Regions Security Policy (RSP): Applying Regions to Network Security

by

Joshua W Baratz

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2004

Author .... [illegible] ....
Department of Electrical Engineering and Computer Science
May 20, 2004

Certified by .... [illegible] ....
Karen Sollins
Principle Research Staff, CSAIL
Thesis Supervisor

Accepted by .... [illegible] ....
Arthur C. Smith
Chairman, Department Committee on Graduate Students

Regions Security Policy (RSP): Applying Regions to Network Security

by

Joshua W Baratz

Submitted to the Department of Electrical Engineering and Computer Science
on May 20, 2004, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

The Regions network architecture is a new look at network organization that groups nodes into regions based on common purposes. This shift from strict network topology groupings of nodes requires a change in security systems. This thesis designs and implements the Regions Security Policy (RSP). RSP allows a unified security policy to be set across a region, fully controlling data as it enters into, exits from, and transits within a region. In doing so, it brings together several existing security solutions so as to provide security comparable to existing systems that is more likely to function correctly.

Thesis Supervisor: Karen Sollins
Title: Principle Research Staff, CSAIL

Acknowledgments

I would like to thank:

My parents James Baratz and Donna Zaremba, who showed tremendous patience over the years in teaching me how to make my writing fit for human consumption.

My advisor Karen Sollins for providing my with a thesis topic, guidance in exploring that topic, and insight into solutions to thorny problems.

My academic advisor Paul Gray, for reminding me that a thesis is never complete, but at some point, it must be finished.

My friends and pledge brothers Michael Bolin, Ryan Jazayeri, and Brian Stube, with whom I fought Frank O Gehry's best efforts at making a building that strove to impede thesis work.

This work was funded in part by the National Science Foundation under Grant ANI-0137403 and gifts from the Cisco University Research Program.

Contents

Chapter 1

Introduction

Alice is a consultant for the major company ConsultCo. Her primary computer is a laptop, which she uses at the ConsultCo office, the office of their client, Bob Inc., and to work from home. Bob Inc is concerned about information security, and allows Alice access only when her computer is physically connected to their network. ConsultCo's administrators have a policy stating that Alice's computer must be protected by the ConsultCo firewall at all times, and that her connections to company machines must be encrypted when traveling over public links (such as from when working from home, or on site).

Current solutions to this problem are inelegant at both an access and authentication level. In order to get connected to the network, Alice must re-configure her computer's IP settings at every location. She must set up a VPN connection into ConsultCo to fulfill the encrypted link requirement. In order to use the ConsultCo firewalls, this must be the only link that she uses. At home, she must either use this to surf the web, or have her company's administrators set up and maintain a personal firewall on her machine. BobInc network administrators must have similar intense involvement with Alice – providing her with an IP address to use while in the building, and remembering to revoke it when the contract has finished. Finding a way of authenticating Alice is even more problematic. Solutions range from simple passwords on telnet connections, to windows domain logins, to web certificates, and huge ranges of solutions in between.

One of the major problems with these solutions stems from the different levels of abstraction, from policy statement to policy enforcement. The stated security goals are fairly simple - anyone with a basic computing background can understand the idea that connections from A to C must be kept private – third parties must not be able to eavesdrop. This statement is a simple expression of the complex notions of actual connection technology (the encrypted transport), and those of identity and authentication (the idea of Alice and ConsultCo). However, our current network models and technologies operate on a much lower level of abstraction. Network protocols deal with the basic connections, a higher layer piece or pieces of software deal with authentication issues. Consequently, there is intentional separation of the two. A network architecture that bridged this gulf between low level protocols and high level network descriptions could be used to form a single, simple solution to Alice's problem.

Now imagine that Alice is traveling on the day of a local election. The election officials, having unveiled electronic voting in the polling place, want to allow electronic absentee balloting. They require not only a secure, authenticated connection to Alice's computer, but also assurances that she is running the election software they certified, and that no other programs are interfering with or controlling the software. This new requirement has greatly expanded the scope of the problem.

Software currently exists that can guarantee isolation and separation of programs. Software and hardware exists that can remotely attest to the state of an operating system and running programs. As previously mentioned, there are many solutions to assure data integrity and secrecy in communications. While a user could theoretically use all of this technology, the amount of configuration and user intervention necessary is so extensive and unwieldy so as to be impossible for an average user[34].

A technological solution for this scenario must encapsulate the requirements of network security, local machine security, and remote authentication and verification. In order to be useful, it must be much simpler than simply combining these existing solutions.

This thesis proposes a solution in the form of the Regions Security Policy (RSP) framework. RSP is a security framework based loosely on the thoughts and constructs of the Regions network architecture (section 3.1). I begin in Chapter 2 by introducing the motivation behind RSP. The chapter continues with an outline of the goals and philosophies of this framework.

These goals are: simplicity, customizability, transparency, and legacy system support. Simplicity stems from an ease of use requirement; security that is easy to use increases the chances of security systems being used at all, and being used correctly. Customizability recognizes that there are no one-size fits all solutions, and that each implementation has unique requirements. Transparency acknowledges the need for security measures to work correctly, and have independent experts audit and confirm the correctness. Finally, legacy system support allows this framework to build on tried-and-true technologies, and to draw from an installed base.

The philosophies are: use of positive-match, positive-action lists, defaulting to passive denial, and the correct abstraction level. A positive-match, positive-action system is one in which an entity must be explicitly enumerated in order to receive privileges on the system. The notion of default denial is related; any entity not explicitly granted access to the system will be completely denied access. By establishing a common abstraction level, the terms that apply to network oriented security are standardized, simplifying descriptions and comparisons of like technologies.

Having established guidelines for building the RSP framework, Chapter 3 evaluates existing technologies against these guidelines. Many of the these current technologies meet the goal of legacy system support by default, in being mature technologies. However, all have different strengths and weaknesses when compared to the RSP goals. Whenever possible, these strengths are adopted by RSP.

The related works in Chapter 3 form the backdrop for the RSP framework, which is presented in Chapter 4. This chapter first gives a high level overview of the framework, and then explains each construct in detail. Rationale is provided not only for every structure present, but also for the exclusion of several seemingly logical structures. The formal design of the RSP framework allows Chapter 5 to detail the construction of a proof of concept program. This program was built using the Linux

IPSec implementation, the use of which is explained in some depth. This concrete implementation allows a practical analysis of the RSP framework. We find that there are several outstanding issues that arise, largely with respect to integration of legacy systems. This chapter argues that these issues, while vital for some implementations, exist outside of the RSP framework, and affect the framework in a modular way such that they may be replaced. We then conclude this thesis in Chapter 6 with a summary of the thesis, and a discussion of future work.

Chapter 2

Regions Security Policy Design Philosophy

The Regions Security Policy (RSP) results from investigation into using the Regions framework as a security system, allowing analysis of the benefits and drawbacks. RSP is the product of setting several design goals that guide which features the RSP framework should provide, how these features should be provided, and the extent of the features. These goals are both basic constraints on simple constructs, and broader philosophy regarding the overall framework. The basic driving goals are; simplicity, customizability, transparency, and legacy system support.

The requirement for a simple system stems from practical experience with traditional software systems. The greater the complexity of a system, the more likely the occurrence of implementation errors. In any security system, a single implementation error can be disastrous. Once data has been compromised a single time, there is no way of controlling its spread. Thus the simpler the design, the lower the likelihood of an error, either in implementation of software or in the construction of the actual policies. This distinction arises from experience reading and writing IPTables[24] policies. IPTables is a popular firewalling program for unix based systems. The configuration is very low level with policies specifying lists of IP addresses and port numbers. Even authors who are considered authoritative on the subject warn that in anything other than a simple policy, it is difficult to track the true behavior of a policy, and it is difficult for the policy writer to assure that the policy meets intentions[9]. By making simplicity a primary design goal, I hope to avoid such problems with Regions.[1]

Despite the goal of simplicity, any security framework must also thoroughly cover the problem domain space. This completeness guarantee assures that a system delivers the promised security. This goal may also be thought of as emphasizing the choice of abstraction level. Once the scope of the RSP framework is determined, the structure of the framework should accurately reflect this level. In organizational discussions, abstraction is usually compared to altitude - a 30,000 foot view covers the same area as a 1,000 ft view, but with less detail. Similarly, importance must be paid to the abstraction level so that it provides all of the detail needed, at the correct level of specificity.

The goal of customize-ability is both a corollary of the completeness goal and a result of the philosophy regarding the scope of coverage (as explained in section 2.3). There is no way to predict all of the situations in which RSP will be used[3], and this reality must be reflected in the framework. This goal serves to set the abstraction level - anything too detailed risks limiting the usefulness of RSPs.

The usefulness of this framework is also limited by the adoption of Regions as a network paradigm. The internet and large scale networks are in general at a point where technologies and standards evolve slowly. Massive, flag day style changeovers are very rare[4], so new technologies should be able to build on, interact with, and evolve from current and legacy systems. There are currently many different network and security technologies, running on heterogeneous equipment. To assure deployment, RSP should support as many of these as possible.

Incorporating existing technology into the RSP framework also serves to improve security. Many security technologies, especially those involving cryptography, are complicated, and difficult to implement correctly. Incorrect implementations easily

[1]This design goal is best summarized by inventor Antoine de Saint-Exupéry, who commented: "A designer knows he has achieved perfection not when there is nothing left to add, but when there is nothing left to take away".[2] [29]

[3]More generally, we hope that the Regions framework and the generalized RSP framework encourages innovation in this field.

[4]See IPv6 for example.

compromise information; as former NSA cryptographer Robert H Morris famously noted, the first rule of cryptanalysis is to check for plaintext[12]. By incorporating proven designs and implementations, RSP provides greater assurances of correctly operating security. This inclusion also lowers implementation costs and effort, by allowing legacy software to be used within the RSP framework.

The desire for correct operation that drives the inclusion of existing technology also influences the final goal, which is transparency. Conventional thinking about cryptography systems holds that implementations should be independently verified by experts to ensure security. Likewise, the RSP framework and its implementations should lend themselves to verification. While this goal of transparency is closely related to the goal of simplicity, the inclusion of many cryptographic systems elevates the importance so as to be listed separately.

