Policy Conflict Detection for Cfengine Project Report

Matúš Harvan

February 2, 2007

Abstract

An ATP system, Bliksem, has been used for policy conflict detection in cfengine configurations. For this purpose, predicates corresponding to cfengine configuration directives have been developed, additional rules describing conflicting actions have been defined and a basic translator tool from the cfengine language to the TPTP language has been developed. Although some parts of the cfengine language are omitted by the translator tool and not all possible policy conflicts are detected, the project shows that using ATP systems for policy conflict detection in cfengine configurations is a feasible approach.

1 Introduction

The goal of the described project was to investigate policy conflicts detection in a cfengine [1] configuration, i.e. to detect contradicting actions, using standard automated theorem provers (ATP). Being able to detect policy conflicts and contradicting actions in an automated way would be helpful especially for sites with complex cfengine configurations which are rather cumbersome for manual inspection.

To allow the usage of ATP systems, first-order logic expressions and predicates have been developed in the TPTP language [3] to represent cfengine language constructs and information about contradicting actions has been added in the form of background knowledge. The theoretical developments have been implemented in the form of a basic translator tool, taking cfengine configuration as input and producing TPTP language as output. Using the TPTP language rather than translating directly to the native language of a particular ATP system allows to choose from a variety of standard automated theorem provers. The particular theorem prover used in this project was Bliksem.

The rest of this document is structured as follows. Section 2 provides a short overview of cfengine, Section 3 describes the translator from the cfengine configuration language to the TPTP language, a short description of the TPTP library can be found in section 4, Section 5 provides details motivating the choice of a particular ATP system, Bliksem, Section 6 shows a simple cfengine configuration example with the corresponding TPTP translation and the report concludes in Section 7.

2 Cfengine

Cfengine [1] is a system administration tool for distributed policy-based configuration management. It uses a high level declarative configuration language to describe the desired state of several hosts in one or several central files. Each of the managed hosts runs an autonomous agent to download the configuration files from a central server, evaluate them for the particular host, determine deviations from the desired state and perform actions needed to converge to the desired state. The high level language used by cfengine encourages to specify the desired state rather than actions to be performed. It tries to hide the differences between various Unix systems, eliminates the hard to read if-then-else constructs and tests which would clutter a home-grown script and results in more concise and easier to read configuration files. Nevertheless, it is possible to call shell scripts from cfengine if the need arises. Hosts can be assigned to various classes, allowing for management of a large number of hosts while still being able to pinpoint specific systems. Cfengine focuses on a few key functions rather than trying to do everything. These functions are network interface configuration, text files editing, symbolic links creation and maintenance, ownership and permissions of files, cleaning up junk files, NFS file system mounting and various sanity checks. As Unix-like hosts are configured mainly by modifying plain text files, the file editing functionality is indeed very powerful.

In this project, only cfengine 2 was considered, in particular version 2.2.21. The new cfengine 3 and promise theory [2] were not considered.

3 Cfengine to TPTP Translator

A tool for translating cfengine configuration files into the TPTP language has been developed. First part of the work was development of logical expressions and predicates for representing the various cfengine configuration statements in the TPTP language. The other part was of a rather practical nature, where the theoretical resutls were used to implement the translator tool. As cfengine already contains a parser for the cfengine language, its code was used as the basis for the translator implementation. The cfengine parser is written in `lex & yacc`, with the rest being written in C.

The logical counterparts for cfengine configuration directives will be described in more detail. For the sake of readability, these will be described using standard logic notation rather than the TPTP notation. A cfengine configuration basically consists of a control part defining various variables and parameters, a part assigning hosts to groups and a part with action statements describing actions to be taken on the managed hosts or the desired state of these hosts.

3.1 Group Membership

The control part is mostly ignored by the translator except for the assignment of hosts to groups. In cfengine it starts with the `groups:` keyword and membership information for each group follows in the form

```
group = ( memeberhip information )
```

In the membership information part, other groups or hosts can be listed. By prepending a host or group with a minus sign, their membership is negated, allowing e.g. the selection of all but one host from a group. NIS netgroups can be used with cfengine as well, but are ignored by the translator as it does not hook into the NIS system. The representation of group membership is based on the *ingroup(Group, Host)* predicate. The cfengine expression is translated then into the following expression

$$\forall \text{Host} : (\text{conjunction of negated membership}) \wedge (\text{disjunction of non-negated membership}) \Rightarrow (\text{ingroup(group,Host)}$$

For example the cfengine construct

```
group1 = ( group2 -host1 host5 )
```

would be translated to

$$\forall H : (\neg \text{ingroup}(\text{host1}, H)) \wedge (\text{ingroup}(\text{group1}, H) \vee \text{ingroup}(\text{host5}, H)) \Rightarrow (\text{ingroup(group,H)}$$

