networkGEOGRAPHICS
guides to your network

Consistent Security Policy in a Changing World

In an ideal world, an organization's security policy will be directly refined into the security implementation and the policy and implementation will always be consistent. Unfortunately, we do not live in an ideal world.

Frequently, there are large disconnects between an organization's security goals and what security is actually enforced. This disconnect means that an improper security stance may go undetected until the next audit pass. In this paper, we outline policy validation techniques that can be used to tighten the binding between network security policy and network security implementation.

The Current State of Policy Driven Security

The design and development of a security architecture and implementation should be driven by the organization's security policy. These security policies are written in a natural language (e.g. English) and come from on high driven by external regulations, internal risk analysis, and sector norms. The natural language policies drive the next level of system design and ultimately result in a particular architecture of security mechanism placement and configuration.

The current process for security policy development and enforcement is much like that of the waterfall model in software engineering. High level policy flows into more detailed policy and standards which finally flow into specific enforcement systems deployment and configuration as shown in Figure 1.

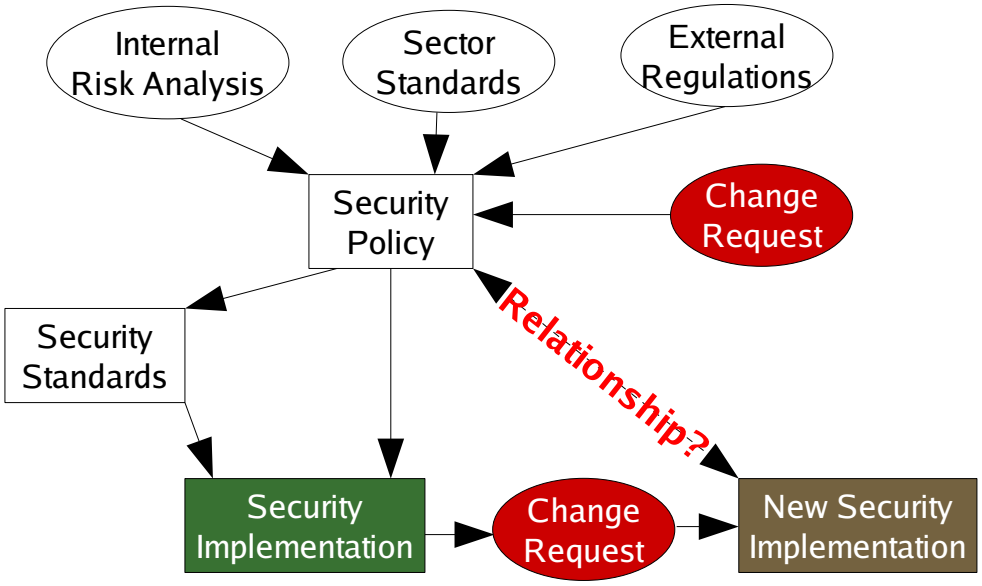

Figure 1: *The standard security architecture development flow which is a waterfall from policy to implementation. To follow the waterfall process, changes to the implementation should feed into the top of the water fall as shown by the top change oval. In reality, many changes will be made directly on the current implementation with policy validation being an ad hoc process.*

The waterfall model is a natural first generation process for a tool and knowledge weak environment. Its single flow of control can be tracked by humans and requires minimal dependence on support tools. However, administrators frequently have to make changes to security mechanisms on a day-to-day basis which are generally driven by changes in

© Network Geographics 2007

functional requirements, e.g., adding communication for new partners, or the addition of new services. Ideally, these changes should feed into the waterfall process as shown by the top red oval in Figure 1, but time change requirement usually prohibit a full policy review for every security implementation change.

In the current waterfall process, the administrator will try to keep the existing security policy in mind while making changes to the enforcing mechanisms as shown in the in the bottom red oval in Figure 1. However, without additional support, we cannot make any strong statements about the relationship of the new implementation to the security policy. Periodically, auditors will be brought in to verify that the security implementation still matches the guiding security policy. However, in a sensitive environment, potentially being out of sync with the security policy for days or weeks provides a large window of opportunity. The legitimate people working in the organization's infrastructure will be quick to note security errors that cause too little access, but only the uninvited will notice security errors that cause too much access, and these uninvited entities will not be likely to file a bug report.

How Can We Do Better?

Just as software engineering processes evolved into a series of iterative approaches (e.g., spiral model, extreme programming), so must security practice evolve to cope with the increasing system complexity and change velocity. We need a way to automatically ensure that day-to-day changes to the security implementation meet the guiding security policy ideally before the implementation changes are made. Through user education, system administrators can be taught to keep the security policy in mind while making changes, but with a large and complex security architecture, it can be quite easy to overlook some unintended consequences to the security policy. To achieve the needed speed and accuracy to ensure that the security implementation is always consistent with the security policy, we must introduce automated tools to perform the validation and verification for us.

