SENG: An Enhanced Policy Language for SELinux

Paul Kuliniewicz <kuliniew@purdue.edu>

What is SELinux?

SELinux adds support for **mandatory access controls** to the Linux kernel. In SELinux, every object is assigned a **type**, and the system-wide policy specifies which permissions each type has over the others.

Macros: The Good, The Bad, and The Ugly

A Linux system could easily have thousands of types, so trying to write a policy using individual **access rules** to grant types permissions is infeasible. Currently, **m4 macros** are used to manage the complexity of an SELinux policy.

Macros are effective at **encapsulating** sets of rules so that they can be used and **reused** together within a policy. However, since macros operate purely through text expansion (similar to the C preprocessor), a macro can have **arbitrary and complex** effects in a policy, unconstrained by the semantics and syntax of the policy language.

As a result, not only can macros be difficult to understand, but they **cannot be handled** by policy analysis tools. Currently, macros are expanded before analysis, leaving it up to the user to apply the results of analysis to the pre-expanded policy.

Introducing SENG

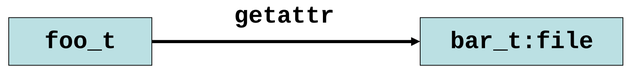
SENG is a new, experimental replacement for the SELinux policy language. It **eliminates** the need for macros by adding support for **well-defined**, **high-level** abstractions that are easy to read, easy to write, and easy to analyze.

Abstract Resources and Permissions

Group together rules that serve a common purpose, such that they can be used together within an allow rule, just like ordinary permissions.

Ordinary Access Rules

```
allow foo_t bar_t:file getattr;
```

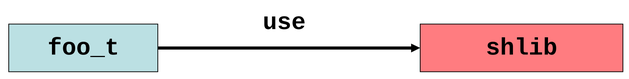


Abstract Resources

```
allow foo_t shlib use;
```

"**allow *$dom* shlib use**" means:

```
allow $dom { root_t usr_t }:dir r_dir_perms;
allow $dom lib_t:lnk_file r_file_perms;
...
```

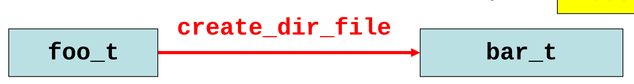


Abstract Permissions

```
allow foo_t bar_t create_dir_file;
```

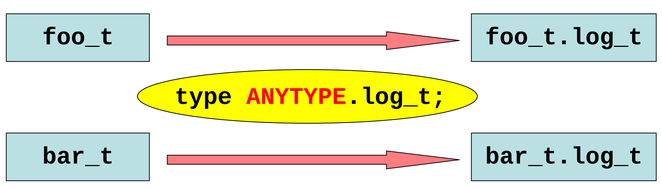
"**allow *$dom $typ* create_dir_file**" means:

```
allow $dom $typ:dir create_dir_perms;
allow $dom $typ:file create_file_perms;
...
```



Type Templates

Automatically generate new types from an existing type or role. Types instantiated from templates can easily be used in abstract resources and elsewhere.



"**allow *$dom* var_log append**" means:

```
allow $dom var_log_t:dir ra_dir_perms;
allow $dom $dom.log_t:file { create ra_file_perms };
type_transition $dom var_log_t $dom.log_t:file;
```

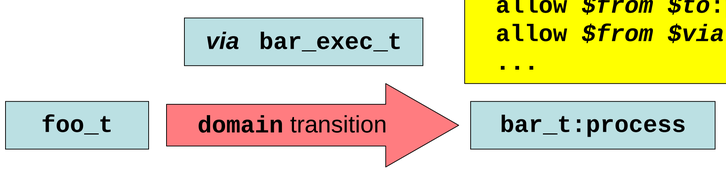
Custom Transitions

Associate a transition with additional supporting rules that automatically apply when the transition is used.

```
type_transition foo_t bar_exec_t
          bar_t:process domain;
```

"**type_transition *$from $via $to*:process domain**" adds:

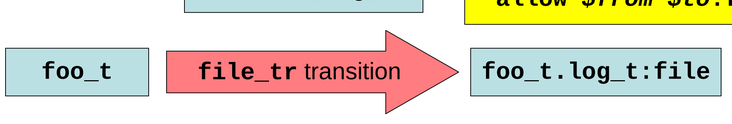
```
allow $from $to:process transition;
allow $from $via:file { read x_file_perms };
...
```




```
type_transition foo_t var_log_t
          foo_t.log_t:file file_tr;
```

"**type_transition *$from $via $to*:file file_tr**" adds:

```
allow $from $via:dir rw_dir_perms;
allow $from $to:file create_file_perms;
```



Current Status

A proof-of-concept **compiler** has been implemented that converts a policy written in SENG to the current policy language.

A representative subset of version 1.26 of the monolithic example SELinux policy has been **rewritten** using SENG. This effort demonstrates that the abstractions SENG provides are effective at **eliminating the need** for most macros.

Future Work

Formal semantics for SENG have yet to be written, although the behavior of the proof-of-concept compiler currently specifies them informally.

SENG was built on top of the **monolithic** SELinux policy. Work is needed to evaluate what additions to SENG may be needed to support recent SELinux developments such as **modular policy** and **MCS/MLS** support.

Although one of SENG's primary design criteria was to support automated analysis of policies, **analysis tools** for SENG have yet to be developed.