Configuring Firewalls

An XML-based Approach to Modelling and Implementing Firewall Configurations

Simon R. Chudley and Ulrich Ultes-Nitsche
Department of Electronics and Computer Science, University of Southampton, Southampton, SO17 1BJ, United Kingdom, e-mail: {src299,uun}@ecs.soton.ac.uk

Key words: XML-based modelling of a firewall, automatic configuration generation

Abstract: We present in this paper an approach for modelling the security infrastructure of a network using XML. The modelled system can then be validated on the XML level. From validated models, configurations of concrete nodes, such as firewalls, can be generated automatically.

1. INTRODUCTION

Modern networking infrastructures often involve complex interactions between participating network nodes and running services. It is often the case that systems feature nodes running varying implementation of the same service. However, these sub-systems still provide similar abstracted functionality to end-users, leading to problems in the configuration and management of the overall system.

This paper proposes a solution to the management of a communications infrastructure involving varying nodes and service implementations. Considering in particular the service of a firewall, it will introduce the idea of an XML-based tool to allow firewall descriptions to be generated and manipulated, whilst keeping the specification of the firewall as abstracted as possible. The tool is then able to perform a level of translation to compile the abstracted description down to implementation-level configuration files. These can then be applied directly to the firewall running on a network node. In addition, this gives rise to simulate any given firewall configuration prior to its implementation to validate its configuration.

Taking the firewalls IPFW [5] and IPFilter [3] as an example, configuration differs between them, even though the required behaviour is the same. The following two configuration scripts both give the same functionality, to prevent packets from private networks coming in via their public network interface, `tun0`, as described in RFC 1918 [7].

Example IPFilter configuration:

```
block in quick on tun0 from 192.168.0.0/16 to any
block in quick on tun0 from 172.16.0.0/12 to any
block in quick on tun0 from 10.0.0.0/8 to any
pass in all
```

Example IPFW configuration:

```
add deny all from 192.168.0.0/16 to any in recv tun0
add deny all from 172.16.0.0/12 to any in recv tun0
add deny all from 10.0.0.0/8 to any in recv tun0
add pass all from any to any
```

These two configurations differ purely on format; hence they could be automatically generated from an abstracted description. However, this may not always be the case as rule descriptions in IPFW and IPFilter differ also in functionality. Both the specification of generic firewall concepts as well as handling settings available in one firewall but not another will be addressed in this paper.

2. RELATED WORK

The Filter Compiler Language project [6] has successfully implemented a conversion process from a set description to various firewall configurations. The approach taken uses the C pre-processor to execute the conversions. This allows the abstracted description to be generated using `if` statements, and variable mappings to be made. An example of this is as follows.

```
        if ( in ) then {
            set protocol tcp
            if ( from host BAR and opening ) then {
                  block .
            }
            if ( from foo and to host bar ) then {
                  log body and block .
            }
            if ( to port 2049 ) then {
                  log and block .
}
            pass .
        }
```

However, on closer examination, this approach would not be applicable to a wide range of services, as it relies on the fact that the configurations are rule based. Also, as one aim of this project is to introduce the possibility of simulation, such a service description would not be rich enough.

Another product was identified called the Firewall Builder [4]. This uses a similar approach to that proposed by this project, where nodes and other network elements are described using XML descriptions, and a GUI editor is used to configure the firewall. Such an approach includes all processing functionality within the GUI editor, and the XML files purely used as storage for the system. This project aims to introduce more advanced dynamic configuration operations into the XML definitions themselves.

In addition, the Firewall-Builder method is still specific to a single service, but will be useful as a comparison during the development of the firewall service. This project also aims to create an architecture using Java and XML, to ensure that it remains as system independent as possible.

3. THE SYSTEM

The key component to this system is the repository of network nodes and service descriptions. These represent abstracted behaviour of the various network elements and provide extended functionality to aid in the overall configuration task, e.g. referencing service behaviour defined within an external library enables complete descriptions to be built up by reusing previously tested and verified constructs. Function calls, in addition, can be used to evaluate abstracted service descriptions, in order to automatically configure elements of services running on the same or other nodes.