The formulation of security design goals provided a general guideline for framework construction, but these goals were in turn answers to more fundamental questions. To maintain a consistency in design, it was necessary to take broad positions, resulting in the underlying philosophy of RSP. Explained below, these positions cover approaches to general permissiveness and the use of major design constructs.

2.1 Default to Passive Denial

In any scenario, the default system action has a great effect on the security of the entire system. In any rule based system, it is very difficult to predict and write a rule for every possible situation that will be covered. In reaction to this reality, the RSP framework is based on a default negative-action. The term negative-action is intended in a passive sense - an event or action that does not have an appropriate rule results in the RSP system doing nothing, as opposed to triggering an action. In the IPtables domain, this is equivalent to having a default deny on all incoming connections, so that an incoming connection that doesn't match any rule-set is dropped.

This policy of passive negative-action is the most secure. As discussed at the beginning of this chapter, a single instance of information release fully compromises that information. Allowing information transfer both intra- and inter- regionally requires an active rule-set, thus requiring a policy writer to explicitly authorize the transfer. The policy maker is thus forced to consider the implications of every rule, and have a firmer grasp of the ramifications of the policy.

The passive negative-action policy shows additional benefits when analyzing typical attack vectors. Denial of Service attacks (DoS) occur when a target host is overwhelmed with false requests, which consume resources needed by legitimate requests. The fewer false requests processed, the less overhead consumed. The passive negative-action default in RSP facilitates this. By encouraging moving preliminary authentication to a lower level in the communication stack, fewer false requests get through to the target application. Another recent widespread class of attack has been worms which exploit vulnerabilities in unneeded server programs. These server programs are often running unbeknownst to users and are often unnecessary.[5] A system in which connections are disallowed by default would severely reduce the impact of such attacks.

2.2 Positive-Match, Negative-Action lists

Positive-match, negative-action lists are ubiquitous both in life and computing. Firewall deny rules, spam blacklists, and the local store's list of bad check passers all serve the same purpose – upon positively identifying an entity against a set list, they deny that entity the ability to continue the requested transaction. This paper uses the term positive-match, negative-action to highlight the workings of the system, and draw distinctions between alternatives. In the firewall and check payment examples, whenever an entity requests a resource from a verifier, the entity must present identification. Upon positively-matching an entity against a list, the verifier then denies the requested resource. Semantically, this system operates in a manner equivalent to negative-match, positive-action lists - if an entity's identification does not match any

[5]The Blaster[18] worm is a classic example. Most users do not need RPC running, and many were unaware that it even was running. Those systems that have a legitimate need for this vulnerable service are more likely to have patched it.

on the list, the requested resource is granted. This system gives the appearance of providing a level of security - by stopping potentially problematic transactions before they take place. In reality, positive-match negative-action lists represent at most a convenience, and at worst a lessening of actual security by giving a false sense of the protection they provide.

The illusion of security is greatest in firewalls. Many have argued that firewalls may reduce overall security by lulling internal administrators into a false sense of confidence, letting them delay patching systems. A similar argument can be made for positive-match negative-action lists. These lists are often devised to deal with nuisance adversaries – one colleague of mine blocks all SMTP traffic originating from Asia – but do very little to stem a dedicated attack. Adding a deny rule may prevent an unskilled adversary from connecting to a target host, but these adversaries are not a threat - any adversary stopped by this will likely be using out of date or simple attacks that should be ineffective against patched systems. The skilled, dangerous adversaries can subvert the deny lists using two general methods. For simple non interactive attacks, a spoofed source address will easily avoid deny lists (in order to be easily maintained, deny lists cover a small subset of the address space – if an accept list were smaller, most administrators would use one). Interactive attacks require slightly more work for the attacker. There are currently over 23.5 million home machines on high speed connections (cable modems or DSL) in the US alone, and that number is growing at 18% each year.[14] These hosts are largely poorly administered by users with limited knowledge of security practices. [7] Connectivity with these networks must be maintained, as many valuable customers, telecommuting employees, or others with a valid reason for communication use these networks[6]. However, these same machines are easy targets for viruses, worms, and trojan horses. An attacker can easily build up an entire network of zombie machines – compromised hosts that are fully, remotely controllable by the attacker.[21] These machines can be used to circumvent deny lists, and provide a nearly untraceable platform from which to launch attacks.

[6]Arbitrarily blocking out large sections of the address space also violates the end-to-end principle.

Spam blacklists present a similar problem. One common way of blocking spam is by use of a Domain Name Service Real-time Block List (DNSRBL). When a mail server running specialized software receives mail, it checks the sender's domain name or IP address against a list of suspected spammers - if a match is found, the mail is either rejected, or marked as suspected spam. [32] Several organizations provide these lists, as people have varying views on what constitutes spam, and there is an extreme degree of difficulty involved in keeping the lists up to date. It is very easy for a spammer to change hosts, sending from new IP addresses (including using zombie networks), and registering new low cost domains.[16] [36] The basic problem is that DNSRBLs attempt to block email based on a credential that is easily changed.

The Regions network is similar to the domain system, in that it is an abstraction built at a layer higher than network addresses. An individual node may sit in multiple regions simultaneously. In addition, the cost associated with region creation is very low. This combination makes positive-match negative-actions lists ineffective. Much like the spam problem, an adversary who has been put on a negative action list merely has to switch regions. However, when dealing with spam, this can be solved with constant vigilance to discover and block the new source: in that problem domain, a solution is successful if it allows ten thousand spam emails to get through, while then blocking ten million. In the security realm, one attack getting through can be one too many. As an example, we'll postulate the existence of two regions: FOO and BAR. The creator of region FOO doesn't trust region BAR, and tries to set a policy forbidding communication with the region. An adversary can create a third region, QUUX. QUUX has a security policy that allows it to communicate with any region, both via the network and on the same node. Thus, an adversary can have one node, in regions QUUX and BAR, that bridges BAR to FOO. This attack can easily be extended to span multiple nodes and regions.

While positive-match negative-action lists do not provide adequate security in general, they may be used in conjunction with a positive-match positive-action list, or for purposes of convenience, when no security is required.

The weakness of these lists as a security tool stems from the ease of changing regions. However, if the region that is to be blocked can be trusted to always report the same region credential, it may safely be blocked. For example, my organization may wish to block connections to major-company.com for philosophical reasons. As major-company.com is not an active adversary, they have no incentive to mis-represent themselves. The negative action on the region major-company.com is implementable using current systems, and may be kept in Regions, with the understanding that it is only a convenience, not a security feature.

The use of positive-match negative-action lists in conjunction with positive-action lists is easier to understand when considered in terms of set theory. Ultimately, the concern is over positive actions - when an entity is allowed to do something. In the basic case, the set of possible regions is infinite, and an infinite number of regions can take positive action. The negative-action list is a finite list, resulting in a still infinite number of regions that can take positive action. When a positive-match positive-action list is used as a primary verifier, and subsequently checked against a positive-match negative-action list, the default action is negative, and the number of regions with positive-action capability is finite. In this scenario, a negative-action list narrows this finite set. A real world example of this construct is a company policy that says: "Allow only connections from regions Authorized by the IS department, with the exception of the finance region". This example shows the value of the positive-action lists. By making the first assertion, new regions are no longer easy to obtain, so an adversary cannot create a new region that is authorized by the IS department.

In any positive-match negative-action scenario, it is important to consider this credential creation effort and expense in determining the effectiveness of list based authorization.

2.3 Scope of RSPs

The RSP framework is ultimately shaped by the scope of coverage. Once it is determined which scenarios and technologies RSP will and will not cover, each goal must be evaluated against this standard. Individual features are included, excluded, and re-factored to assure that the final product covers exactly what it claims to cover.

Many existing technologies are concerned with low level security implementations. Specifications from SSL to IPsec provide detailed configuration specifications, and formal proofs of correctness. These systems do an excellent job solving the limited low level problems they are designed to address. Regions is a higher level architecture, and must solve a larger problem, thus the RSP should attempt to aggregate these solutions. For example, rather than precisely specifying how a TCP connection is to be encrypted, RSP leaves that low level detail to the implementation, assuming that the lower level problems have known solutions. Many of these implementations have fundamental issues that are not addressed. These are specific to the low level nature of the problem, and thus are below the level of the RSP abstraction. Many of these outstanding issues (see section 5.4) are difficult problems that are being actively worked on among specialized communities. The RSP specification must be written in a scope that encompasses these solutions, but does not rely on them.

The scope of RSPs is limited by the choice of abstraction level, which is partially set by the desire to combine related low level implementations. In precisely setting a limit of the scope, existing policy languages were analyzed.

The KeyNote trust management system (explained in detail in section 3.6) is designed to be a generalized security policy language. It is implementation independent, and extensible to a point where it can be used for non-computer based policies. In practice this results in a level of coverage that is too broad for the RSP framework. KeyNote has been used in few actual systems. The expandability of the language means that either a simple subset must be used, or a complex interpreter is needed. The desire for simplicity and transparency rule out the use of overly complex interpreters for RSP - both because the policies are difficult to read, and the translation to implementation is difficult to audit. The small subset of the KeyNote language that is used in practice directly influences the content of the RSP framework. A lighter weight framework can provide the same de-facto coverage, while being much simpler. Work with existing low level implementations show that they share a general class of abstraction concepts, so a policy specification that focuses on identifying these common abstractions can be small in size and simple in scope.

The RSP framework can be bounded further by analyzing the threats to security, and determining the best way to address those threats. Security may be compromised through several different attack vectors[7]. An adversary may try to attack a server via the network, connecting directly to a target application. In a manner further removed, an adversary can break into a target computer, and then launch an attack locally. Finally, an authorized user or program may leak information. These first two scenarios are well tailored to the Regions approach. By making careful distinction of region boundaries, one can control the information flow across the boundaries. A network or local attack that traverses regions should clearly be covered by the RSP. Furthermore, an RSP should provide mechanisms for keeping information secure while within the region, covering all possible attacks within this scenario realm.

