN_FILENO, buf, BUFSIZE);
```

```c
return buf;
```

```c
char* get_user_info() {
```

Flow in `get_user_info`

```c
char* get_user_info() {
#define BUFSIZE 1024
    char* buf = (char*) malloc(BUFSIZE * sizeof(char));
    int count;
    // Disable buffering to avoid need for fflush
    // after printf().
    setbuf( stdout, NULL );
    printf("*** Welcome to sql injection ***\n");
    printf("Please enter name: ");
    count = read(STDIN_FILENO, buf, BUFSIZE);
    if (count <= 0) abort();
    /* strip trailing whitespace */
    while (count && isspace(buf[count-1])) {
        buf[count-1] = 0; --count;
    }
    return buf;
}
```

Agent Smith

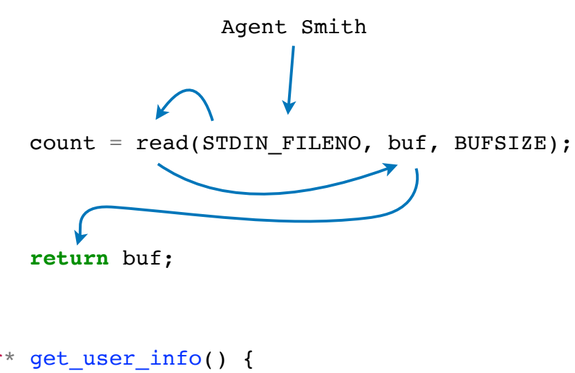
```c
count = read(STDIN_FILENO, buf, BUFSIZE);
```

```c
return buf;
```

```c
char* get_user_info() {
```

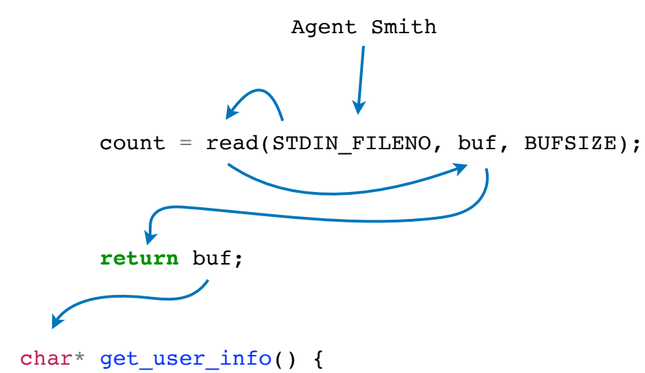
Flow in `get_user_info`

```c
char* get_user_info() {
#define BUFSIZE 1024
    char* buf = (char*) malloc(BUFSIZE * sizeof(char));
    int count;
    // Disable buffering to avoid need for fflush
    // after printf().
    setbuf( stdout, NULL );
    printf("*** Welcome to sql injection ***\n");
    printf("Please enter name: ");
    count = read(STDIN_FILENO, buf, BUFSIZE);
    if (count <= 0) abort();
    /* strip trailing whitespace */
    while (count && isspace(buf[count-1])) {
        buf[count-1] = 0; --count;
    }
    return buf;
}
```



Flow in get_user_info

```c
char* get_user_info() {
#define BUFSIZE 1024
    char* buf = (char*) malloc(BUFSIZE * sizeof(char));
    int count;
    // Disable buffering to avoid need for fflush
    // after printf().
    setbuf( stdout, NULL );
    printf("*** Welcome to sql injection ***\n");
    printf("Please enter name: ");
    count = read(STDIN_FILENO, buf, BUFSIZE);
    if (count <= 0) abort();
    /* strip trailing whitespace */
    while (count && isspace(buf[count-1])) {
        buf[count-1] = 0; --count;
    }
    return buf;
}
```



Flow in `write_info`

```c
void write_info(int id, char* info) {
    sqlite3 *db;
    int rc;
    int bufsize = 1024;
    char *zErrMsg = 0;
    char query[bufsize];

    /* open db */
    rc = sqlite3_open("users.sqlite", &db);
    abort_on_error(rc, db);

    /* Format query */
    snprintf(query, bufsize,
             "INSERT INTO users VALUES (%d, '%s')",
             id, info);
    write_log("query: %s\n", query);

    /* Write info */
    rc = sqlite3_exec(db, query, NULL, 0, &zErrMsg);
    abort_on_exec_error(rc, db, zErrMsg);

    sqlite3_close(db);
}
```

```c
void write_info(int id, char* info)
```

```c
snprintf(query, bufsize, "INSERT INTO users VALUES (%d, '%s')", id, info);
```

```c
rc = sqlite3_exec(db, query, NULL, 0, &zErrMsg);
```

Flow in `write_info`

```c
void write_info(int id, char* info) {
    sqlite3 *db;
    int rc;
    int bufsize = 1024;
    char *zErrMsg = 0;
    char query[bufsize];

    /* open db */
    rc = sqlite3_open("users.sqlite", &db);
    abort_on_error(rc, db);

    /* Format query */
    snprintf(query, bufsize,
             "INSERT INTO users VALUES (%d, '%s')",
             id, info);
    write_log("query: %s\n", query);

    /* Write info */
    rc = sqlite3_exec(db, query, NULL, 0, &zErrMsg);
    abort_on_exec_error(rc, db, zErrMsg);

    sqlite3_close(db);
}
```

```c
void write_info(int id, char* info)
```

```c
snprintf(query, bufsize, "INSERT INTO users VALUES (%d, '%s')", id, info);
```

```c
rc = sqlite3_exec(db, query, NULL, 0, &zErrMsg);
```

Flow in `write_info`

```c
void write_info(int id, char* info) {
    sqlite3 *db;
    int rc;
    int bufsize = 1024;
    char *zErrMsg = 0;
    char query[bufsize];

    /* open db */
    rc = sqlite3_open("users.sqlite", &db);
    abort_on_error(rc, db);

    /* Format query */
    snprintf(query, bufsize,
             "INSERT INTO users VALUES (%d, '%s')",
             id, info);
    write_log("query: %s\n", query);

    /* Write info */
    rc = sqlite3_exec(db, query, NULL, 0, &zErrMsg);
    abort_on_exec_error(rc, db, zErrMsg);

    sqlite3_close(db);
}
```

```c
void write_info(int id, char* info)
```

```c
snprintf(query, bufsize, "INSERT INTO users VALUES (%d, '%s')", id, info);
```

```c
rc = sqlite3_exec(db, query, NULL, 0, &zErrMsg);
```

Flow in `write_info`

```c
void write_info(int id, char* info) {
    sqlite3 *db;
    int rc;
    int bufsize = 1024;
    char *zErrMsg = 0;
    char query[bufsize];

    /* open db */
    rc = sqlite3_open("users.sqlite", &db);
    abort_on_error(rc, db);

    /* Format query */
    snprintf(query, bufsize,
             "INSERT INTO users VALUES (%d, '%s')",
             id, info);
    write_log("query: %s\n", query);