It should be noted that the treatment of groups by cfengine is slightly different from the translator. Cfengine is usually run on each managed host, so it only has to determine in which groups that particular host is. However, the translator has to determine group membership for each host. Due to implementation details, cfengine accepts even expressions of the form `group1 = ( host1 -host1 host1 )`. Here, cfengine would simply use the right-most piece of information and `group1` would contain `host1`. The translator, on the other hand, would use all constraints, resulting in an empty `group1` given the expression

$$\forall H : (\neg \text{ingroup}(\text{group1}, H)) \wedge (\text{ingroup}(\text{group1}, H) \vee \text{ingroup}(\text{group1}, H)) \Rightarrow (\text{ingroup(group1,H)}$$

For most purposes hosts are treated as groups and implicitly each host is a member of its own group. This is represented by the background knowledge

$$\forall H : \text{ingroup}(H, H)$$

Furthermore, for each group an expression isgroup(group_name) is added. The motivation for adding the *isgroup* predicate is to allow distinguishing between hosts and groups should the need arise.

Furthermore, for each host or group encountered, the *isgrouporhost(group_or_host_name)* predicate is added. This predicate is useful in quantified formulas.

3.2 Actions

Cfengine actions are represented as logic predicates. Due to the limited time frame of the project, translation of all actions has not been implemented. Furthermore, the translation omits several details or options for most of the actions as these options were not considered necessary for detecting the basic policy conflicts. However, the predicates could easily be extended to contain the omitted details. An overview of the predicates and corresponding cfengine actions can be found in Table 1.

cfengine action	logical predicate
`resolve`	resolve(H)
`files`	files(H, file)
`tidy`	tidy(H, path, pattern, recursion)
`disable`	disable(H, file)
`shellcommands`	shellcommand(H, command)
`packages`	package(H, package, action)
`editfiles`	editfile(H, file)

Table 1: Cfengine actions and their translations. H represents the host to which the action applies.

Actions in cfengine are usually restricted to certain hosts or groups of hosts. Therefore, a complete cfengine action statement is translated into an expression of the form

$$\forall H : \text{group restrictions} \Rightarrow \text{action}$$

where H stands for the particular hosts and is used in both, group restrictions and action. For example, the following cfengine configuration snippet

```
editfiles:
   (group1.!group2)|group3::
      { /etc/hosts
         ...
      }
```

would be translated to

$$\forall H : ((\text{ingroup}(\text{group1}, H) \wedge \neg\text{ingroup}(\text{group2}, H)) \vee \text{ingroup}(\text{group3}, H)) \\ \Rightarrow \text{editfile}(H, \text{'/etc/hosts'})$$

Cfengine allows to split the configuration into several files. The *import* action can then be used to include the other configuration files. As the import action can also be restricted to only certain hosts or groups, these restrictions would apply to every action in the imported file. In order to correctly handle such restrictions, the translator records import restrictions into a queue. When translating actions from an imported file, these restrictions are added to the usual restrictions in the action statements.

The arguments to predicates in the translated expressions are quoted to ensure they would be treated as constants rather than variables in the TPTP translation. This should also mitigate problems arising from the usage of special characters in these arguments.

3.3 Background Knowledge

While the translated expressions can be understood by theorem provers, they are not necessarily contradicting in the logical sense. In order to allow theorem provers to detect policy conflicts or contradictions, additional rules are needed to describe which actions are mutually contradicting. As an example, consider the following two actions

```
edit file A
disable file A
```

Without the background knowledge that removal and editing of the same file do not go well together, the theorem prover would not know these actions are contradicting. Therefore, additional rules need to be defined and added to the translations as background knowledge. The development of such rules basically requires a human matching all possible configuration directives with each other. Several such rules have been defined. In general, they are of the form

$$\neg\exists(\text{Host}, \ldots) : \text{isgrouporhost}(\text{Host}) \wedge \text{action1} \wedge \text{action2} \ldots$$

The contradiction rules defined so far are:

$\neg\exists H, F : \text{isgrouporhost}(H) \wedge \text{editfile}(H, F) \wedge \text{disable}(H, F)$

$\neg\exists H : \text{isgrouporhost}(H) \wedge \text{disable}(H, \text{"/etc/resolv.conf"}) \wedge \text{resolve}(H)$

$\neg\exists H, F : \text{isgrouporhost}(H) \wedge \text{files}(H, F) \wedge \text{disable}(H, F)$

$\neg\exists H, A, B : \text{isgrouporhost}(H) \wedge \text{link}(H, A, B) \wedge \text{disable}(H, A)$

$\neg\exists H, A, B : \text{isgrouporhost}(H) \wedge \text{link}(H, A, B) \wedge \text{disable}(H, B)$

$\neg\exists H, P : \text{isgrouporhost}(H) \wedge \text{package}(H, P, \text{install}) \wedge \text{package}(H, P, \text{remove})$

The motivation for these equations is that following combinations of actions are contradictory: editing and disabling (removing) a file, disabling `/etc/resolv.conf` and using the `resolve` action, modifying permissions of and disabling a file, removing a link source or target and setting up the link, and installing and removing the same package.