If the requirements of the natural language security policy can be distilled into a set of formal security policies, then one can create tools to automatically enforce policy constraints against implementation changes as shown in Figure 2. With explicit ties from implementation changes back to policy, the security architecture can adapt more quickly, more accurately, and with a higher degree of assurance. This paper focuses on emerging technologies that are enabling this transition to a more iterative approach to security architecture development and maintenance. In particular we present techniques for validating network security implementations against security policy.

© Network Geographics 2007

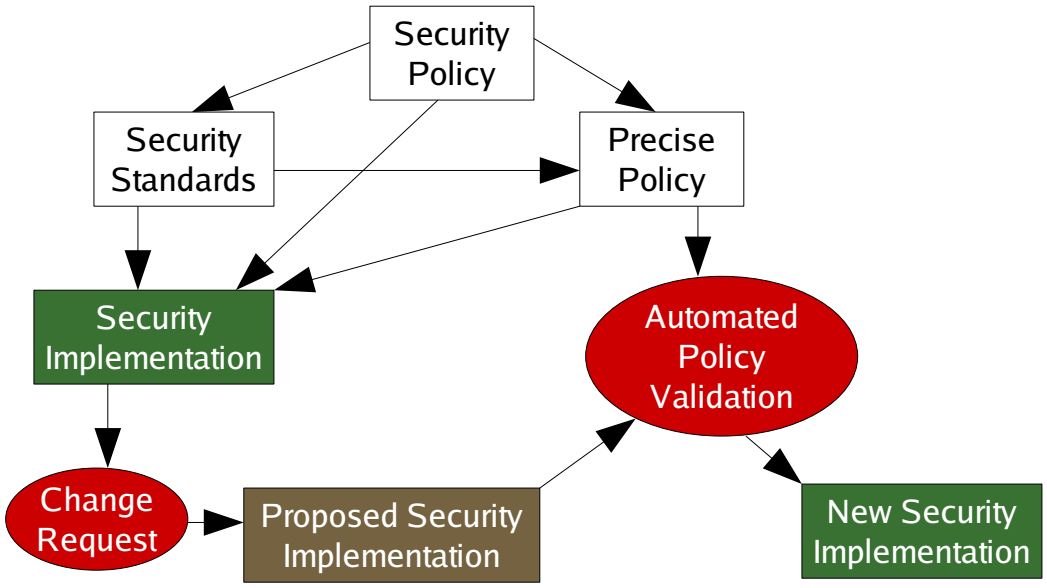

Figure 2: *If we can create a precise expression of the key requirements of the security policy, we can use the precise policy to drive an automated validation of any proposed security implementation changes.*

Existing Policy Validation and Verification Technologies

Today many technologies exist to verify that currently deployed technology is operating within the ranges specified by the security policy. The classic example of this post-deployment verification is auditing. Many operating system features and services generate audit logs after key events. IT staff can manually review the audit logs to look for unexpected events or write simple scripts to look for key events in the audit logs.

Security Information Management (SIM) tools such as Net Forensics can help with this log analysis. Other tools such as Cisco MARS use signatures to automatically correlate audit events from multiple sources (e.g., web syslogs and authentication logs) to identify potential securit problems.

In very specific areas, pre-deployment policy validation tools are already available. For example, password policy validation tools exist. Many authentication systems incorporate password checking tools to ensure that users only select "good" passwords (e.g., long enough, non-dictionary words). Many systems also implement password aging schemes to sure that password changing policies are enforced. Unlike the auditing solutions which verify

whether an action disagrees with policy after the action occurs, password policy enforcement can validate the implementation change before it is deployed.

Some technologies can already be deployed to help keep an organization's security implementation synchronized with the security policy. Some of these technologies detect actions outside of the policy (e.g. auditing technologies). Other technologies can validate whether the requested configuration change is within the security policy (e.g., password checkers). Ideally, we need to expand the scope of tools that validate security compliance before the configuration change is committed. With broad coverage of such tools, an organization can keep a tight synchronization between the security policy goals and the security mechanism realities.

Emerging Network Security Policy Validation Technologies

While there are high level policies in some technology spaces, there are significant areas of security implementation where vendor-specific configurations enforce the security implementation rather than a standardized formal security policy. Different security vendors for the similar types of security devices have different configuration models.