    /* Write info */
    rc = sqlite3_exec(db, query, NULL, 0, &zErrMsg);
    abort_on_exec_error(rc, db, zErrMsg);

    sqlite3_close(db);
}
```

```c
void write_info(int id, char* info)
```

```c
snprintf(query, bufsize, "INSERT INTO users VALUES (%d, '%s')", id, info);
```

```c
rc = sqlite3_exec(db, query, NULL, 0, &zErrMsg);
```

Flow in `write_info`

```c
void write_info(int id, char* info) {
    sqlite3 *db;
    int rc;
    int bufsize = 1024;
    char *zErrMsg = 0;
    char query[bufsize];

    /* open db */
    rc = sqlite3_open("users.sqlite", &db);
    abort_on_error(rc, db);

    /* Format query */
    snprintf(query, bufsize,
             "INSERT INTO users VALUES (%d, '%s')",
             id, info);
    write_log("query: %s\n", query);

    /* Write info */
    rc = sqlite3_exec(db, query, NULL, 0, &zErrMsg);
    abort_on_exec_error(rc, db, zErrMsg);

    sqlite3_close(db);
}
```

```c
void write_info(int id, char* info)
```

```c
snprintf(query, bufsize, "INSERT INTO users VALUES (%d, '%s')", id, info);
```

```c
rc = sqlite3_exec(db, query, NULL, 0, &zErrMsg);
```

Flow in `write_info`

```c
void write_info(int id, char* info) {
    sqlite3 *db;
    int rc;
    int bufsize = 1024;
    char *zErrMsg = 0;
    char query[bufsize];

    /* open db */
    rc = sqlite3_open("users.sqlite", &db);
    abort_on_error(rc, db);

    /* Format query */
    snprintf(query, bufsize,
             "INSERT INTO users VALUES (%d, '%s')",
             id, info);
    write_log("query: %s\n", query);

    /* Write info */
    rc = sqlite3_exec(db, query, NULL, 0, &zErrMsg);
    abort_on_exec_error(rc, db, zErrMsg);

    sqlite3_close(db);
}
```

```c
void write_info(int id, char* info)
```

```c
snprintf(query, bufsize, "INSERT INTO users VALUES (%d, '%s')", id, info);
```

```c
rc = sqlite3_exec(db, query, NULL, 0, &zErrMsg);
```

Flow in `write_info`

```c
void write_info(int id, char* info) {
    sqlite3 *db;
    int rc;
    int bufsize = 1024;
    char *zErrMsg = 0;
    char query[bufsize];

    /* open db */
    rc = sqlite3_open("users.sqlite", &db);
    abort_on_error(rc, db);

    /* Format query */
    snprintf(query, bufsize,
             "INSERT INTO users VALUES (%d, '%s')",
             id, info);
    write_log("query: %s\n", query);

    /* Write info */
    rc = sqlite3_exec(db, query, NULL, 0, &zErrMsg);
    abort_on_exec_error(rc, db, zErrMsg);

    sqlite3_close(db);
}
```

```c
void write_info(int id, char* info)
```

```c
snprintf(query, bufsize, "INSERT INTO users VALUES (%d, '%s')", id, info);
```

```c
rc = sqlite3_exec(db, query, NULL, 0, &zErrMsg);
```

Flow in `write_info`

```c
void write_info(int id, char* info) {
    sqlite3 *db;
    int rc;
    int bufsize = 1024;
    char *zErrMsg = 0;
    char query[bufsize];

    /* open db */
    rc = sqlite3_open("users.sqlite", &db);
    abort_on_error(rc, db);

    /* Format query */
    snprintf(query, bufsize,
             "INSERT INTO users VALUES (%d, '%s')",
             id, info);
    write_log("query: %s\n", query);

    /* Write info */
    rc = sqlite3_exec(db, query, NULL, 0, &zErrMsg);
    abort_on_exec_error(rc, db, zErrMsg);

    sqlite3_close(db);
}
```

```c
void write_info(int id, char* info)
```

```c
snprintf(query, bufsize, "INSERT INTO users VALUES (%d, '%s')", id, info);
```

```c
rc = sqlite3_exec(db, query, NULL, 0, &zErrMsg);
```

Flow in `write_info`

```c
void write_info(int id, char* info) {
    sqlite3 *db;
    int rc;
    int bufsize = 1024;
    char *zErrMsg = 0;
    char query[bufsize];

    /* open db */
    rc = sqlite3_open("users.sqlite", &db);
    abort_on_error(rc, db);

    /* Format query */
    snprintf(query, bufsize,
             "INSERT INTO users VALUES (%d, '%s')",
             id, info);
    write_log("query: %s\n", query);

    /* Write info */
    rc = sqlite3_exec(db, query, NULL, 0, &zErrMsg);
    abort_on_exec_error(rc, db, zErrMsg);

    sqlite3_close(db);
}
```

```c
void write_info(int id, char* info)
```

```c
snprintf(query, bufsize, "INSERT INTO users VALUES (%d, '%s')", id, info);
```

```c
rc = sqlite3_exec(db, query, NULL, 0, &zErrMsg);
```

Flow in `main`

```
int main(int argc, char* argv[]) {
    char* info;
    int id;
    info = get_user_info();
    id = get_new_id();
    write_info(id, info);
}
```

```
info = get_user_info();
```

```
id = get_new_id();
```

```
write_info(id, info);
```

Flow in `main`

```
int main(int argc, char* argv[]) {
    char* info;
    int id;
    info = get_user_info();
    id = get_new_id();
    write_info(id, info);
}
```

```
info = get_user_info();
```

```
id = get_new_id();
```

```
write_info(id, info);
```

Flow in `main`

```
int main(int argc, char* argv[]) {

    char* info;

    int id;

    info = get_user_info();

    id = get_new_id();

    write_info(id, info);