Taking the firewall example, we aim to initially analyse how the service responds to sequences of IP packets of varying forms. Furthering this, the ability to track whole sessions across the simulated system is desired, hence locating the nodes that may prevent such communication from taking place. Common network threats to enterprise security could be emulated within the simulation process, establishing the extra firewall rules, for example, that may be required to protect against such vulnerabilities.

Once the desired behaviour of the abstracted service descriptions has been achieved via the simulation process, the next task is to translate these into configuration files that can be directly applied to applications running on nodes. This is a two-stage process, featuring an initial verification step, then the translation itself. During verification the aim is to establish whether there are any restrictions in the translation process, such as the use of features not supported by the chosen end level firewall system. Post verification, possibly after making configuration changes due to identified restrictions, the translation process executes. With a rich enough translation process, generating rules for two differing firewall implementation will still give us the overall behaviour we expected.

3.1 Overall Structure of System

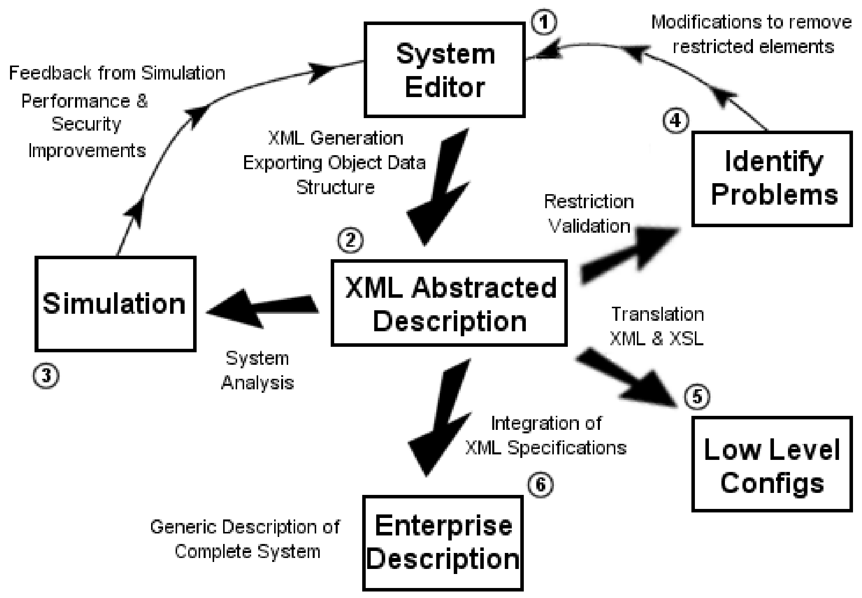
One primary aim of this project was to provide an architecture that could generically be applied over a wide service base. It should be able to transparently support the addition of new translations from the abstracted description to implementation level configurations, and expand easily to cover new services.

The most versatile approach to use is a fully object oriented design. A major aspect of the entities' responsibilities is to parse and generate the XML to represent themselves, being able to expand variable mappings and function calls for example. The idea is to maintain maximum encapsulation within the service objects, so that new services can be added by the alteration of the minimal number of external objects.

The diagram on the following page, figure 1, shows the overall system structure and processes that occur. Initially, the user's task is to generate descriptions of their network (stage 1), the nodes within it and the services they intend to configure. At this stage all descriptions use an abstracted syntax, implying that their specification is not tied to a single service implementation or architecture.

At stage 2 within the diagram, the system will maintain an XML abstracted description of all elements. Such syntax is used to maintain state; hence these descriptions can be stored and recalled from disk. This stage forms the central repository of the entire system, with other stages referring to it when performing further processing. XML service descriptions will contain unresolved external references and function calls, as these are evaluated prior to translation/simulation.

With the desired behaviour of the nodes and services encapsulated within abstracted XML descriptions, simulation can be performed to validate specifications prior to implementation. Simulation is outside the scope of this project, but research is in place to develop such a tool. It is envisioned that feedback criteria such as performance and security improvements will be fed back to the system editor, allowing the user to incorporate these new rules into their overall specifications.