Attacks where the user is an adversary, whether active or an unwitting accomplice, fall outside the realm of Regions and the RSP framework for several reasons: the coverage and deployment of existing policy languages, the specificity of desktop user policy, and the inherent limitations of accomplishing managing human resources using computers.

There are policy languages already in existence which control a user's software experience. Whether NIS (nee YP)[2], Microsoft roaming profiles[1][3][4], or a wide variety of kiosk applications, the software exists to control a user's access to local programs. Unlike network communication, this domain is often a mono-culture, so a one size fits all solution is workable. The user space software restrictions also overlap with a solution provided by the RSP framework. If a program has access to data that should not be user accessible, one may either limit access to the program, or never allow the program to access that data. Stated differently: one may prevent a user from writing to disk or the network either by restricting how users may operate the system interfaces, or by setting local and remote policies in an RSP and relying on the low level RSP implementation to check. Finally, these policies which closely

[7]This sense of security is intended to be information integrity - a security compromise is any in which data is read by anyone other than authorized parties, or modified by any unauthorized party. Other attacks, such as DDoS attacks, are discussed further in section 5.4.

restrict the user environment are by necessity tied very closely to the implementation, and do not abstract well.

A legitimate user that is an active adversary is fully outside the scope of technical measures. There is no way of determining user intentions once required authentication has occurred, thus no way of stopping an adversarial user. While such actions as forbidding out-of-region network connections may hinder data transfer, a truly motivated adversary could use other means, such as using a camera phone to take a screenshot of data[8]. RSPs are merely one piece of a comprehensive security policy, and can not be used to cover shortfalls in other security measures.

[8]At high security government installations, the computer networks are isolated, and personal electronics such as laptops, cell phones, and PDAs are forbidden from the premises.

Chapter 3

Related Work

The RSP framework was not conceived in a vacuum, it builds on the notion that there are many positive design features in existing technologies. RSP utilizes these legacy systems while still evolving in a way consistent with the overall goals and philosophies of the framework. In this section, related technologies are grouped together by function, in an attempt to determine the conceptual lessons that may be adopted by Regions, and in many cases implemented by these technologies.

3.1 Regions: Network Architecture Abstraction

Research into security in the Regions architecture, and how Regions will shape computer security in general, was the underlying basis for the RSP framework. Regions [31] is an architecture that groups nodes by common criteria. Rather than using network addressing and topology to force a single grouping on a node, each node is instead granted membership to one or more regions based on common functionality. In the scenario given in the introduction, Alice's computer may be a member of the ConsultCo region, the BobInc contractor region, and her home network region. These associations persist regardless of the physical network connection. This network level abstraction sets an abstraction level for the entire RSP framework. The use of this abstraction allows a security policy written at the Regions level to be simplified. Conventional network security configurations are complicated because they attempt to group access based on role, but are configured via rules specifying network addresses. By using the Regions model, an RSP author can concern himself with setting the appropriate level of access, rather than identifying which nodes are deserving of access.

3.2 General Security

Over the years, a wide range of papers and books have been produced containing general insight into computer security. These form the security knowledge base upon which systems are built, the language used to describe problems and solutions. This knowledge is augmented by works describing a narrower domain, such as a paper on basic cryptographic protocols which discuss the properties they do and don't provide.

Voydock and Kent [33] provide one of the early descriptions of how security issues arise in networks. In addition to the definition of what are now standard terms (such as what constitutes a passive attack), they outline five goals needed for a secure network, which are to prevent message contents release, prevent traffic analysis, detect message modification, detect DoS attacks, and detect invalid initializations. The paper then suggests that the best way to provide many of these is by using an end to end security model.

The RSP framework bases its abstraction level on these goals; specifically the goals of preventing content release and detecting message modification. This is an appropriate abstraction, as it expresses structure common in network security software. These structures are present in software analyzed by Voydock and Kent while forming their goals. Software authored since the paper was published was influenced by this paper and by earlier software, and thus also contains these structures.

3.3 Network

The RSP framework works from an abstract viewpoint of networks. However, policies must still be realized by an implementation, and care must to be taken to assure that this implementation is safe from attack. Doing so requires knowledge of attack methods, of which, there are three major types; passive, active, and analysis. A passive attack involves reading the data in transit. The most common type of passive attack is simple eavesdropping. An active attack involves modifying that data. Solutions [10] [17] exist that can protect data in transit from being easily read, and can detect modification. Staging these two types of attacks requires compromising either the network that the data is traversing or one of the endpoints. Under the Regions network model, a security policy which specifies that data within a region must be fully contained and that the region must maintain integrity eliminates this attack. A compromised intermediary is no longer a threat, as the policy prohibits any access to the data. The third type of attack is a traffic analysis attack, whereby an adversary gains an advantage merely by knowing that data is flowing between two nodes. The RSP framework does not explicitly address this type of attack, instead relying on low level network implementations to provide protection.

3.4 Machine

The capabilities provided by Regions force an RSP to protect all connections within the framework. A side effect of nodes residing in multiple regions is that inter-regional connections may take place entirely within a single node. The RSP framework must be able to protect these connections in a manner similar to network data. While several operating systems have basic protection and separation mechanisms, often the network stack is in a shared space and able to compromise the entire system. Several techniques have been proposed to manage this risk, all of which have different implementation implications. One such technique that was recently successfully tested in a real world setting is privilege separation [27], which is used in OpenSSH. A recent bug in this program provided a root exploit on systems that did not use privilege separation, however systems utilizing it were vulnerable only to a denial of service attack [8], which caused significantly less long term damage than full system compromise. Other projects [20] [26] have looked to limit the calls to network daemons, hoping to minimize potential exploits. At a higher level of abstraction, application sandboxing [13] [25] completely separates processes and even core network daemons. Any of these technologies could be used as a low level implementation within Regions. As in the network domain, they share common abstractions due to the problem domain.

A related mechanism embodies the RSP framework philosophy at the level of program separation. The field of Domain Type Enforcement (DTE) [5] is another way of separating program privileges. DTE is essentially a fine grained access control list (ACL), where programs and data are associated with various domains (which is somewhat analogous to a region). Domains have types associated with them, each of which is a list of allowable low level actions - reads, writes, network access, areas of the file system that they can access, etc. The very useful aspect of DTE is a separation between policy and enforcement. Policies are written at a high level for easy administrative usability and understanding, and then automatically turned into the low level rules needed to enforce the policy. In this model, a domain is the functional equivalent of a region, and the policy translation is the same that must take place from a specific RSP to implementation-specific configuration files.

3.5 Machine Authentication

The previous implementations provide a means of securing data transfer between regions and nodes, but provide less assurance as to the integrity of the remote node. Preliminary work has been conducted through the Trusted Computer Platform Alliance (TCPA) and Microsoft Next Generation Secure Computing Base (NGSCB)[1] projects to provide code attestation. Code attestation gives a provable record of the exact code stack that is running on a piece of hardware. This allows an administrative entity to have a greater assurance that the software state on a remote node is as intended [11]. The mechanisms used to verify this can fit easily within the RSP framework, and can be used for verification and in turn, stronger authentication.

[1]nee Microsoft Next Generation Trusted Computing Base (NGTCB), nee Palladium.

3.6 Formal Policies

The final influence on the RSP framework are systems which formally define policies. Early policy definition work[35] is very abstract, which allows a wide range of policy types, while requiring more customization for different applications. One of the standardized security policy languages currently in use is KeyNote[6]. KeyNote provides formal trust definitions, while leaving underlying implementations open. Unfortunately, existing security products that use KeyNote tend toward proof of concept.[15].

The KeyNote system consists of a policy, a verifier, and a requester. A KeyNote policy is very free form and flexible. It consists of credentials, actions, conditions and principals. A principal in the KeyNote system is any entity with privileges to conduct operations on the system. Credentials are used to identify principals, and to delegate actions amongst principals. These are implemented using free form strings, which can contain anything, including cryptographic keys. The conditions of a policy govern which actions may be performed, and the actions themselves are application specific tasks. These actions are listed using simple logic expressions.

As an example, the following KeyNote policy is taken from a proof of concept distributed firewall[15].

```
Authorizer: ''Policy''
Licenses: ''rsa-hex:1023abcd''
Conditions: (local_port == ''23'' && protocol == ''tcp'' &&
             remote_address > ''158.130.006.000'' &&
             remote_address < ''158.130.007.255'') -> ''true'';
    local_port == ''22'' && protocol == ''tcp''  -> ''true'';
```

The operation of a KeyNote system is relatively straightforward. A requester, which is simply a KeyNote aware program, wants to perform an action. It submits the action, the values of any parameters relevant to the proposed action, and any credentials it may possess to the verifier. The verifier takes this information, evaluates the conditions with respect to the values received from the requestor, and returns a value of true, meaning the requestor is authorized to execute the action, or false, meaning the requestor is not authorized. The requestor then must act according to the response.

There are a two important features of this system to note. Primarily, the verifier is completely unaware of the context of the system. It can be used for any KeyNote policy, and in turn, a KeyNote policy may be written for any situation. The problem domain specific logic is contained in the conditions of the policy. This is in contrast to domain specific security system, where the domain logic is intrinsic in the policy evaluator. Secondly, the requestor must be a trusted application, with knowledge of the policy nomenclature. This means that existing programs cannot be used without either being rewritten, or by using a wrapper program that contains this additional domain specific translation. Furthermore, one cannot run untrusted programs directly. Implementing KeyNote verification for an untrusted application would require verifier checks at level below the application - which is likely an operating system or similar low level program.

The complexity of KeyNote as used in that firewall implementation has caused other distributed firewall projects to take a different approach.The Smart firewalls project uses [30] its own policy language which is better tailored for firewall descriptions, and have implemented a more usable system.[19][37]

This chapter reviewed many technologies, both mature and developing, which provide host and network security. The IP Security Protocols protect network traffic. Domain Type Enforcement, privilege separation and application sandboxing provide isolation of processes on a given node. Palladium and TCPA furnish remote process verification, and KeyNote demonstrates a working formal policy language. Looking at more theoretical work, the Regions framework affords us a unique way of looking at network topology, and various papers set the language for this discussion. None of these technologies individually meets the goals of the RSP framework, but together they cover the entire problem domain. The next chapter borrows and evolves from these technologies to specify the RSP framework.