}
```

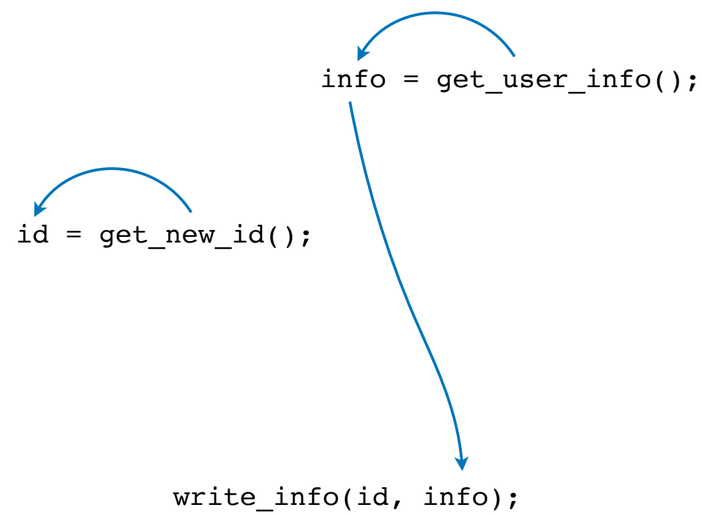
```
info = get_user_info();
```

```
id = get_new_id();
```

```
write_info(id, info);
```

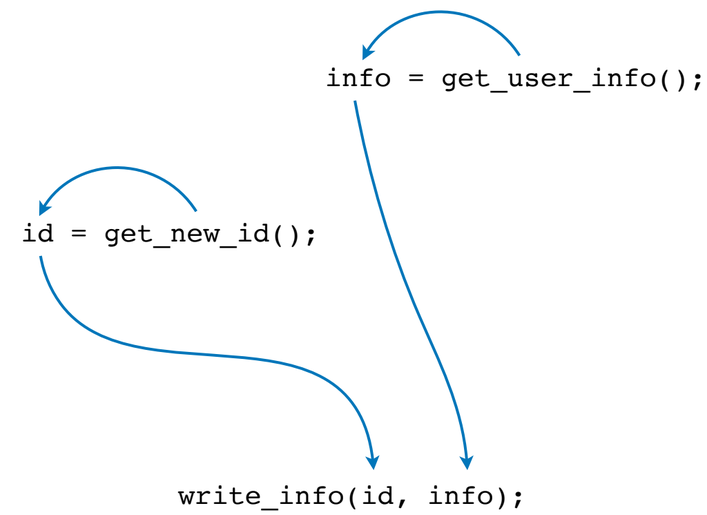
Flow in `main`

```
int main(int argc, char* argv[]) {

    char* info;

    int id;

    info = get_user_info();

    id = get_new_id();

    write_info(id, info);

}
```



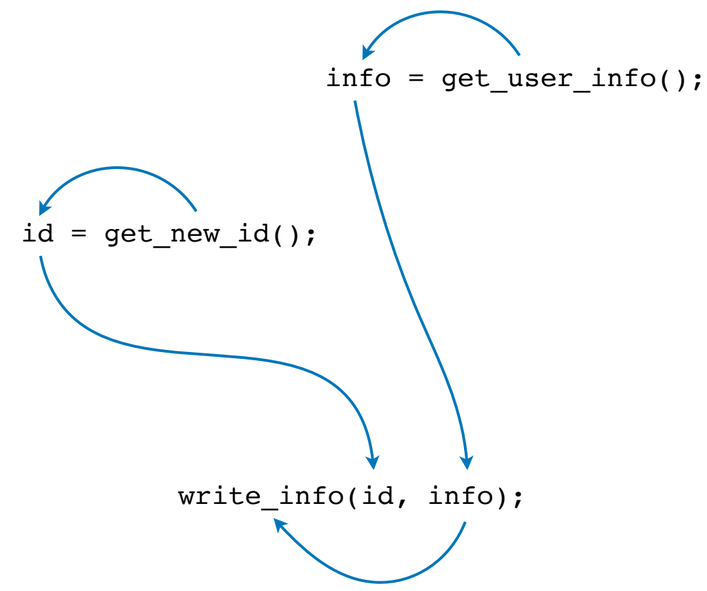
Flow in `main`

```
int main(int argc, char* argv[]) {

    char* info;

    int id;

    info = get_user_info();

    id = get_new_id();

    write_info(id, info);

}
```



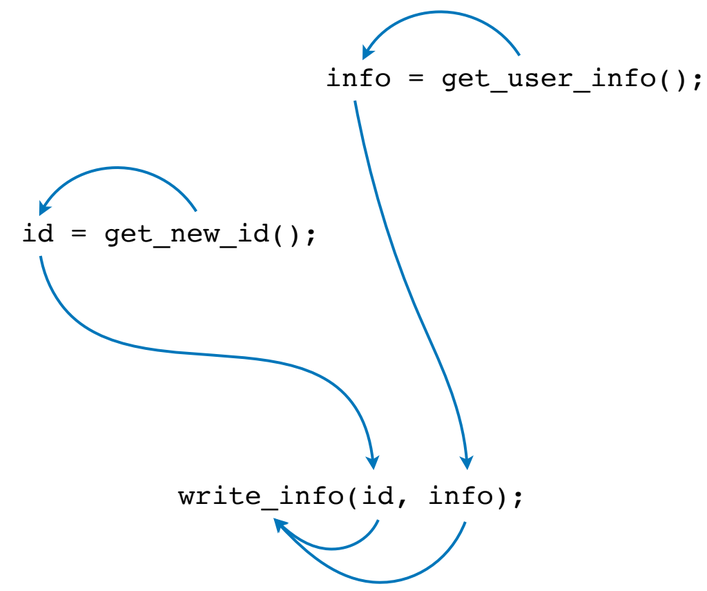
Flow in `main`

```
int main(int argc, char* argv[]) {

    char* info;

    int id;

    info = get_user_info();

    id = get_new_id();

    write_info(id, info);

}
```



Flow in `main`

```
int main(int argc, char* argv[]) {

    char* info;

    int id;

    info = get_user_info();

    id = get_new_id();

    write_info(id, info);

}
```



Flow combined

```
int id = getpid();

return id;

int get_new_id() {
```

```
Agent Smith

count = read(STDIN_FILENO, buf, BUFSIZE);

return buf;

char* get_user_info() {
```

```
id = get_new_id();

info = get_user_info();

write_info(id, info);
```

```
void write_info(int id, char* info)

snprintf(query, bufsize, "INSERT INTO users VALUES (%d, '%s')", id, info);

rc = sqlite3_exec(db, query, NULL, 0, &zErrMsg);
```

- sink on bottom: second argument to `sqlite3_exec`
- propagation through `snprintf` needs taint flow
- this is roughly the flow we expect to see; may have to help CodeQL to capture flow across some functions

Flow combined

- inter-procedural (global) data flow

```
int id = getpid();

return id;

int get_new_id() {
```

```
Agent Smith

count = read(STDIN_FILENO, buf, BUFSIZE);

return buf;

char* get_user_info() {
```

```
id = get_new_id();

info = get_user_info();

write_info(id, info);
```

```
void write_info(int id, char* info)

snprintf(query, bufsize, "INSERT INTO users VALUES (%d, '%s')", id, info);

rc = sqlite3_exec(db, query, NULL, 0, &zErrMsg);
```

- sink on bottom: second argument to `sqlite3_exec`
- propagation through `snprintf` needs taint flow
- this is roughly the flow we expect to see; may have to help CodeQL to capture flow across some functions

Flow combined

- inter-procedural (global) data flow