System Editor
Modifications to remove restricted elements
Feedback from Simulation
Performance & Security Improvements
XML Generation Exporting Object Data Structure
Identify Problems
Restriction Validation
XML Abstracted Description
Simulation
Translation XML & XSL
System Analysis
Integration of XML Specifications
Low Level Configs
Generic Description of Complete System
Enterprise Description
1
2
3
4
5
6

Figure 1. Overview of the System Structure

Stages 4 and 5 represent the process of creating implementation level configuration files from the descriptions now stored in stage 2, hence outputting files that can be directly loaded into service applications. Initially a level of restriction validation is performed. This takes a set of restriction filter rules, describing possible problems in the translation process, and reports to the user the location of such restrictions, and the XML elements they affect. Such a filter is required as different service solutions provide different functionality, and the user should be notified of any undesired changes in behaviour. As with simulation, the system editor will be used to make design changes in response to located restrictions.

Translation, in stage 5, aims to convert these descriptions to implementation level configuration files. At this stage, the user will be aware of the expected behaviour of the to-be implemented services, and have knowledge of any included restrictions. Finally, stage 6 represents the enterprise description, implying that the combination of all element specifications define the overall abstracted behaviour of the entire system.

4. SERVICE TRANSLATION PROCESS

4.1 Specifying the Translation

Translation is the process of converting the abstracted service description into rules and configuration elements that can be directly applied to the desired service implementation. The process used to achieve this is XSLT using XSL style sheets [8]. For each implementation, such as IPFilter or IPFW for firewalls, an XSL sheet is created to perform the transformation to the end level rules. An XML file of the same name is used to act as a wrapper, providing extra functionality such as detailing limitations of the translation, described in the next section.

The aim is to create a rich enough translation between the abstracted description and the final configurations, so that the behaviour directly reflects that of the original XML specification. If this could be achieved, a node could in fact be swapped with one running a different implementation of a given service. Generating the configurations for this new node should give the same functionality as before, enabling services to be swapped for performance, security and testing reasons.

4.2 Translation Restriction Calculation

4.2.1 Overview of Restrictions

Due to the level of abstraction introduced in describing the behaviour of network services, there is commonly a mismatch between the two levels of configuration. Features supported by the XML description may translate directly into one implementation level service but not another, which implies that the overall behaviour of the system will not be as the user expected. However, this section describes a process that enables the user to test whether there will be any problems in the translation process. In addition it reports to the user information on the XML elements of their configuration that are restricted, along with textual reasoning.

Such restrictions can only be identified with knowledge of the specific target implementation language, all details of which are stored within the translation XSL file itself. Therefore, in addition to the XSL file, each translation also contains an XML file describing a set of restrictions of that process.

Restrictions are described as mapping a combination of XML configuration elements to some text reason explaining why they are not supported. These will be specified by the creator of the translation, and can contain multiple rules that an XML element must match to be declared as a restriction.

Restriction calculation is performed using a pre-processing style. The full XML output of the service is fed into the pre-processor, specifying which translation we desire to test for restrictions. The pre-processor itself is in fact a serial processing XML filter, using a set of predefined rules (specified as restrictions) to analyse the stream of XML as it passes. Elements that match the rules will be returned to the user, along with the text reason for that restriction matching.

The restriction filter provides a rich syntax for matching XML configuration elements, with the intent of being able to specify any restrictions that may occur. The basic element of a restriction maps from the status of an attribute or element to a text reason that is outputted on the match of such an attribute. However, combinations of these can be made and put into groups forming logical operations such as AND/OR. Further groups can be created consisting of other groups and sub-elements allowing complex node configurations to be matched.