Chapter 4

Policy Design and Explanation

The Regions Security Policy (RSP) framework is a security policy language that is based on the Regions architecture. This is a formal policy that describes how data is processed whenever it enters, leaves, or transits a region. The policies themselves are hierarchal, with detail increasing with depth. This chapter first provides a quick overview of the structure of an RSP and the features. The constructs are then fully explained, giving the reasons behind the construct, the goals it meets, and the advantages over alternatives.

4.1 Design Overview

- Version: The version of the RSP specification used in the policy.
- Override: A list of any regions that may change this policy
- Credential: The data used to identify regions and control information flow between them.
  - Name: A policy-specific reference for this credential
  - System Identifier: The underlying implementation that this credential represents.

  - System Data: Data specific to the underlying implementation of this credential.

- Connection: The top level construct containing the inter- and intra- region data transfer rules.

  - Type: The nature of the data transfer.
  - Direction: The direction in which data is flowing.
  - Endpoint: The other entity involved in the communication.
  - Remote: Rules which apply to communications to a different node.
    * Authentication:
    * Encryption:
    * Transport Path:
    * Verification:
  - Storage: Rules which apply to data persistence, whether on the local node, or remotely.
    * Expiration:
    * Encryption:
    * Authentication:
  - Local: Rules which apply to data transport between processes on the local node.
    * Verification:

4.2 Version

The RSP specification is designed to be adaptable to new technologies, and new uses of current technologies. While one can safely predict that advances will occur, it is impossible to accurately foresee the direction they will take. The Regions Security Policies must be able to adopt these advances to remain relevant, and useful. In order to maintain the same level of security abstraction, it may be necessary to add fields that have meaning under the new system. Although one can disparage the speed at which network standards evolve, version increments will be rare, and less time sensitive. These major changes in security design theory are infrequent, and when they occur, take some time to be widely adopted, and standardized. Official additions to RSP can go through the standardization process at the same time as the new technology, resulting in little lag time between both standardizations.

4.3 Override

An individual node will often sit in many regions, forcing multiple RSPs to share the same space on a physical node. On some nodes, software and / or hardware mechanisms will exist to provide this separation. However, these mechanisms will not always be available nor desired. For example, a region may exist solely to provide an authentication credential to a website, not to provide connection rules. While this is an example of a common case, many regions will have further connection rules in place. These rules may be general guidelines associated with the region which can be discarded if they conflict with rules from other, co-existing regions on the node.

There four obvious ways of resolving rules conflicts between RSPs: ignoring the conflict, invoking user intervention, automatically resolving the conflict, and refusing to install policies with conflicting rules.

Traditional uses for formal policies – to maintain uniformity across an organization as indicated by a central element – eliminates the possibility of user input on arbitrary policies. In other words, a policy may exist specifically to limit privilege to a user, so the policy framework should not allow a user to override this restriction.[1] Likewise, the framework can not ignore a conflict. Ignoring problems can leave a region with a rule set not anticipated by the publisher, breaking the security, and nullifying the purpose of having a security policy. Automated resolution can suffer similar problems.

[1]RSP is a connection oriented framework, and itself does nothing to protect the implementation mechanisms from the user at an OS level. Other policy languages and enforcement mechanisms provide this protection.

It is impossible to guess the intentions of the policy without knowing the intentions of the publisher. Any advanced automatic resolution attempts to do just this, and can in turn weaken the policy in undesirable ways.

The final option for resolving RSP rules conflicts is to fully fail to install one of the regions. This solution provides sound security: an RSP provides both identifying certification and behavioral rules. If these rules can not be realized, the node must not be able to use the credential, as possession of a credential implies that an entire RSP is intact on that node.[2] Because of this behavior, the decision of which RSP to install may be left to the user, who can not use the conflict to his advantage.

While the fail safe solution assures the integrity of a region, it is very limiting, as it does not allow refinement of rule sets. This restriction limits the amount of hierarchy possible in Regions, which in turn limits the scalability. The *override* modifier (section 4.3 implements a simple automated resolution that defaults to fail-safe secure. This construct allows hierarchal rule refinement by establishing an automatic, deterministic method of resolving rule discrepancies, driven by policy publisher input. Any region listed under this section (see section 4.4.1 for naming information) can still be installed, even if the behavioral rules conflict. In these cases, the overriding region's policy takes precedence.[3] In addition, the keywords ALL and NONE (Section 4.3) may be used in the regions list, and overriding regions may be authoritative (Section 4.3)

The ALL and NONE keywords

In any list of regions, the keywords NONE and ALL may be used. As Regions operate so as to be restrictive by default (as explained in sections 2.2, 2.1), NONE is a default, implied keyword that may also be explicitly listed. It signifies that no regions should be considered under the current section. The NONE keyword supersedes all other regions that may be listed. This is a result of the RSP philosophy that the principle of negative action is most secure (as explained in section 2.1).

[2]The enforcement of this implication may be provided by various implementation mechanisms, specifically by using Palladium (section 3.5 to enforce verification rules (section 4.5.4).

[3]When multiple regions have conflicting rules, precedence is granted in order of listing.

The ALL keyword provides a complimentary function to NONE, specifying any possible region. However, as granting broad, general access is against the principles of the RSP (see section 2) the ALL keyword must be the sole modifier in a regions list. If any other region is listed, the ALL modifier is negated. This is primarily to aid human comprehension of the final policy; the all modifier could easily be overlooked in a long list of parameters.[4] This policy decision also eliminates redundancy in the RSP - any additional regions are already covered by ALL, and don't provide any more information.

The AUTHORITATIVE qualifier

The AUTHORITATIVE qualifier is another feature that stems from the desire to support hierarchy in Regions. This qualifier may be appended to any region in a list, and signifies that the region may attest to other regions, which then assume the same properties under the current policy.

The AUTHORITATIVE qualifier is not transitive. A region may list another as AUTHORITATIVE, but that region cannot further mark others as AUTHORITATIVE. This limits the scope of the delegation. Preliminary designs allowed for a greater degree, however, this was limited in the final design for the sake of simplicity - both in design, and more importantly of the resulting policies. The authoritative qualifier allows a region to formally express what may be colloquially phrased "I trust my friends' friends". A greater degree would be trusting "My friends' friends' friends". With this degree of indirection, it can become quite difficult to follow the trust chain, and understand what privileges the policy is granting. The few cases where it may be desirable to allow greater degrees of delegation are outweighed by the greater security brought by the simplicity of shorter trust chains. Capping the AUTHORITATIVE trust chain at the second degree - "Friend of a Friend" (FOAF) is the best compromise.

As an example, this construct allows a corporate IT dept to distribute one de-

[4]I anticipate the ALL keyword being used in preliminary testing of RSPs, making its inadvertent inclusion in final policies a likely scenario.

partmental credential. Individual workers, after having their personal credential authorized by the department credential, can publish policy updates under their own credential. This aids in constructing audit trails, and simplifies the policies distributed to end users.

The AUTHORITATIVE qualifier arises from the concept of delegation in an organizational hierarchy. This construct allows a more powerful entity to delegate certain tasks, without relinquishing full power. In addition, one does not have to enumerate a full list of delegates, or even the complete list at the time of policy creation.

As with every Regions construct, the AUTHORITATIVE qualifier is intended to be implementation independent; one can easily realize this construct using digital signatures.

4.4 Credential

RSP uses the notion of credentials to represent region identification, and provide related functions. Each credential contains three parts: a simple name, a system identifier, and further data for use within that system.

The name is a small, human readable string that is bound only within the specific policy. Globally, the region is identified using the implementation specific data within that implementation system. This construct simplifies the rest of the policy, as credentials are always referred to by their name, rather than by complex and lengthy data. This also eliminates the need for a global naming system, which both greatly simplifies implementation mechanisms and provides greater anonymity.

Anonymity is a key feature of the system. A node will sit in many different regions, and when initiating communication with a new node, must determine which region connection behavior rules apply to the session. Users of the system should have the option not to disclose every region to which they belong, as that is potentially valuable information to an adversary. One possible implementation involves sending all communication encrypted using a key tied to the region. If the receiving host has the corresponding key to decrypt the traffic, the connection may progress further, otherwise, no communication is possible using that encryption key.[5] Thus a receiving host at most learns what regions it shares in common with the initiating host, and no information[6] otherwise.

The credential format also affords the RSP framework great flexibility. The implementation specific data may be anything, including; public key, private key, MAC input, or plain-text identifier. Thus, one policy construct serves multiple purposes depending on the implementation and the context. This provides both the desired abstraction of implementation, and simplicity of policy.

4.4.1 Name

The name of a credential must be unique to this policy, and should be a simple name to aide comprehension of the policy. The name creates a simple, human followable reference, the reasons for which are discussed in the previous section. There must be one region named 'SELF' to allow easy self reference. The names 'All' and 'NONE' are reserved (see section 4.3) as wild cards.

4.4.2 System Identifier

The system identifier serves to specify the format and content of the Certificate Data. New formats can be added without changing the RSP specification, and implementations can share common formats. These systems are not limited to cryptographic protocols - they may be free form, and only require an associated Certificate Data specification. This allows a system such as "Plaintext" which may only provide a common use region name.

[5]A similar method is used by IPSec (the details of which, including the use of ESP and SPIs, is explained in section 5.1). A connection using ESP is encrypted from the beginning, with the header containing the ESP protocol number, and an SPI. The receiving host must then have the same key installed with a matching SPI to decrypt the packet, and instantiate communication. A lightly modified system where the receiving host ignores the SPI and tries decrypting using every key it has until a match that results in a clean checksum would accomplish a similar goal, but with increased startup overhead.

[6]Depending on the implementation used, one can likely substitute the phrase "no information, with high probability". This distinction is the result of the fact that many cryptographic systems make a guarantee of protecting information only with (extremely) high probably, not with certainty.

4.4.3 Certificate Data

The implementation specific data conforms to the Certificate Data Specification for the system identifier under which it falls. The rationale for this division is explained in previous sections. In addition, the thoughts behind the scope of specific implementations are discussed in Section 2.