Besides the contradiction rules, other background knowledge is needed for group membership. As both hosts and groups are implicitly treated as groups, the knowledge that each host is a member of its group is needed.

$$\forall H : \text{isgrouporhost}(H) \Rightarrow \text{ingroup}(H, H)$$

Furthermore, each host is a member of the special *any* group and each subgroup is its subgroup.

$$\forall H : \text{isgrouporhost}(H) \Rightarrow \text{ingroup}(\text{any}, H)$$

Technically, a file containing the background knowledge is appended to the output of the translator before it is passed further on to a theorem prover.

3.4 Limitations

Due to the limited time frame of the project, only a subset of the cfengine language is translated, several details have been omitted and various issues were not addressed. Although some policy conflicts can already be detected, many were not addressed. Some of the limitations and shortcomings of the project at its current state will be described.

Variables and arrays are not supported. The main problem is that variables would take different values for different hosts. While this problem could be solved, the values of variables often depend on the result of a shell script on the managed host.

The `IPRange` and `HostRange` commands for defining ranges of hosts are not supported.

Several cfengine commands accept wildcards and regular expressions as parameters. While it might be possible to decide whether two wildcard or regular expressions match the same string or file, or at least whether they match a particular file, they are currently treated as constant strings. One of the problems is that the matching might have to be performed by the theorem prover rather than the translator. Expressing such functionality in the TPTP language seems to be difficult if not impossible. A possibility might be to add formulas with wildcard or regular expressions replaced by each matching constant string seen in the cfengine configuration.

Group membership may be activated dynamically if a process is or not found running, a package has been installed or a file was edited. As the configuration is only parsed, but not executed on particular hosts, hosts would simply not be added to such dynamic classes. For similar reasons membership in time classes is also not evaluated. A possibility might be to check both branches for policy conflicts.

Cfengine directives `strategies`, `required`, `disks`, `acl`, `alerts`, `method` and nfs-related functionality such as mounting and unmounting are ignored by the translator.

Membership in so-called hard classes is not known from cfengine config files. These classes are based on the IP address of hosts or type and version of operating system, which unless added as background knowledge, are not available.

4 TPTP

TPTP is a logic language understood by several automated theorem proving (ATP) systems. Furthermore, translation programs and scripts exist for translating problems from the TPTP language to other languages used by several ATP systems. The TPTP language not only allows for writing ATP problems, but can also accommodate for ATP solutions. A BNF definition of the language is available in [3].

The language allows problems to be stated using first-order form and conjunctive normal norm. For this project, the first-order form was used. All formulas need to have a name and type assigned to them as well. The type of formulas was set to *axiom* for all formulas. For the name a dummy constant was used.

The TPTP Problem Library (Thousands of Problems for Theorem Provers) is a library of test problems for automated theorem proving systems. The distribution also includes a prolog-based script for translating the TPTP language to the native input languages of several theorem provers.

5 Automated Theorem Prover

Several ATP systems have been considered. The evaluation criteria for choosing a particular theorem prover were the ease of installation on a FreeBSD system, i.e. existence of a FreeBSD port or an easily working build process. Furthermore, the theorem prover should either accept the TPTP language directly or a

translator from TPTP to the native input language should be available, ideally provided by the translation script in the TPTP library distribution. The ATP systems considered will be shortly described.

Otter was easy to install from the FreeBSD ports and the TPTP translator supports the Otter language. However, the translator was crashing with prolog exceptions on the TPTP examples, discouraging its further use in the project.

Spass was not supported by FreeBSD ports and sources of *Vampire* could not be easily found on the web.

The winner of the ATP systems was *Bliksem* with an easy build `./configure; make` process and a functional TPTP translator. A minor problem in the build process was solved by a `touch inoutput.c` in the Bliksem's source directory. Another problem was that Bliksem could not cope with too long strings in predicate arguments. The problem occurred with longer shell commands. Except for that, Bliksem could process the translated cfengine configuration in the `Auto` mode and on several tested examples has correctly decided whether it contained a contradiction. The answer "found a proof!" can be interpreted as "a contradiction was found" and the "found a saturation!" answer can be interpreted as no contradiction was found.

One might wonder why to bother with the TPTP language at all when it has to be translated to the Bliksem language anyway. Although it might be easier to translate to the Bliksem language directly, using the TPTP language allows to easily change the ATP system.