```
int id = getpid();

return id;

int get_new_id() {
```

```
Agent Smith

count = read(STDIN_FILENO, buf, BUFSIZE);

return buf;

char* get_user_info() {
```

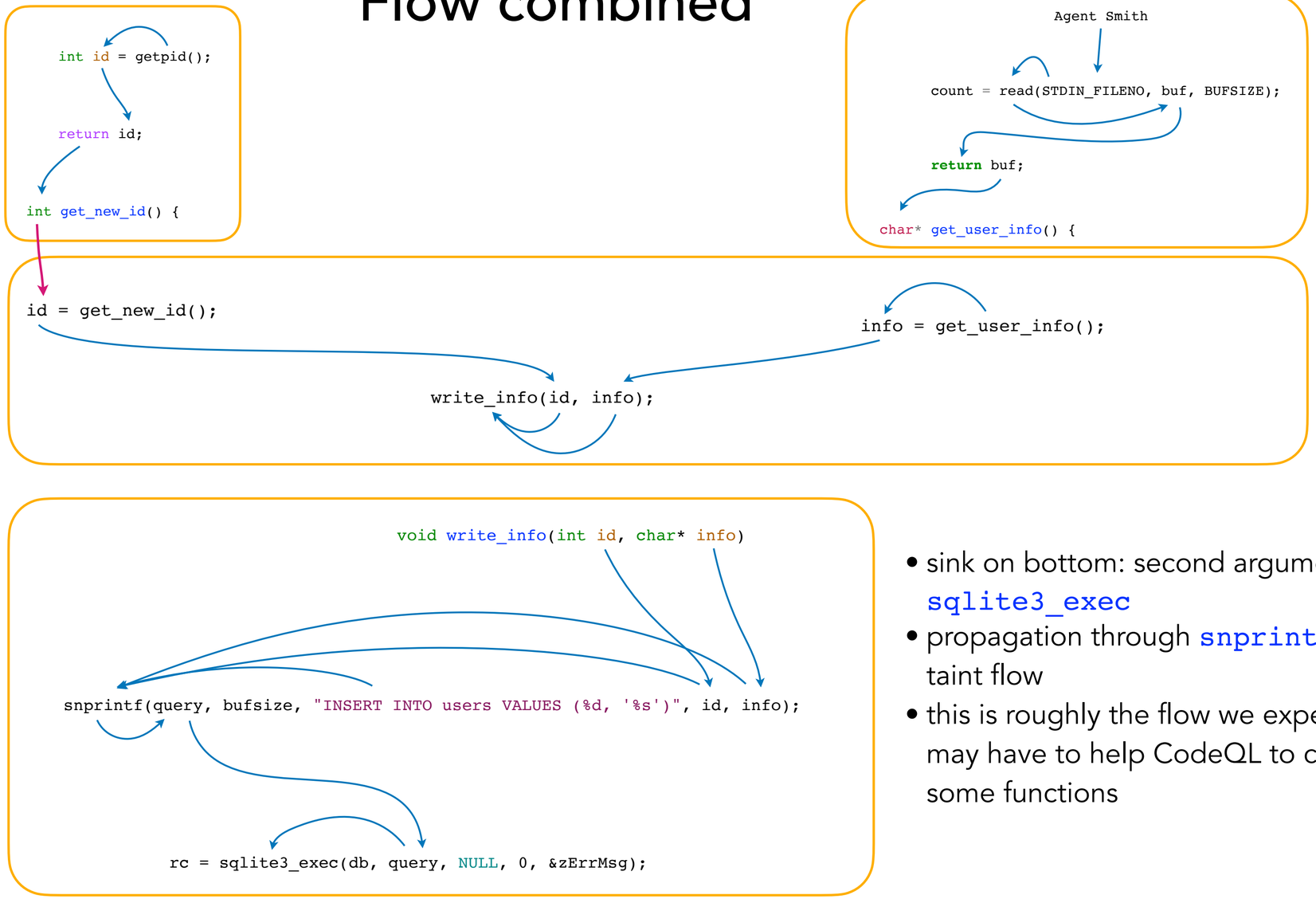
```
id = get_new_id();

info = get_user_info();

write_info(id, info);
```

```
void write_info(int id, char* info)

snprintf(query, bufsize, "INSERT INTO users VALUES (%d, '%s')", id, info);

rc = sqlite3_exec(db, query, NULL, 0, &zErrMsg);
```

- sink on bottom: second argument to `sqlite3_exec`
- propagation through `snprintf` needs taint flow
- this is roughly the flow we expect to see; may have to help CodeQL to capture flow across some functions

Flow combined

- inter-procedural (global) data flow



- sink on bottom: second argument to `sqlite3_exec`
- propagation through `snprintf` needs taint flow
- this is roughly the flow we expect to see; may have to help CodeQL to capture flow across some functions

Flow combined

- inter-procedural (global) data flow

```
int id = getpid();

return id;

int get_new_id() {
```

```
Agent Smith

count = read(STDIN_FILENO, buf, BUFSIZE);

return buf;

char* get_user_info() {
```

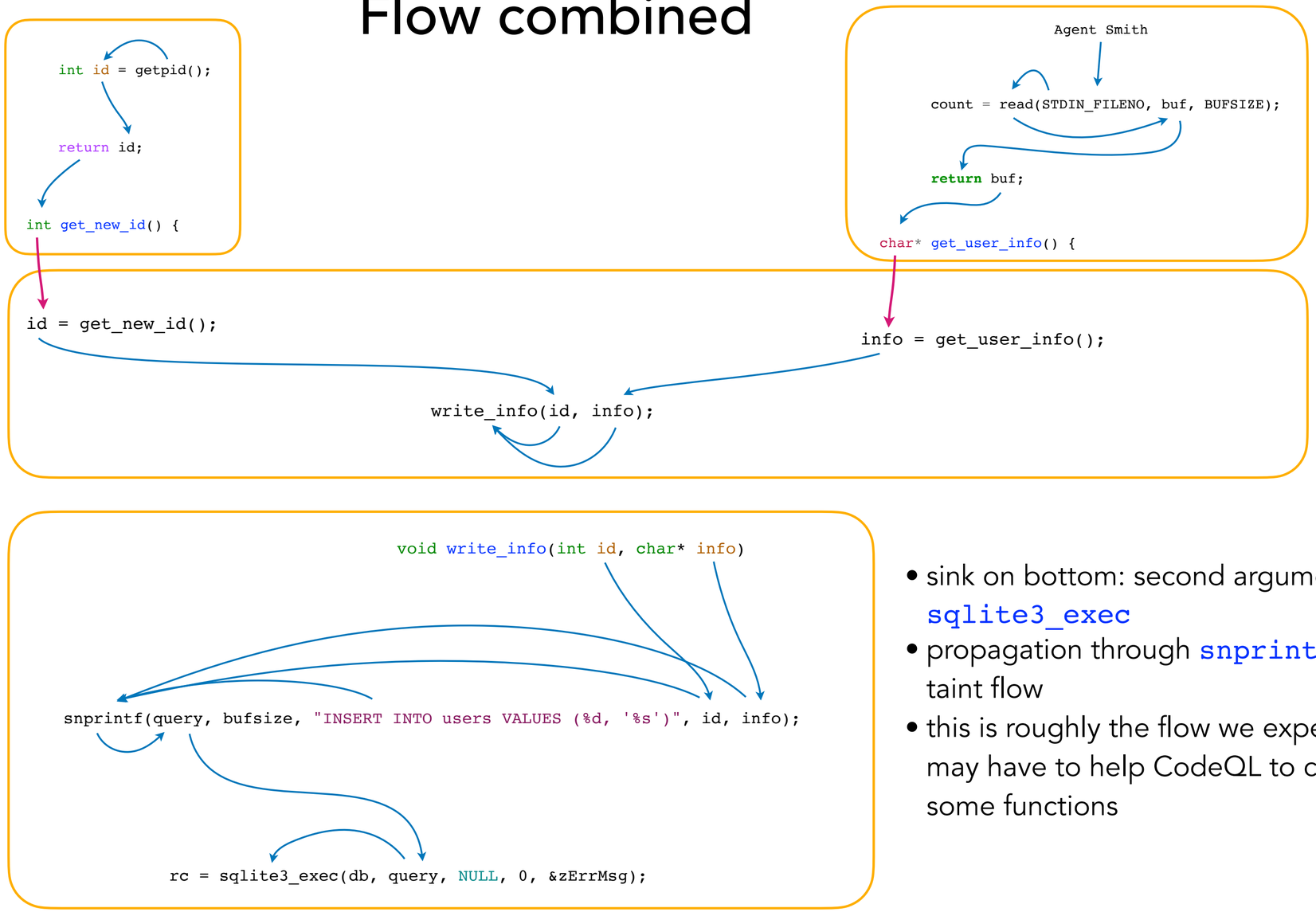
```
id = get_new_id();

