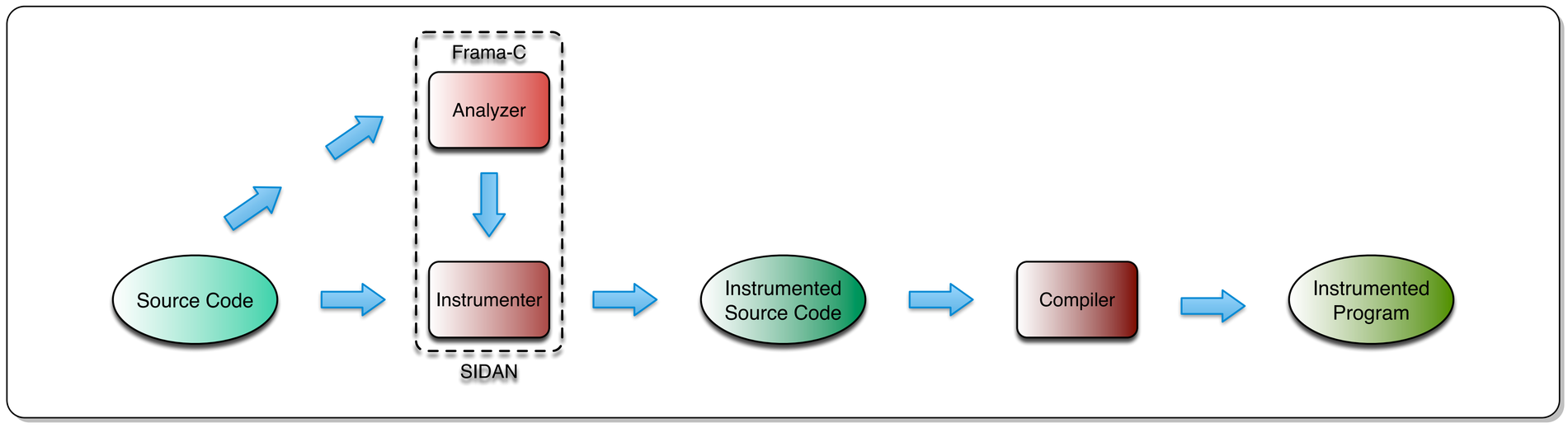
Software Instrumentation for the Detection of Attacks on Non-control-data

Jonathan-Christofer Demay, Éric Totel and Frédéric Tronel
SUPELEC, Rennes, France, first_name.last_name@supelec.fr
http://www.rennes.supelec.fr/ren/rd/ssir/outils/sidan/

An Integrated Software Instrumentation Process

Our work focuses on the detection of non-control-data attacks by checking for variable corruptions that may lead to an illegal system call executed by a valid code (i.e. not injected). Since those attacks need to corrupt specific data with specific values, they may put the memory of the process in an inconsistent state. Our approach consists in discovering consistency properties in a program through static analysis in order to detect data corruptions induced by non-control-data attacks. We thus build a data-consistency-oriented model based on these properties and use it to instrument the program to detect such attacks at runtime. We have developed a prototype based on *Frama-C* that computes this normal behavior model and inserts executable assertions derived from this model. The resulting tool is a source to source translator that integrates itself in the building process and transforms untrusted programs into data security checking programs.



Intrusion Sensitive Data Set

To detect data inconsistencies induced by a non-control-data attack, our approach have to consider the set of variables a system call depends on. We did choose to use the *Frama-C*'s plugin *Program Dependence Graph* to determine such a set.

Data Constraints Discovery

To detect inconsistencies on a set of data, our approach needs to discover constraints on the paths that lead to the corresponding system call where the data are accessible. We use the *Value Analysis* feature of *Frama-C* to compute them.

Source Sample

```
packet_start(SSH_SMSG_AUTH_NEEDED);
packet_send();
int auth_ok = 0;
if(passwd != NULL)
    while(auth_ok != 1){
        type = packet_read(data);
        switch (type){
            ...
            case SSH_CMSG_AUTH:
                auth_ok = auth(passwd, data);
                break;
            default:
                log(UNKNOWN_MESSAGE, type);
                break;
    }}
log(AUTH_SUCCESSFUL, user);
authenticated(uid);
```

Program Slice

```
int auth_ok = 0;
if(passwd != NULL)
    while(auth_ok != 1){
        type = packet_read(data);
        switch (type){
            ...
            case SSH_CMSG_AUTH:
                auth_ok = auth(passwd, data);
                break;

    }}

authenticated(uid); //point of interest
```

Variation Domain

auth_ok	passwd	type	data	uid
{0}	#UNDEF	#UNDEF	#UNDEF	#UNDEF
{0}	{!NULL}	#UNDEF	#UNDEF	#UNDEF
{0}	{!NULL}	#UNDEF	#UNDEF	#UNDEF
{0}	{!NULL}	#UNDEF	#UNDEF	#UNDEF
...				
{0}	{!NULL}	{SSH_CMSG_AUTH}	#UNDEF	#UNDEF
{0,1}	{!NULL}	{SSH_CMSG_AUTH}	#UNDEF	#UNDEF

(auth_ok,type) in {({0},#UNDEF),({1},{SSH_CMSG_AUTH})}

SIDAN Results

We have implemented our approach as a *Frama-C*'s plugin and have tested it on a vulnerable version of *OpenSSH*. The instrumentation process covers 8% of the function calls. The program now runs with a runtime overhead of about 0.5% and the two known non-control-data attacks exploiting this vulnerability are detected.

Attack

```
int auth_ok = 0;
if(passwd != NULL)
        while(auth_ok != 1){
                /* an integer overflow in packet_read impacts a
                   boundary check thus allowing malicious users
                   to remotely overwrite at arbitrary locations
                   data such as auth_ok or passwd to bypass the
                   authentication mechanism */
                type = packet_read(data);
                switch (type){
                        ...
                        case SSH_CMSG_AUTH:
                                auth_ok = auth_passwd(passwd, data);
                                break;
                }
        }
authenticated(uid);
```

Detection

```
int auth_ok = 0;
if(passwd != NULL)
        while(auth_ok != 1){
                type = packet_read(data);
                switch (type){
                        ...
                        case SSH_CMSG_AUTH:
                                /* check passwd according to
                                   the point of execution     */
                                assert(passwd != NULL);
                                auth_ok = auth_passwd(passwd, data);
                                break;
                }
        }
/* check auth_ok according the type of the last message */
assert((auth_ok == 1 && type = SSH_CMSG_AUTH) || auth_ok == 0);
authenticated(uid);
```