6 Example

A simple example with contradicting actions will be presented. The cfengine configuration is in Figure 1 and its translation to TPTP is in Figure 2. Translation to Bliksem is omitted. Using Bliksem, a contradiction was found in the example. The contradicting actions are that `host2` both edits and disables `/tmp/filea`. After removing host2 from group2 and rerunning the example, no contradiction was found.

```
groups:
        group1 = ( host1 host2 )
        group2 = ( host2 host3 )

editfiles:
        group1::
        { /tmp/filea
                AutoCreate
        }

disable:
        group2::
                /tmp/filea
```

Figure 1: Cfengine configuration example with a contradiction

```
fof(group_memebership,axiom,! [H] : (((ingroup('host2',H) | ingroup('host1',H)))
 => ingroup('group1',H) )).
fof(group,axiom,isgroup(group1)).
fof(group_memebership,axiom,! [H] : (((ingroup('host3',H) | ingroup('host2',H)))
 => ingroup('group2',H) )).
fof(group,axiom,isgroup(group2)).
fof(action,axiom,! [H] : (((ingroup(group1,H))) => editfile(H,'"/tmp/filea"'))).
fof(action,axiom,! [H] : (((ingroup(group2,H))) => disable(H, '"/tmp/filea"'))).
fof(grouporhost,axiom,isgrouporhost(group2)).
fof(grouporhost,axiom,isgrouporhost(group1)).
fof(grouporhost,axiom,isgrouporhost(host3)).
fof(grouporhost,axiom,isgrouporhost(host1)).
fof(grouporhost,axiom,isgrouporhost(host2)).
/* every host is a member of it's own group */
fof(group_memebership,axiom,! [HOST] : (isgrouporhost(HOST) => ingroup(HOST,HOST
))).
/* everyone is in the "any" group */
fof(group_memebership,axiom,! [HOST] : (isgrouporhost(HOST) => ingroup(any,HOST)
)).

/* entities which are not groups are hosts */
/* fof(hosts,axiom,! [HOST,GROUP] : ((ingroup(GROUP,HOST) & ~isgroup(HOST) => is
host(HOST)))). */

/* fof(hosts,axiom,! [A,GROUP] : ((ingroup(GROUP,A) => isgrouporhost(A)))). */

/* conflicts */
fof(bg1,axiom,~(? [H,F] : (isgrouporhost(H) & editfile(H,F) & disable(H,F)) )).

fof(bg1,axiom,~(? [H] : (isgrouporhost(H) & disable(H,'"/etc/resolv.conf"') & re
solve(H)) )).

fof(bg1,axiom,~(? [H,F] : (isgrouporhost(H) & file(H,F) & disable(H,F) ))).

fof(bg1,axiom,~(? [H,A,B] : (isgrouporhost(H) & link(H,A,B) & disable(H,A) ))).
fof(bg1,axiom,~(? [H,A,B] : (isgrouporhost(H) & link(H,A,B) & disable(H,B) ))).

fof(bg1,axiom,~(? [H,P] : (isgrouporhost(H) & package(H,P,'install') & package(H
,P,'remove')) )).
```

Figure 2: TPTP translation of the cfengine configuration from Figure 1.

7 Conclusion

The described project has shown that it is possible to use standard automated theorem provers for policy conflict detection in cfengine configurations. For this purpose first-order logic predicates and expressions have been defined as translations of cfengine configuration directives, background knowledge describing conflicting actions has been defined and a translator from cfengine configurations to the TPTP language has been developed. Using the Bliksem theorem prover, the translations have been checked for policy conflicts, existence of which has been correctly identified in the tested examples.

Due to the limited time frame of the project, only basic functionality is available and the translator as implemented so far has several limitations and does not preserve all the details of the cfengine language. The background knowledge database could be enlarged with more complex policy conflict definitions. A more detailed description of the limitations is presented in Section 3.4. However, many of these limitations were caused by the limited time frame of the project rather than a flaw in the approach and therefore were left for further work. The Bliksem translator provides also details about the proof it finds, so it may be possible to even pinpoint the contradicting cfengine actions rather than just give a yes or no answer.

At its current stage, the translator was capable of processing the cfengine configuration used to manage the approximately 20 hosts administered by the Computer Networks and Distributed Systems group at International University Bremen.

A completely different approach to detection of contradicting actions might be to simply count how frequently each action is executed. As contradicting actions should be executed on each run of cfengine, they could be detected by a rather high execution frequency. This approach might also be more practical in the sense that it is easy to implement and does not require a human to set up a background knowledge describing conflicting actions. The drawback is that configurations could not be checked without actually executing them.

References

[1] cfengine. `www.cfengine.org`.

[2] Promise theory. `http://eternity.iu.hio.no/promises.php`.

[3] TPTP language. `http://www.cs.miami.edu/~tptp/cgi-bin/DVTPTP2WWW/view_file.pl?Category=Documents&File=SyntaxBNF`.