4.5 Connection

Each connection, of which there may be several, expresses both the manner in which data moves from the current region to other regions, and inter-regionally across nodes. Although there are many different protocols and mechanisms that control data movement, all fall within three domains, which are encapsulated in the three sub-fields of the connection: Remote, Local, and Storage. The remote section (4.5.4) governs the handling of data as it leaves the current node including interactions that take place via a network, and the general issues that apply to the network domain. The Local section (4.5.6) is responsible for interactions on a given node. The Storage section (4.5.5) contains policy relevant to the persistence of data, whether stored in RAM, on a long term archive tape, or all manner of devices in between.

4.5.1 Type

The Connection Type field allows a policy to specify rules for specific kinds of data movement. The Regions network design is an evolution of modern systems, and must cooperate and coexist with legacy software. Because of this limitation, a general abstraction for data transfer is too coarse. For example, there is a significant difference between SSH and HTTP traffic. The type field provides the means for this distinction. As with the Credential System Identifier (4.4.2), this field is both integral to Regions and defined outside the RSP specification, allowing it to evolve more rapidly to implementation and outside software changes. However, connection types must still be general enough to remain implementation independent: the aforementioned

SSH traffic may be defined as a connection on port 22, or a connection initiated by sending a TCP packet starting with "SSH-protoversion-softwareversioncomments".

A connection is fully determined by three components: the Endpoints, the Direction, and the Type. Once these three are defined, the Remote, Local, and Storage fields provide the configuration. However, there are multiple ways of determining the connection three-tuple. Endpoint, Direction and Type can be nested, forming a hierarchal descriptor. A hierarchal system can be more efficient in certain cases. Consider a policy with the same Endpoints and Direction, but with rules that vary based on Type. Under a configuration where the tuple components are at the same level, this rule set requires two different Connections. To alleviate this, one could group Endpoints and Direction as top level, and make Type a sub-level. This configuration reduces the number of connections and consequently, overall size of the RSP for the example given. However, one can imagine a policy situation where the Type is most important, which would necessitate many top level entries have the same Type field underneath them. In this situation, a system where the Type field is top level, and Endpoints and Direction are nested is optimal.

There are thirteen unique groupings and nestings of these three components. It would require extensive usability and real world testing to further research the optimal balance, which may not even be possible. In this version of the RSP specification, Endpoint, Type, and Direction are grouped together at a single level to define a connection. This is a compromise grouping, as it will never be the most inefficient for any system.

4.5.2 Endpoint

A connection is a directional communication between regions or nodes. Every connection has a start point of the current region (named SELF) running on the current node. The termination points are defined by the Endpoint section. An endpoint may be any region (referenced by its credential name (4.4.1)), including the SELF region. This self reference allows one to set storage options and intra-regional network transport options in the same manner as inter-regional transport policies.

Any region may be additionally modified as either AUTHORITATIVE or EXCLUDE. The AUTHORITATIVE modifier was discussed in section 4.3. The EXCLUDE modifier is self descriptive; it is used to exclude a specific region credential from a larger set. The exclude modifier must be carefully used in the RSP architecture, as it presents unique security challenges, as discussed in detail in section 2.2. The exclude option only excludes a region that identifies itself as such, and consequently is best used in conjunction with an AUTHORITATIVE qualifier. For example, the Foo corporation has many regions. In this particular policy, the master signing key for the region is called FooCorp_Master. The SELF region represents the development department. Organizationally, the corporation wants every employee except for tech support personnel to have access to the development server. In this policy, the support region credential has a name of FooCorp_Support. This particular RSP would have a rule set in which one connection allows access. This connection has two endpoints; FooCorp_Master labeled as Authoritative, and FooCorp_Support labeled as EXCLUDE. A member of the support organization can not obtain a certificate authorized by FooCorp_Master (this is an offline administrative action), and thus can not access the development region.

4.5.3 Direction

The direction field is the final field used in characterizing a connection, and refers to the direction of data flow. A connection may be characterized as INBOUND, meaning the rule set applies to data moving into the current instance of this region. An OUTBOUND connection involved data moving out of the current instance, and BIDIRECTIONAL, the default, encapsulates both these INBOUND and OUTBOUND data movement. The direction field has different implications depending on the style of enforcement mechanism. For example, in a connection terminating at SELF, with a Storage modifier, an OUTBOUND connection may involve encrypting contents while writing to disk, and an INBOUND connection the decryption process. allowing for asymmetrical ciphers to be used while implementing disk storage.

4.5.4 Remote

An RSP's behavioral rules are encoded in the Remote, Storage, and Local subfields of the connection policy. These encapsulate the general classes of inter-region and intra-region connection. Each contains subfields carrying details needed for the individual implementations.

The subfields contain Credentials, referenced by local name (section 4.4.1). Each region may be further modified by AUTHORITATIVE, EXCLUDE, REQUIRED, or REQUESTED options. The AUTHORITATIVE modifier was described in Section 4.3 and the exclude in Section 4.5.2. A REQUIRED modifier signifies that the implementation of Regions is required to use the provided credential in the specific capacity listed. A REQUESTED modifier is similar to, but weaker than, a REQUIRED modifier. The REQUESTED option causes an implementation to use a credential only if possible.

The Remote section of an RSP encapsulates rules for transportation of data to other nodes, whether to the same or to different regions. The actions provided by each sub-field are implementation dependent.

Authentication

The authentication section lists credentials that may be used to authenticate a session. This may be either a login style authentication, or a Message Authentication Code (MAC – e.g., MD5).

Encryption

The encryption section lists credentials used to provide confidential transport. While these may be actual keys, these credentials could also a class of ciphers that is auto-negotiated, or other encryption technologies.

Transport Path

The transport path section lists the credentials used in a region's underlying routing system. This is the source routing equivalent for Regions.

Verification

The verification section lists credentials used to verify the software identity / region integrity of the endpoint. This is different from the authentication field, which is used to authenticate the actual data in transit.

4.5.5 Storage

The storage policy section specifies policies for data persistence, whether on local or remote stores, for all lengths of data storage.

Expiration

The expiration field lists credentials used to verify or enforce data expiration.

Encryption

See section 4.5.4

Authentication

See section 4.5.4

4.5.6 Local

The Local policy section governs the inter-regional interactions that occur on the local node.

Verification

See section 4.5.4

This chapter combined the RSP goals (simplicity, customizability, transparency, and legacy system support), the groupings of related (network systems, verification systems, process separation software, and policy languages), and the high level abstractions provided by security research to build the RSP framework. The framework itself specifies controls on data movement. These rules are grouped into three categories: data moving between two nodes, data moving between two regions on the same node, and data storage. The framework strikes a balance between contradictory forces; one such balance is that a system that is too similar to legacy software is not easily customized when new applications are invented, but anything that is too abstract requires a much more complicated implementation. Although many such trade-offs were identified and dealt with during the design phase, many shortcomings with systems don't become apparent until the system is built and an implementation is used and evaluated.

Chapter 5

RSP Implementation

As a proof of concept, a basic implementation was constructed which enforces the remote section of an RSP. This implementation is a check as to whether or not RSP may feasibly meet some of the basic design goals.

5.1 IPSec Overview

For this proof of concept implementation, I relied on the linux IP Security Protocols (IPSec) software. The IPSec family of protocols may be used both as an end-to-end solution, and in a VPN-like manner - both as a pure gateway to gateway tunnel, or a one to many tunnel. IPSec protects network communications by encrypting packets using the Encapsulating Security Protocol (ESP), detecting packet modifications (via a Message Authentication Code) using the Authentication Header (AH) standard, or by applying both ESP and AH simultaneously. These protocols are inserted into the middle of the network stack, between the TCP and IP layers. ESP and AH in turn can use many different cryptographic technologies to assure adequate functionality.

The specific IPSec implementation used in this work consists of the Linux kernel space support in the 2.6 kernel series, and user-space tools ported from the KAME project. Network traffic is affected by IPSec in the following manner. Any outgoing traffic is first queried against the Security Policy Database (SPD). The SPD is a list of source and destination host ranges, traffic direction, and policy classes. If the traffic matches an entry in the SPD, IPSec is applied according to the entry, otherwise, the packet is processed normally.[1] The SPD entry specifies whether IPSec protection is applied, the packet is processed normally, or dropped. If the entry specifies IPSec management, it specifies the mode, IPSec protocols, and the level. The mode may be either transport or tunnel, which corresponds to the end-to-end versus VPN style operation. The IPSec protocols are AH and / or ESP. The level specifies how strictly the SPD must be followed; as an example, one may set an SPD to use ESP and AH if they're available on the remote host, but default to normal transport otherwise.

Network traffic that is marked for IPSec processing by the SPD is then checked against the Security Association Database (SAD). The SAD entries are much more detailed, specifying individual hosts, not host ranges. The SAD entries also contain the specific parameters for AH and ESP protocols - the encryption or Message Authentication Code (MAC) algorithms to use, the keys for these algorithms, and a Security Parameter Index (SPI). The SPI is a parameter in the ESP and AH headers that identifies the SAD entry associated with a packet.

Traffic marked by the SPD as needing IPSec, but which lacks a corresponding SAD entry triggers a daemon (called racoon in this implementation) which performs Internet Key Exchange (IKE) via Internet Security Association and Key Management Protocol (ISAKMP). The key exchange daemons coordinate the two ends of a communication, and if communication is allowed per racoon configuration rules (which are separate from the SPD/SAD), dynamically create SAD entries that use secure, one time keys[2]. After matching traffic to a SAD, the IPSec protocols are applied as specified, and the traffic is sent as normal to the IP layer.

The processing of incoming packets is slightly different, in the interests of optimization. Any incoming packet that uses ESP or AH also contains an SPI. The SPI is an index into the SAD, where the decryption or message authentication keys may

[1]This behavior is contrary to the RSP philosophies of default denial and positive-match, positive-action. This is a linux IPSec implementation decision made in the interests of interoperability with non-IPSec systems. In a network where every communicating host was running IPSec and using Regions, a final 'catch-all' entry that drops all traffic not IPSec protected should be added.

[2]Static, manually created SAD entries must use pre-shared secret keys, which are more vulnerable to compromise.