4.2.2 Restriction Rule Specification

The following restriction segment shows matching on the existence of a sub-element of XML. This will therefore match any `Rules` that have one or more `protocol` sub-elements within them. A specific filter syntax is used to address restrictions:

```
<RestrictionElement
  name="Firewall::Filter:FirewallConstruct/Rule/protocol"
  reason="Matching existence of sub-element"/>
```

The following restriction matches `src` elements of XML that have an attribute `address` defined, with or without a specific value. The lower rule matches `interface` specifications that don't have an attribute `via` defined.

```
<RestrictionElement
 name="Firewall::Filter:FirewallConstruct/Rule/src@address"
 reason="Existence of element attribute"/>
<RestrictionElement
 name="Firewall::Filter:FirewallConstruct/Rule/interface@!via"
 reason="Absence of element attribute"/>
```

There are many more restriction rules that one may specify. Their entire presentation would go far beyond the scope of this paper. The interested reader is referred to [2].

4.2.3 Pre-processor Output

The aim of the pre-processing stage is not just to identify the existence of possible restrictions, but also to direct the user towards the offending parts of their XML service description. To perform this the package can produce additional information when restrictions are found to help the user identify the location of the problems.

Following is an example of the matched output produced by the pre-processor.

```
<Restriction line="57" col="18"
      path="/Firewall(1)/FirewallConstruct(1)/Rule(7)"
      reason="Matching some random group of data">
   <fw:Rule RuleID="1007" >
      <fw:action perform="pass"/>
      <fw:protocol type="other" name="udp"/>
            <fw:src type="ip" mask="255.255.255.255"
         address="192.168.2.100"/>
            <fw:dst type="ip" mask="255.255.255.0"
         address="192.168.2.0"/>
      <fw:interface direction="out"/>
   </fw:Rule>
</Restriction>
```

Initially details on the restriction are outputted, including the line and column numbers that the problem XML starts on, and also the exact processing path to that element. This path is detailed by including the index of the various sub-elements that were encountered to get to the start of that element.

In the above example we can conclude that the matched XML element is the seventh `Rule`, within the first `FirewallConstruct` all within the first `Firewall`. As the input XML is just a service, there will be no additional node definitions on the path.

In addition to reporting the path, the package also fetches the actual source XML input, using the reported `path`, and outputs this within the final restriction report. In effect this can report straight back to the user the source of the restriction, allowing them to make the required changes. To aid interpreting systems using this package, such as GUIs, the restriction report is itself valid XML, hence enabling further processing to be performed.

If the restriction element or group specifies the `outputXML` attribute to equal "`no`", then only the path and restriction reason details will be generated with no source XML.

4.2.4 Complications During Filter Process

The ability of the filter pre-processor to generate source XML on restriction match is itself not trivial. As the filter is processing in a serial manner, minimal state is maintained about seen XML, and there is no knowledge of future XML. This implies that the filter is unable to remember matched XML, hence needs to use the stored `path` and fetch the source XML directly from the object structure.

However, fetching the source object can cause complications. Initially the input to the pre-processor was simply a stream of fully expanded XML (implying variables, external constructs and functions have been resolved). It is quite possible that the restriction the package is attempting to locate was defined within an external construct, or even the output of some function call.

The package will attempt to fetch the exact XML element that matched the restriction, which is fine with standard definitions and external constructs. Variable mappings at that exact level are also resolved to provide the user with as much information as possible. However, function calls cannot be inspected by the pre-processor, as they don't return objects (they in fact return another XML stream), so in this situation, the result of the whole function is returned to the user.

4.3 The Translation Process

The next stage is to generate implementation level configuration files. Prior to translation, the fully expanded XML representation of the chosen service is generated, involving the resolution of external constructs and function calls. This is then fed into an XSL processor, along with the XSL style sheet describing the translation for the desired implementation level service. The output from this process will be a series of configuration files that can be applied directly to the intended service. Once implemented, these should have the same behaviour as previous specified in the abstracted XML description.

A translation is described using XSL, mapping combinations of elements within the abstracted description to their lower level syntax. Such a process should be able to convert all elements within the input to their equivalent at this lower level, unless specified otherwise within the restrictions stage.