info = get_user_info();

write_info(id, info);
```

```
void write_info(int id, char* info)

snprintf(query, bufsize, "INSERT INTO users VALUES (%d, '%s')", id, info);

rc = sqlite3_exec(db, query, NULL, 0, &zErrMsg);
```

- sink on bottom: second argument to `sqlite3_exec`
- propagation through `snprintf` needs taint flow
- this is roughly the flow we expect to see; may have to help CodeQL to capture flow across some functions

Flow combined

- inter-procedural (global) data flow



- sink on bottom: second argument to `sqlite3_exec`
- propagation through `snprintf` needs taint flow
- this is roughly the flow we expect to see; may have to help CodeQL to capture flow across some functions

Flow combined

- inter-procedural (global) data flow

```
int id = getpid();

return id;

int get_new_id() {
```

```
Agent Smith

count = read(STDIN_FILENO, buf, BUFSIZE);

return buf;

char* get_user_info() {
```

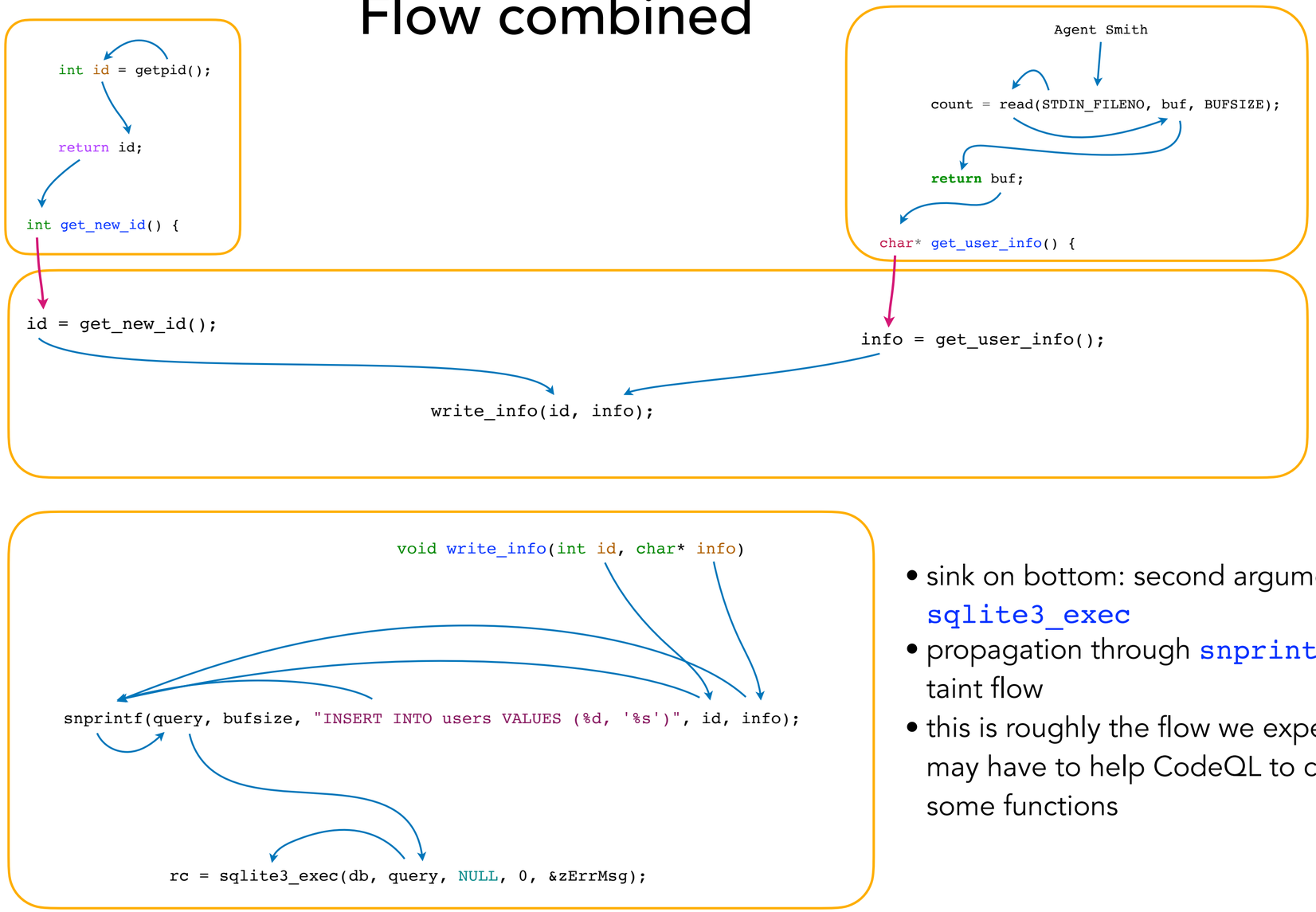
```
id = get_new_id();

info = get_user_info();

write_info(id, info);
```

```
void write_info(int id, char* info)

snprintf(query, bufsize, "INSERT INTO users VALUES (%d, '%s')", id, info);