Table 5.1: Acronyms and their meanings used in this chapter

AH	Authentication Header (RFC 2402)
CBC	Cipher Block Chaining
DES	Data Encryption Standard
ESP	Encapsulating Security Protocol (RFC 2406)
HMAC	Keyed-Hashing Message Authentication Code
IKE	Internet Key Exchange (RFC 2409)
IPSec	Security Architecture for IP (RFC 2401)
ISAKMP	Internet Security Association and Key Management Protocol (RFC 2408)
MD5	Message Digest #5 (RFC 1321)
SAD	Security Association Database
SHA1	US Secure Hash Algorithm 1 (RFC 3174)
SPD	Security Policy Database
SPI	Security Parameter Index

be found. Use of the SPI bypasses a more time-intensive lookup in the SPD and the SAD for all incoming packets.

5.2 Implementation

The actual implementation uses perl to create and manage the IPSec configurations. Configuration creation is done by a single script. This script reads the node's RSPs, figures out the dependencies, and writes the appropriate SAD entries and racoon configuration file. Then credentials are parsed, converted into a format understood by racoon, the SAD entries are updated in the kernel, and racoon is signaled to reread its configuration file. The final component is a script that updates the SADs to have the correct local IP address when the node changes address.[3]

5.3 Results

This implementation is only a proof of concept, and is missing some functionality (see section 5.4.3), however it allows analysis of the underlying framework. The test policy

[3]These changes of address are more likely to be common in a future where wireless access is ubiquitous, and a consumer views the network not from an address, but from a Region.

used is shown in appendix A, and the resulting SPD entries and racoon configuration files in appendix B.1. The goal of simplicity is clearly met by RSP. The policy and configurations provide the exact same information, but the RSP presents it in a more straight forward manner. The guiding philosophy of positive-match, positive-action, and default passive negative-action are prevalent in the SAD. The SAD requires all connections to use IPSec. Remote hosts that don't have a certificate in the racoon configuration file cannot connect to this host. The goal of legacy system support is clearly met, as this implementation relies on IPSec, which is an existing security implementation. The verifiability of this implementation is demonstrated by appendix C, which shows the program used to translate the RSP into IPSec configuration files. This is a small program, easily auditable by anyone with some basic programming experience. Because of this simplicity, it is easily customizable for different IPSec behavior.

5.4 Outstanding Issues

This particular implementation also demonstrates the areas that Regions doesn't address. Section 2.3 discusses the reasons behind this decision, however, these choices are much more apparent when looking at an actual implementation.

5.4.1 Application level authorization

The Regions Security Policy framework is primarily a network oriented construct, concerned with handling data movement. A consequence of this focus is that RSP attempts to separate itself from applications. Although implementations of RSPs may serve to separate two applications on the same machine, each application sees the interaction as a simple data transfer. Thus, application needs that don't directly address this data movement are not supported by the RSP framework. Application authorization is one such feature. Although many applications may wish to use certificates or other cryptographic features for authentication, Regions does not provide this. RSP credentials provide basic assurances of membership in a region. These assurances do not extend to a specific user identity, nor should they. Allowing applications to piggy-back on network authentication would violate the design principle of simplicity. Adding interfaces for applications would make an implementation more complicated. In addition, this removes an additional layer of security that the RSP framework adds, and makes Regions credential management much more difficult, as it must coordinate between many applications. Credential management is currently a difficult problem, without the additional requirements applications impose.

This separation also allows Regions to take a less aggressive policy toward region membership, which benefits widespread deployment. A Regions credential may be a communal piece of identification - effectively a widely shared secret key. If the issuer of an RSP wishes to evict an entity, they may eliminate access at the application level, and be able to gradually roll out updated credentials and corresponding policies to legitimate nodes.

5.4.2 Certificate Revocation Lists (CRLs)

Regions credentials are a technology independent identifier. However, they will often be implemented using certificates. This use of certificates will raise some concerns, as there are several issues with certificate management in distributed systems, chiefly the mechanisms used to enforce revocation of certificates. These are not relevant within the RSP framework for several reasons. Primarily, certificates are not essential for Regions. Because of the abstraction, other technology can be substituted. A corollary of this is that work done on certificate revocation, which is an active topic of research[23][28][22], is immediately applicable to any Regions implementation which uses certificates. Finally, many of these issues arise when trying to issue certificates to authenticate many individual users. Regions differs in approach. The difference between application authorization and Regions membership was explained in Section 5.4.1.

5.4.3 Implementation Issues

The separation of the RSP framework and individual implementations is critical when one evaluates these implementations. No one technology is perfect, and may contain features that are undesirable for the end user or region administrator. Thus, an RSP solution must take two parts, the actual policy, and the technology used to implement the policy.

This particular implementation uses IPSec, which provides largely transparent protection of network traffic. However, it is not fully transparent. When initiating a connection that doesn't have an SAD entry, the connection will fail, with an error that the requested resource is temporarily unavailable. An application that receives this error must be smart enough to try again. It occurs because in order to fill in the SAD entry, racoon must be called, to perform IKE and set up an appropriate entry. After this takes place, the application can then re-initiate the connection, and have it be protected via IPSec. Unfortunately, this behavior may cause failure in legacy applications being used within this framework. This failure is not the fault of RSP, and could be averted with a different implementation of IPSec, or by using a different transport security layer altogether.

Chapter 6

Final Remarks

This paper examined the thoughts behind, and the resulting structure of, a new way of looking at network security. By beginning with a basis in the Regions system, in Chapter 2 we first formulate design goals of simplicity, customizability, transparency, and legacy system support. Next we specify overall design philosophies, which include defining the abstraction level, and the concept of positive-match, positive-action lists that default to passive denial. Having explained these goals and philosophies, we use them to evaluate contemporary security systems in Chapter 3. Many of these existing systems are found to be too narrow in focus to meet the goals of RSP, but may grouped into several categories. Network security technologies such as IPSec ensure protected communications between hosts. Palladium and TCPA provide remote attestation of the software state of a given node. Domain type enforcement, sandboxing, and others separate processes on a particular node. At the opposite end of the spectrum of detail, the KeyNote policy language proves to be too general for use. Taking inspiration from these systems, and guidance from the goals and philosophies, Chapter 4 outlines the design of the Regions Security Policy (RSP) framework.

RSP is a hierarchal policy language which specifies rules governing the transfer of data both within and between regions. It applies to data movement across two nodes, between two separate regions on the same node, and data persistence, whether on an individual node or as networked storage.

After completing design work on RSP, Chapter 5 discusses a proof of concept implementation. This implementation contains a policy governing the network behavior of regions, which is enforced by the Linux IPSec software. This implementation shows that RSP meets its goals, however, it also highlights the deficiencies of this implementation, and of the RSP framework in general. Section 5.4 discusses these shortcomings, and explains why they fall outside of the scope of the problem that RSP attempts to solve.

6.1 Future Work

There is still much work that may be accomplished with RSP and its related implementations. This thesis contains proof of concept code to enforce remote policy. This code is not robust nor complete enough to be used in any production environment. The logical next step would be expanding this code to include support for the override feature, and complete checks for conflicting policies. Beyond these obvious features lies a difficult path. Ultimately, regions that must share a node will have contradictory policies. Demand for secure computing, but also for wide usability will drive a demand for a solution.

This thesis investigates some technologies that promise to provide separation on a single machine. These technologies need time to mature, at which time, they may be better evaluated as a solution. Future work on Regions Security Policies will likely focus on this problem domain - implementing and evaluating when the technology becomes better suited to the problem. I suspect that this intra-node separation will ultimately be provided by a different mechanism not explicitly listed in this thesis - perhaps virtual machines which are in turn separated by domain-type-enforcement.

The implementation of remote and local enforcement issues will clearly change given more work in the field. However, the changes I've described can be accommodated by the RSP framework described in this thesis. RSP will not be able to accommodate every new security technology, but is designed to evolve with changes in technology, so that it may remain useful and relevant as a security tool.

Bibliography

[1] Designing a managed environment: Teaching users about roaming profiles. http://www.microsoft.com/resources/documentation/WindowsServ/2003/all/d%eployguide/en-us/Default.asp?url=/resources/documentation/WindowsServ/2003/all%/deployguide/en-us/dmebc_dsm_tqpg.asp.

[2] Solaris naming services. http://docs.sun.com/db/doc/816-4856/6mb1q0bgf?a=view#a00intro-97078.

[3] Web kiosk commander. http://www.rockmedia.com/webkiosk.html.

[4] Enhance security in a public access environment. http://www.microsoft.com/technet/prodtechnol/office/office2000/maintain%/security/ensecrty.mspx, July 2001.

[5] Lee Badger, Daniel F. Sterne, David L. Sherman, and Kenneth M. Walker. A domain and type enforcement UNIX prototype. *Computing Systems*, 9(1):47–83, 1996.

[6] M. Blaze, J. Feigenbaum, J. Ioannidis, and A. Keromytis. Rfc 2704: The keynote trust-management system version 2. http://www.ietf.org/rfc/rfc2704.txt, September 1999.

[7] Your web site might be playing host to a hacker. http://www.businessweek.com/smallbiz/content/mar2000/sb20000306_426.htm%.

[8] Cert advisory ca-2003-24 buffer management vulnerability in openssh. http://www.cert.org/advisories/CA-2003-24.html.

[9] William Cheswick, Steven Bellovin, and Aviel Rubin. *Firewalls and Internet Security (Second Edition): Repelling the Wily Hacker.* 2002.

[10] T. Dierks and C. Allen. Ref 2246: The transportation layer security (tls) protocol version 1.0. `http://www.ietf.org/rfc/rfc2246.txt`, January 1999.

[11] Tal Garfinkel, Mendel Rosenblum, and Dan Boneh. Flexible os support and applications for trusted computing. In *Proceedings of the 9th Workshop on Hot Topics in Operating Systems (HotOS-VIII)*, May 2003.

[12] Jim Gillogly. Notes on non-cryptographic ways of losing information a talk by robert morris. `http://www.ieee-security.org/Cipher/PastIssues/1995/issue9509/issue9509%.txt`.