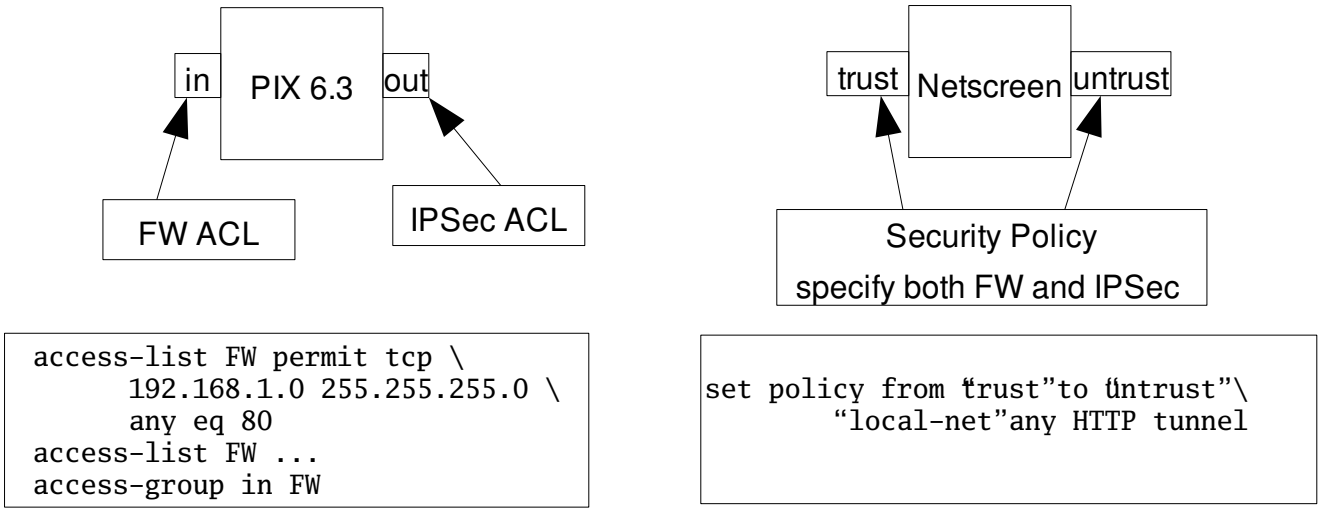

Figure 3: *Differences in how PIX and Netscreen associate firewall and tunnel rules with interfaces. PIX requires the definition of an access list for each feature. Netscreen defines both features in a single rule.*

For example, when configuring the firewall feature on PIX 6.3 [1], we must create an ordered access list and associate it with the appropriate interface via the access-group command. To configure IPSec another access list must be created and bound to an interface with the crypto map command.

However, to configure the same features on a Netscreen device[2] we must generate the set policy command for the appropriate pair of Zones. Zones label interfaces and the associated policy will apply to traffic leading from an interface labeled with the source zone and leaving the device on an interface labeled with the destination zone. Rather than creating multiple access lists, the Netscreen configuration language lets us enable firewall filtering and IPSec tunneling in the same policy configuration statement.

Policy and Implementation Mismatches

To explore the impact of the disconnect between policy and implementation, consider the following broad organizational policy.

> "Partners should only be given access to a specific set of partner servers and only necessary communication protocols should be permitted. Partner traffic must be filtered and analyzed before reaching company servers."

This policy will rarely if ever change. A more specific organizational policy or standard will further define how to identify partner traffic (through specific IP networks or via authenticated networks) and what necessary protocols. This standard may also elaborate what technology is sufficient to analyze the traffic. This policy or standard is more specific and therefore will change more often as technologies and partner details change.

For our example, assume that the partner IP networks are enumerated, the internal partner service addresses are enumerated, and the protocols HTTP, HTTPS, and SSH are deemed necessary for partner communication. In addition, our more specific document says that deep packet inspection or state-full analysis performed by most of today's enterprise firewalls is sufficient for the broader policy's analysis requirements.

When the set of partners is updated, the configurations will need to be changed to reflect the current set of valid partner addresses. This change maps directly onto the specific partner task and is likely to be performed without error. The more difficult problem is dealing with other change requests that have unintended consequences. Consider the following two scenarios.

Scenario 1: The system administrator is told to give FTP access to the sensitive document repository machine to machines from network 192.168.0.0/24. Say the configuration writer accidentally mistyped the mask so access was allowed from 192.168.0.0/23. If 192.168.1.0/24 was a partner network, this seemingly unrelated change causes a conflict between the security implementation and the policy on partner communication.

Scenario 2: The system administrator is instructed to give all internal users HTTP access to the new internal wiki server. If the configuration writer enters an early rule that specifies "any" as the source, partner addresses will also match this rule and get network access to the new wiki server. Perhaps this is policy writer's intention if partners are considered to be internal systems. It is likely that the security policy is ambiguous on this point. If so, it would be best

that this inconsistency is raised to someone's attention so either the implementation can be fixed or the policy can be refined.

In both of these scenarios the supposedly unrelated changes will affect how the security policy with respect to partner communication will be implemented. It is unlikely in reviewing these unrelated change, the IT folks will think to check for policy requirements that "should not" be affected by the current change. This error may not be caught until the next audit cycle. In the meantime, some partners have access to a sensitive internal machine that they should not.