rc = sqlite3_exec(db, query, NULL, 0, &zErrMsg);
```

- sink on bottom: second argument to `sqlite3_exec`
- propagation through `snprintf` needs taint flow
- this is roughly the flow we expect to see; may have to help CodeQL to capture flow across some functions

Flow combined

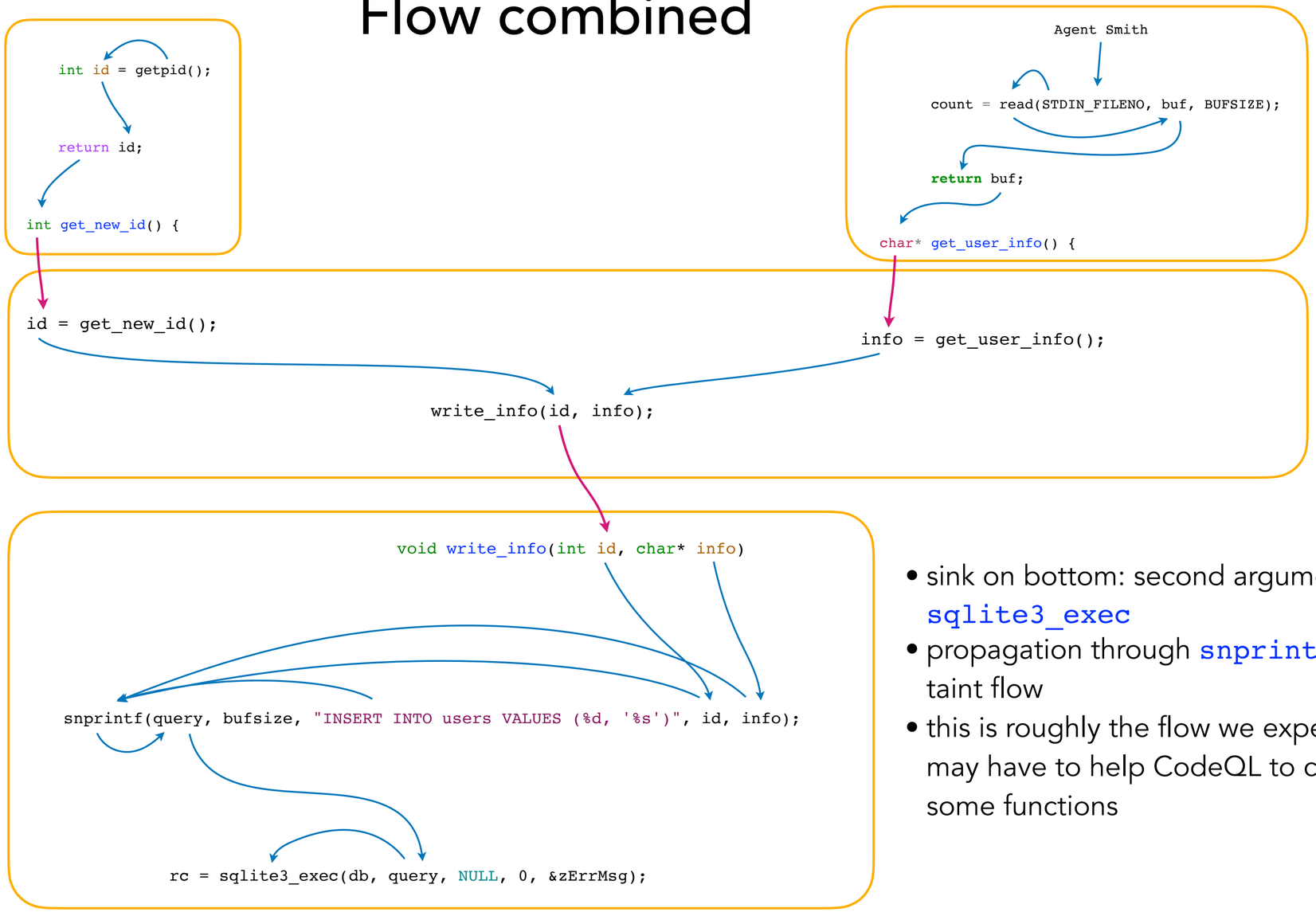
- inter-procedural (global) data flow



- sink on bottom: second argument to `sqlite3_exec`
- propagation through `snprintf` needs taint flow
- this is roughly the flow we expect to see; may have to help CodeQL to capture flow across some functions

Flow combined

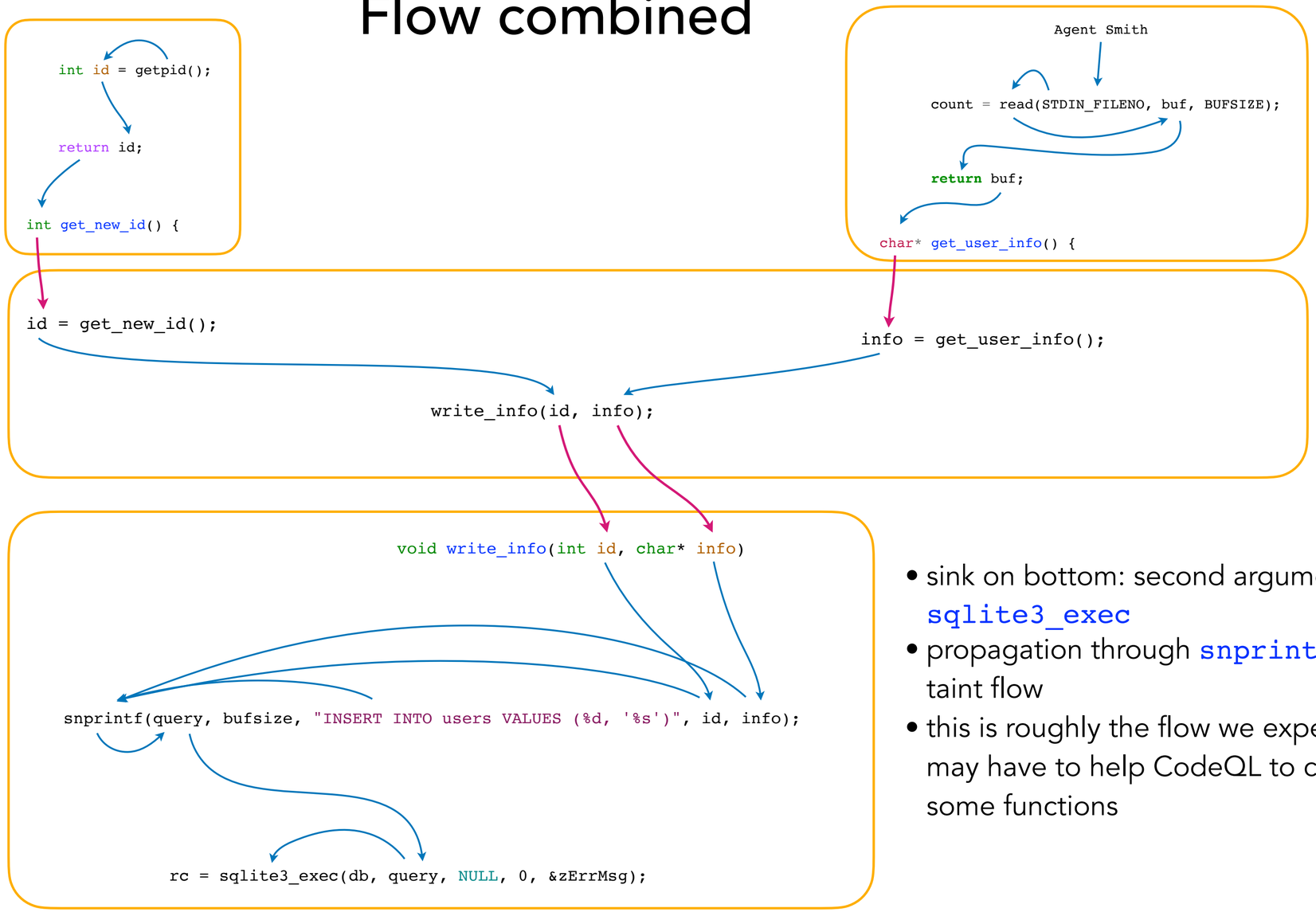
- inter-procedural (global) data flow



- sink on bottom: second argument to `sqlite3_exec`