[13] Ian Goldberg, David Wagner, Randi Thomas, and Eric A. Brewer. A secure environment for untrusted helper applications. 1996. `http://citeseer.nj.nec.com/goldberg96secure.html`.

[14] Jim Hopkins. Tech investor: Other nations zip by usa in high-speed net race. `http://www.usatoday.com/tech/techinvestor/2004-01-19-broadband_x.htm`.

[15] Sotiris Ioannidis, Angelos D. Keromytis, Steven M. Bellovin, and Jonathan M. Smith. Implementing a distributed firewall. In *ACM Conference on Computer and Communications Security*, pages 190–199, 2000.

[16] Philip Jacob. The spam problem: Moving beyond rbls. `http://theory.whirlycott.com/~phil/antispam/rbl-bad/rbl-bad.html`.

[17] S. Kent and R. Atkinson. RFC 2401: Security architecture for the Internet Protocol. `http://www.ietf.org/rfc/rfc2401.txt`, November 1998. Obsoletes RFC1825. Status: PROPOSED STANDARD.

[18] Douglas Knowles, Frederic Perriot, and Peter Szor. Symantec security response - w32.blaster.worm. `http://securityresponse.symantec.com/avcenter/venc/data/w32.blaster.wor%m.html`.

[19] Alexander V. Konstantinou, Yechiam Yemini, Sandeep Bhatt, and S. Rajagopalan. Managing security in dynamic networks. In *13th USENIX Systems Administration Conference (LISA'99)*, pages 109–122, 1999.

[20] William LeFebvre. Restricting network access to system daemons under sunOS, 1992. `http://citeseer.nj.nec.com/lefebvre92restricting.html`.

[21] Corey Merchant and Joe Stewart. Detecting and containing irc-controlled trojans: When firewalls, av, and ids are not enough. `http://www.megasecurity.org/Trojaninfo/IRC\%20controled\%20Trojans.html%`.

[22] S. Micali. Efficient certificate revocation. Technical Report MIT/LCS/TM-542b, 1996.

[23] Moni Naor and Kobbi Nissim. Certificate revocation and certificate update. In *Proceedings 7th USENIX Security Symposium (San Antonio, Texas)*, Jan 1998.

[24] netfilter/iptables project homepage. `http://www.netfilter.org/`.

[25] David S. Peterson, Matt Bishop, and Raju Pandey. A flexible containment mechanism for executing untrusted code. *Proceedings of the 11th USENIX Security Symposium*, pages 207–225, August 2002.

[26] Niels Provos. Improving host security with system call policies. *12th USENIX Security Symposium*, August 2003.

[27] Niels Provos, Markus Friedl, and Peter Honeyman. Preventing privilege escalation. *12th USENIX Security Symposium*, August 2003.

[28] Ronald L. Rivest. Can we eliminate certificate revocation lists?, February 1998. Springer Lecture Notes in Computer Science No. 1465.

[29] Antoine De Saint-Exupery. *Wind, Sand, and Stars.* Harcourt, 1992.

[30] Smart firewalls: Automatic management of security policy in dynamic networks. `http://govt.argreenhouse.com/SmartFirewall/`.

[31] Karen R. Sollins. Regions: a new architectural capability for networking. `http://krs.lcs.mit.edu/regions/docs/regions-wh.pdf`, August 2001.

[32] Spews.org - the internet's spam prevention early warning system. [faq]. `http://www.spews.org/faq.html`.

[33] V. L. Voydock and S. T. Kent. Security mechanisms in high-level network protocols. *ACM Computing Surveys*, 15(2):135–171, June 1983.

[34] Alma Whitten and J. D. Tygar. Why Johnny can't encrypt: A usability evaluation of PGP 5.0. In *8th USENIX Security Symposium*, 1999.

[35] R. Wies. Policies in network and systems management - formal definition and architecture. 1994. `http://citeseer.nj.nec.com/wies94policies.html`.

[36] Meng Weng Wong. [spf-discuss] how blacklisting will work in the future. `http://archives.listbox.com/spf-discuss@v2.listbox.com/200401/0239.html%`.

[37] Yechiam Yemini, Alexander V. Konstantinou, and Danilo Florissi. Nestor: An architecture for network self management and organization. `http://www1.cs.columbia.edu/dcc/nestor/`.

Appendix A

XML RSP

This appendix contains the XML formatted RSP used in the proof-of-concept implementation. This policy specifies that connections to the same region are allowed only via http, and that these connections may be initiated or received by either node.

```
<policy>
<version>0.99b</version>

<override EXCLUDE="false" AUTHORITATIVE="false">REMOTE</override>
<override EXCLUDE="true" AUTHORITATIVE="false">FooCorp_Support</override>
<override EXCLUDE="false" AUTHORITATIVE="true">FooCorp_Master</override>

<credential>
<name>SELF</name>
<system_identifier>x509</system_identifier>
<system_information>
<public_key>
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 1 (0x1)
        Signature Algorithm: md5WithRSAEncryption
        Issuer: CN=REGION_test_AUTHORITATIVE
        Validity
            Not Before: May  3 22:39:26 2004 GMT
        Not After : May  3 22:39:26 2005 GMT
    Subject: CN=REGION_test
    Subject Public Key Info:
        Public Key Algorithm: rsaEncryption
        RSA Public Key: (1024 bit)
            Modulus (1024 bit):
                00:c6:2a:cf:a1:99:72:cd:bc:0f:05:68:d2:a1:93:
                6d:86:01:52:d6:74:03:05:23:f0:d0:10:32:36:ce:
                b7:66:e2:07:d6:63:df:07:a0:f7:bb:89:d5:10:50:
                35:92:f5:9d:1b:f9:e1:98:67:4f:90:4b:9a:6e:39:
                64:67:bc:57:cb:ea:cb:2d:76:0a:29:9c:1e:a0:e4:
                25:18:95:2f:ca:d3:f7:23:08:d5:80:7d:75:cb:fd:
                65:fb:7d:b4:2f:79:f8:96:0f:45:ff:c5:d7:09:97:
                33:d7:79:5c:99:13:ca:f9:c3:25:55:b1:a7:3d:f8:
                b2:03:6f:1d:70:36:f7:11:9b
            Exponent: 65537 (0x10001)
    X509v3 extensions:
        X509v3 Basic Constraints:
            CA:FALSE
        Netscape Comment:
            OpenSSL Generated Certificate
        X509v3 Subject Key Identifier:
            71:53:46:4C:0B:3C:48:86:7D:89:62:34:3A:96:FF:57:0E:E4:5E:EC
        X509v3 Authority Key Identifier:
            keyid:20:90:77:2D:28:FB:3D:98:22:49:AC:2D:EC:76:88:0D:BB:28:AC:41
            DirName:/CN=REGION_test_AUTHORITATIVE
            serial:00

Signature Algorithm: md5WithRSAEncryption
    9d:6f:e4:2a:65:0d:7f:bd:90:a9:5d:07:80:2b:fd:26:d1:ef:
    a4:bb:48:e1:e3:6c:d7:e2:03:5d:59:cc:e8:38:ed:14:75:2a:
    74:dc:a6:dc:bb:b9:4d:bf:fb:85:37:68:65:14:6f:29:6e:cc:
    ba:2c:3a:a4:e2:9f:79:94:3b:b9:0e:86:19:5d:6e:37:4c:00:
    41:85:64:e9:2a:ec:4e:4d:9a:ca:a1:52:98:23:d6:a9:95:7b:
    f8:42:9d:7c:04:f1:9a:f9:df:23:f0:c5:ef:b4:14:8a:10:b8:
    e1:5c:45:62:bf:55:c0:5c:6f:d0:02:3b:fa:dd:06:8b:f8:5d:
```

08:34
------BEGIN CERTIFICATE------
MIICWjCCAcOgAwIBAgIBATANBgkqhkiG9w0BAQQFADAkMSIwIAYDVQQDFBlSRUdJ
T05fdGVzdF9BVVRIT1JJVEFUSVZFMB4XDTA0MDUwMzIyMzkyNloXDTA1MDUwMzIy
MzkyNlowFjEUMBIGA1UEAxQLUkVHSU9OX3Rlc3QwgZ8wDQYJKoZIhvcNAQEBBQAD
gY0AMIGJAoGBAMYqz6GZcs28DwVo0qGTbYYBUtZ0AwUj8NAQMjbOt2biB9Zj3weg
97uJ1RBQNZL1nRv54ZhnT5BLmm45ZGe8V8vqyy12CimcHqDkJRiVL8rT9yMI1YB9
dcv9Zft9tC95+JYPRf/F1wmXM9d5XJkTyvnDJVWxpz34sgNvHXA29xGbAgMBAAGj
gakwgaYwCQYDVR0TBAIwADAsBglghkgBhvhCAQ0EHxYdT3BlblNTTCBHZW5lcmF0
ZWQgQ2VydGlmaWNhdGUwHQYDVR0OBBYEFHFTRkwLPEiGfYliNDqW/1cO5F7sMEwG
A1UdIwRFMEOAFCCQdy0o+z2YIkmsLex2iA27KKxBoSikJjAkMSIwIAYDVQQDFBlS
RUdJT05fdGVzdF9BVVRIT1JJVEFUSVZFggEAMA0GCSqGSIb3DQEBBAUAA4GBAJ1v
5CpℓDX+9kKℓdB4Ar/SbR76S7SOHjbNfiA11ZzOg47RR1KnTcpty7uU2/+4U3aGUU
byluzLosOqTin3mUO7kOhhldbjdMAEGFZOkq7E5NmsqhUpgj1qmVe/hCnXwE8Zr5
3yPwxe+0FloQuOFcRWK/VcBcb9ACO/rdBov4XQg0
------END CERTIFICATE------
</public_key>
<private_key>
------BEGIN RSA PRIVATE KEY------
MIICXgIBAAKBgQDGKs+hmXLNvA8FaNKhk22GAVLWdAMFI/DQEDI2zrdm4gfWY98H
oPe7idUQUDWS9Z0b+eGYZ0+QS5puOWRnvFfL6sstdgopnB6g5CUYlS/K0/cjCNWA
fXXL/WX7fbQvefiWD0X/xdcJlzPXeVyZE8r5wyVVsac9+LIDbx1wNvcRmwIDAQAB
AoGAC2JoPTtoigM0xbXI6/lhQGKRFLrjdYckDX/wso9bn/B6TMm+BV0s/jwj3mUN
Pt0XYoUPfcbpnjuJqq1nZEJAtN9VFA2Se0DvcZIjijJv2503Mv6xC5eejfZRv9+8
NmmCaGPCY6eVQFIOty7rA0V7un/JWRctTr6fGW73RqGhxfkCQQD1tXsePIHvpr5l
bARehMHK8V3PjNZxe3T8hDN75QHTPmlm2G8IGv3Y/265NczaIURCXGPu/11vKt1I
2Wp7HNVVAkEAzneVMnAbKxhRWtByQHSi1ZAxeo5W6X6eM9RfiMXaYG7X23BknU63
i2kFTJ2A6+CDj1igIYyd1tk3AfnvgXxLLwJBAMOGv4q4K85BqpGa+38btfuBR126
fYug6t9ndHDLNECeEdI9uV2B3S+pVLseDP4oKuGEFCJEJF4qhh1KpmmB8GECQQCR
E1HQqRlOBL5Vk5ZUWCB68+DwfsfvNbswLBAc6PlzPS+Lz8PDDSbHXLoOhbWrCI0o
0ℓtid44JA4Q1bUuvroidAkEA06k0z74nDsz9GXXYybu/UyB5cDHQ0BKo1Luiv0Ba
7hc6LZOye8mPGVBxws6Uq7XMx1eXgW9PIJLQwJvCiPGk7Q==
------END RSA PRIVATE KEY------
</private_key>
</system_information>
</credential>

```
<connection>
<type>regions</type>
<type>http</type>
<direction>BIDIRECTIONAL</direction>
<endpoint EXCLUDE="false" AUTHORITATIVE="false">SELF</endpoint>
<remote>
<encryption>SELF</encryption>
<authentication>SELF</authentication>
</remote>
<storage></storage>
<local></local>
</connection>
<connection>
<direction>INBOUND</direction>
</connection>
</policy>
```