The following XML segment is a simple rule with various extra `options` definitions. It matches a TCP packet, travelling from anywhere to anywhere, that is a connection set up request (TCP SYN flag set). In addition, the packet must not be set as using source routing (either strict of loose), used for specifying a fixed route of travel for that packet.

```
<fw:Rule Desc="Test Rule" RuleID="100">
   <fw:action perform="pass"/>
   <fw:protocol type="other" name="tcp"/>
   <fw:src type="any"/>
   <fw:dst type="any"/>
   <fw:options setup="true" established="no" gid="200">
      <fw:ipoption spec="lsrr" absent="true"/>
      <fw:ipoption spec="ssrr" absent="true"/>
   </fw:options>
</fw:Rule>
```

When using the input XML rule above, processing with the IPFW translation, the following rule is produced.

```
add 100 pass tcp from any to any setup ipoptions lsrr,ssrr gid 200
```

The IPFilter translation produces the following two rules. Note that the `gid` option is not supported by IPFilter, and hence is removed (refer to section 5.2.5). Another restriction of IPFilter is that rules must have an explicit direction of travel, whereas the abstracted firewall description does not enforce this. The translation process can resolve these conflicts automatically by generating duplicate rules, creating two rules with different IDs and directions of travel.

```
@100 pass in quick proto tcp all flags S/AUPRFS with opt lsrr,opt ssrr
@101 pass out quick proto tcp all flags S/AUPRFS with opt lsrr,opt ssrr
```

5. CONCLUSIONS

We have reported on the firewall specific aspects of a tool development project [1,2] to implement network components automatically from XML specification of network services. From the XML description of a firewall we can configure a particular firewall product automatically using an XML-to-configuration-file translation described in XSL. Should we decide at some stage to change the firewall in our network, we can configure the new firewall product from the same XML specification using the XSL-based translation. The new firewall will then be guaranteed to operate in exactly the same manner as the previous one did.

We have discussed some aspects of the configuration and simulation of firewall using XML specifications. Also we have discussed how to deal with feature available in one firewall but not another. Some technical details would have gone beyond the scope of this paper: We have not discussed, for instance, how the translation process works in details nor have we presented any parts of the XSL. The interested reader will find more information at the project's web-page [2].

One of the major benefits of the system discussed in this paper is the scope for specification-level simulation, and therefore validation, of the firewall's behaviour prior to its concrete implementation and integration within a network: We can first convince ourselves of the correct settings in respect of a given security policy before we make the firewall operational; thus reducing the risk of introducing security holes by testing the firewall in the life network environment. The simulation part of the discussed tool is currently under development. The core simulation functionality has been completed, i.e. simulating the behaviour of a single data packet; more elaborate testing functionality will be developed in the next step.

Until now we have only dealt with the specification and translation of stateless firewalls. It will be another topic for future research to integrate stateful firewalls into our approach. All latest developments will be contained at this projects web-page [2].

6. REFERENCES

[1] S.R. Chudley and U. Ultes-Nitsche. Simulation and Implementation of an E-Commerce Communications Infrastructure using XML Specifications. In: *Proceedings of Business Information Systems 2002 Conference*, Poznan, Poland, 2002.
http://www.ecs.soton.ac.uk/~src299/xmlnetman/bispaper.pdf

[2] S.R. Chudley. *XML abstracted network management*, 2002.
http://www.slyware.com/projects_xmlnetman.shtml

[3] E. B. Conoboy and E. Fichtner. *IP Filter Based Firewalls HOWTO*, 2002.
http://www.obfuscation.org/ipf/ipf-howto.html
http://www.gsp.com/cgi-bin/man.cgi?section=8&topic=ipfw

[4] V. Kurland and V. Zaliva. *Firewall Builder*, 2001.
http://www.fwbuilder.org/

[5] D. Lavigne. *IPFW firewall configuration details*. O'Reilly & Associates 2001.
http://www.onlamp.com/pub/a/bsd/2001/06/01/FreeBSD_Basics.html

[6] D. Reed. *Filter language compiler specification*
http://coombs.anu.edu.au/~avalon/flc.html

[7] Y. Rekhter, B. Moskowitz, D. Karrenberg, G. de Groot, and E. Lear. *Address allocation for private networks (RFC 1918)*, 1996
http://www.faqs.org/rfcs/rfc1918.html

[8] World Wide Web Consortium. XML Style Sheets (XSL)
http://www.w3.org/Style/XSL/