- propagation through `snprintf` needs taint flow
- this is roughly the flow we expect to see; may have to help CodeQL to capture flow across some functions

Flow combined

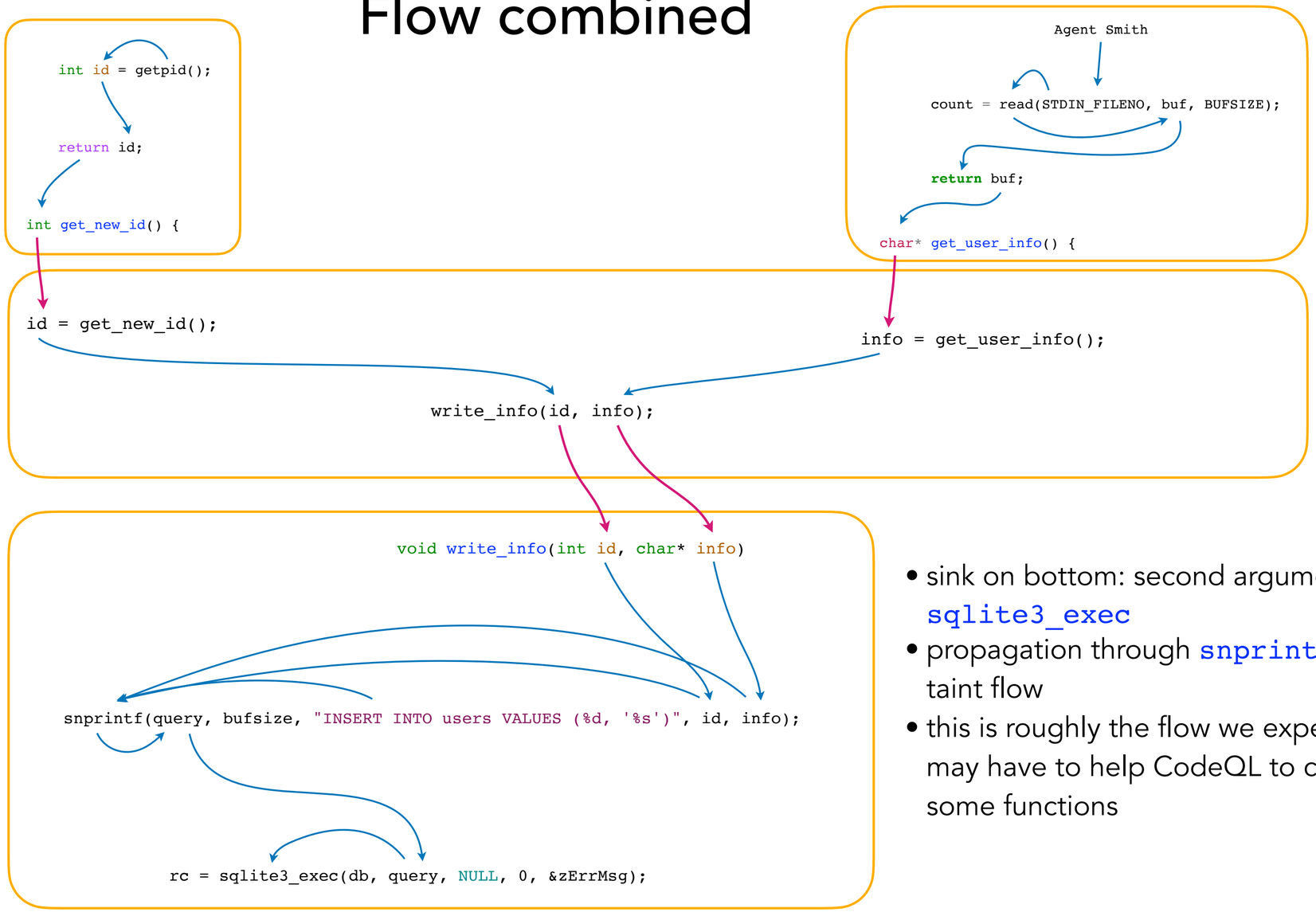
- inter-procedural (global) data flow



- sink on bottom: second argument to `sqlite3_exec`
- propagation through `snprintf` needs taint flow
- this is roughly the flow we expect to see; may have to help CodeQL to capture flow across some functions

Flow combined

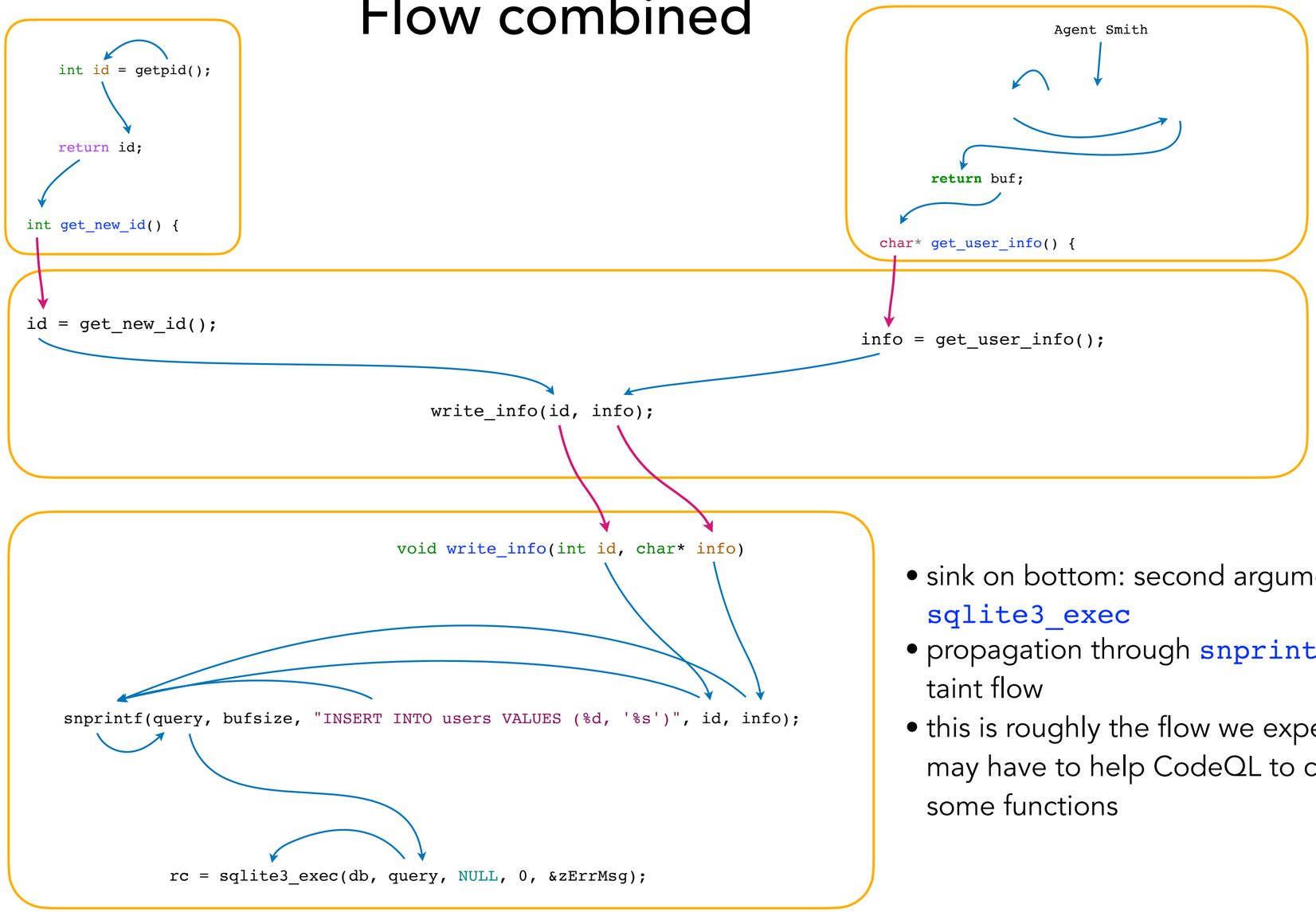
- inter-procedural (global) data flow
- source on top: second argument to `read`



- sink on bottom: second argument to `sqlite3_exec`
- propagation through `snprintf` needs taint flow