Appendix B

Implemenation Configuration Files

B.1 IPSec Configuration

The IPSec configuration file generated from the RSP XML. The verbosity comes from the configuration language in this implementation, which only allows a connection to list a single port and direction, requiring the repetition of similar data.

```
spdadd 0.0.0.0/0 18.26.0.15[1025] any -P in ipsec esp/transport//require;
spdadd 18.26.0.15[1025] 0.0.0.0/0 any -P out ipsec esp/transport//require;
spdadd 0.0.0.0/0[1025] 18.26.0.15 any -P in ipsec esp/transport//require;
spdadd 18.26.0.15 0.0.0.0/0[1025] any -P out ipsec esp/transport//require;
spdadd 0.0.0.0/0 18.26.0.15[80] any -P in ipsec esp/transport//require;
spdadd 18.26.0.15[80] 0.0.0.0/0 any -P out ipsec esp/transport//require;
spdadd 0.0.0.0/0[80] 18.26.0.15 any -P in ipsec esp/transport//require;
spdadd 18.26.0.15 0.0.0.0/0[80] any -P out ipsec esp/transport//require;
```

B.2 Racoon Configuration

The configuration file for the racoon IKE daemon. Note that this file also requires the indentifying certificates, which are fully contained in the RSP XML, to be saved into seperate files, increasing the complexity of data management.

```
path certificate "/etc/rsp/keys/";
remote anonymous {
  exchange_mode aggressive, main;
  generate_policy on;
  certificate_type x509 "SELF-x509-public.pem" "SELF-x509-private.pem" ;
  verify_cert off;
  my_identifier asn1dn;
  peers_identifier asn1dn;
  proposal {
       encryption_algorithm 3des;
        hash_algorithm md5;
        authentication_method rsasig;
        dh_group modp1024;
  }
}
sainfo anonymous {
        pfs_group modp1024;
        encryption_algorithm 3des;
        authentication_algorithm hmac_md5;
        compression_algorithm deflate;

  }
```

Appendix C

PERL Implementation

This appendix contains the perl code used to translate the RSP into configuration files for IPSec and racoon. Note that as per Section 5.3, this is not a complete implemention, but still shows that an RSP implemenation is straightforward. The many lines of repeated print statements stems from generating configuration files thathave many lines with small differences, and is not a function of RSP.

```
#!/usr/local/bin/perl -w

#this MUST be run as root, otherwise, racoon will complain
#about file permissions.

require XML::Simple;
use Data::Dumper;

sub GetIP {
    open (INPUT, "/sbin/ifconfig eth0 |");
    join('', (<INPUT>)) =~ m/addr:(\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})/;
    close (INPUT);
    return($1);
}

my $setkey = "/usr/sbin/setkey";
my $racoon = "/usr/sbin/racoon";
my $ipsec_conf = "/etc/rsp/ipsec.conf";
my $racoon_conf = "/etc/rsp/racoon.conf";
my $keydir = "/etc/rsp/keys/";
my $policy_file = "policy.xml";

my $xs = new XML::Simple();

my $policy = $xs->XMLin("$policy_file",
      forcearray=>[qw(override credential connection)],
      keyattr=>{override=>"content", credential=>"name" });

my $ip = GetIP();

open (IPSEC, ">$ipsec_conf") or die ("couldn't open $ipsec_conf");
open (RACOON, ">$racoon_conf") or die ("couldn't open $racoon_conf");

# add excludes to spd entries.

foreach (@{$policy->{'connection'}}){
    my $connection = $_;
    my $port;

    foreach (@{$connection->{'type'}}){
  $port = $_;

  #Placeholder - do this via getservbyname
  #check port, translate to number.
  if ($port =~ m/REGIONS/i) { $port = 1025;}
  if ($port =~ m/SSH/i) { $port = 22;}
  if ($port =~ m/http/i) { $port = 80;}
  if ($port =~ m/ALL/i){ undef $port;}

  if($port){
      if ($connection->{'direction'} =~ m/INBOUND/i){
    print (IPSEC "spdadd 0.0.0.0\/0 $ip\[$port]",
          "any -P in ipsec esp\/transport\/\/require;\n") ;
    print (IPSEC "spdadd $ip\[$port] 0.0.0.0\/0",
          " any -P out ipsec esp\/transport\/\/require;\n") ;
      }elsif ($connection->{'direction'} =~ m/OUTBOUND/i){
    print (IPSEC "spdadd 0.0.0.0\/0[$port] $ip any",
          " -P in ipsec esp\/transport\/\/require;\n") ;
    print (IPSEC "spdadd $ip 0.0.0.0\/0[$port] any",
          " -P out ipsec esp\/transport\/\/require;\n") ;
      }elsif ($connection->{'direction'} =~ m/BIDIRECTIONAL/i){
    print (IPSEC "spdadd 0.0.0.0\/0 $ip\[$port] any -P in",
          " ipsec esp\/transport\/\/require;\n") ;
    print (IPSEC "spdadd $ip\[$port] 0.0.0.0\/0 any -P out",
          " ipsec esp\/transport\/\/require;\n") ;
    print (IPSEC "spdadd 0.0.0.0\/0[$port] $ip any -P in",
          " ipsec esp\/transport\/\/require;\n") ;
    print (IPSEC "spdadd $ip 0.0.0.0\/0[$port] any -P out",
          " ipsec esp\/transport\/\/require;\n") ;
      }
  }else{
      print (IPSEC "spdadd 0.0.0.0\/0 $ip any -P in ipsec",
       " esp\/transport\/\/require;\n") ;
      print (IPSEC "spdadd $ip 0.0.0.0\/0 any -P out ipsec",
       " esp\/transport\/\/require;\n") ;

  }
    }
}



#FIXME: I don't do the ports in the racoon file yet, add this.
#also, fix the order / deal with overrides
print (RACOON "path certificate \"$keydir\";\n");

foreach (keys %{$policy->{'credential'}}){
    my $key = $_;
    my $cert = $policy->{'credential'}->{$key};
    die ("only x.509 supported")
  unless ($cert->{'system_identifier'} =~ m/x.?509/i);

    my ($private_key, $pub_key);

    $private_key = $key . "-x509-private.pem";
    #BUG only works with one policy file...
    $pub_key = $key . "-x509-public.pem";

    unlink ("$keydir$pub_key") if (-e "$keydir$pub_key");
    unlink ("$keydir$private_key") if (-e "$keydir$private_key");

    open(PUBLIC_KEY, ">$keydir$pub_key")
  or die ("couldn't open $keydir$pub_key");
    open(PRIVATE_KEY, ">$keydir$private_key") or
  die ("couldn't open $keydir$private_key");

    print(PUBLIC_KEY $cert->{'system_information'}->{'public_key'});
    print(PRIVATE_KEY $cert->{'system_information'}->{'private_key'});

    close(PUBLIC_KEY);
    close(PRIVATE_KEY);

    chmod 0400, "$keydir$pub_key";
    chmod 0400, "$keydir$private_key";

print (RACOON "remote anonymous {\n",
        "  exchange_mode aggressive, main;\n",
        "  generate_policy on;\n",
        "  certificate_type x509 \"$pub_key\" \"$private_key\" ;\n",
  "  verify_cert off;\n",
        "  my_identifier asn1dn;\n",
        "  peers_identifier asn1dn;\n",
        "  proposal {\n",
        "       encryption_algorithm 3des;\n",
        "        hash_algorithm md5;\n",
        "       authentication_method rsasig;\n",
        "       dh_group modp1024;\n",
        "  }\n",
       "} \n",
       "sainfo anonymous {\n",
       "       pfs_group modp1024;\n",
       "       encryption_algorithm 3des;\n",
       "       authentication_algorithm hmac_md5;\n",
       "       compression_algorithm deflate;\n",
       "  }\n");
}

close (IPSEC);
close (RACOON);

system ("$setkey -f $ipsec_conf") ;# or die ("couldn't run setkey $!");
system ("$racoon -f $racoon_conf") or die ("couldn't run racoon $!");
```