- this is roughly the flow we expect to see; may have to help CodeQL to capture flow across some functions

Flow combined

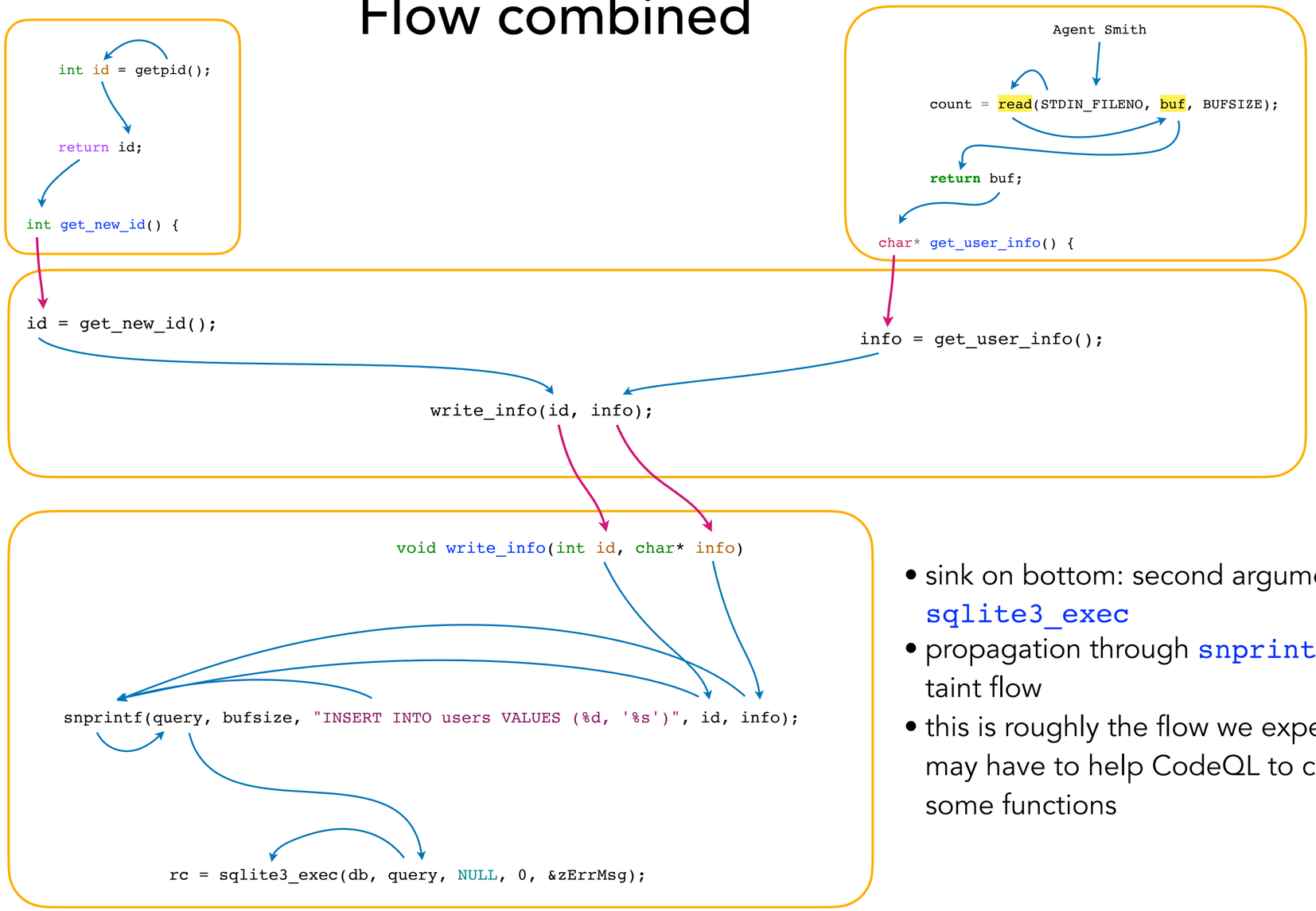
- inter-procedural (global) data flow
- source on top: second argument to `read`



- sink on bottom: second argument to `sqlite3_exec`
- propagation through `snprintf` needs taint flow
- this is roughly the flow we expect to see; may have to help CodeQL to capture flow across some functions

Flow combined

- inter-procedural (global) data flow
- source on top: second argument to `read`



- sink on bottom: second argument to `sqlite3_exec`
- propagation through `snprintf` needs taint flow
- this is roughly the flow we expect to see; may have to help CodeQL to capture flow across some functions