Current Solutions

Today organizations rely on a combination of manual configuration review and network scanning to ensure that configuration changes to the network security implementation are still consistent with the guiding security policy. In many organizations, network security changes are only allowed within a particular change window each week. No configuration change will be deployed until it has been reviewed by the key IT staff. Hopefully, many eyes will catch potential problems.

Many organizations also use network traffic scanners such as *nmap* and *nessus* to probe the newly deployed configuration to ensure that only the expected traffic is passed. Based on the guiding security policy the IT staff can determine what to scan and review the scanning logs to see if there are any surprises. Generally, the scanning checks are made after the configuration changes are deployed in the production network. If the changes are extreme or an organization is extremely paranoid, they may deploy the changes in a similar network in an isolated lab and perform the scans in the lab before deploying the changes on the production network.

While these solutions catch many potential problems, it would be desirable to have a more precise understanding of how the network will behave with the proposed changes deployed without going through the trouble of setting up a separate lab or opening up the organization to a potential vulnerability window on their production network.

© Network Geographics 2007

Policy Constraint Validation

While the specifics of the vendor configuration languages differ, they all configure operations on the same basic packet model, which is *"Sp ecify a packet and describe what happens to it"*. The configuration language may require you to specify the packet for each feature as in the Cisco family of devices, or the configuration language may specify all features at once as in the Netscreen and Checkpoint models, but ultimately for a single packet, the configuration will specify all actions that the device will apply to it.

Thus, we can parse configurations into a canonical security policy model. Then we can perform a variety of policy checks against the model. The policy checks can be expressed in a common high level policy language and thus be isolated from the specifics of any particular vendor.

In the last ten years, a number of groups have developed technology to parse network security configurations into canonical forms for firewall management or analysis [3,4,5,6]. We have built the InfoSecter tool set based on results in this area to aid administrators in evolving network security configurations while staying within the bounds of the organization's security policy.

The InfoSecter framework takes network security device configurations and models how regions of potential packets will be processed using an efficient n-dimensional packet space[6]. With such a framework, one can efficiently perform many calculations of interest to the end user including rule conflict analysis, enumeration of functional changes between configurations, and configuration disambiguation over configurations approaching 100,000 lines.

From the natural language security policy, one can express in a formal policy the sets of flows and the desired actions (e.g., permit, proxy, tunnel). These formal policy constraints can then be used by InfoSecter to compute whether a particular network security configuration meets the requirements. If it does not, InfoSecter enumerates the lines of configuration that caused the divergence.

To take up our previous example with the partner communication restrictions, the IT staff could create a set of standing policy constraints based on the partner IP addresses, the internal partner server addresses, and the services (HTTP, HTTPS, and SSH) and the required actions (permit, inspect).

Constraint1 – Ensure that the actions specified by policy are allowed

```
source_address ^ (partner_net1 | ...  | partner_net_n) &
destination_address ^ (internal_server_net) &
destination_svc ^ (HTTP | HTTPS | SSH) &
action = (permit & inspect)
```

Constraint2 – Ensure that no other actions for the partner networks are allowed

```
source_address ^ (partner_net1 | ... | partner_net_n) &
(!(destination_address = internal_server_net) |
 !(destination_svc ^ (HTTP || HTTPS || SSH))) &
action = deny
```

The n-dimensional tree enables us to efficiently find the packets that match the constraints. Then we can walk the matching regions and determine if the actions match what is specified in the constraint.

The constraints could be tested as part of the standard operating procedure on each proposed changed configuration. Packets that match the packet specification but do not match the actions will be highlighted. In the case of the FTP configuration change with unexpected partner effects, the second constraint check will catch the mismatch with existing policy and alert the IT staff before the proposed configuration is even deployed.

One might ask once you can express the security policy as high level constraints, why not just generate the correct implementation directly. There has been work in this area [7,8,9]. However, for a variety of technical and social issues these policy-driven configuration generation solutions have not gained wide acceptance

In general, the problem with the policy-driven configuration is the all-or-nothing aspect of the solution. Some aspects of the configuration may map very well to the high level policy, but perhaps other aspects of the configuration are easier done by someone with expertise in the configuration language. Some people in the organization may be comfortable working with the policy language, but others are more comfortable with the command line. In general it is hard to have a configuration that is sometimes configured from policy generation and sometimes through direct human editing[10].

Therefore, in the near term, using high level policy specifications as constraints seems like a useful yet incremental trade-off between strict policy enforcement and operation in the low level vendor specific configuration languages that the day to day administrators are most comfortable with.

Conclusions

Natural language policies are being used by most organizations to drive the development of their security implementations. However, there is a disconnect from the natural language policy to the security implementation which means that day-to-day changes may cause the security implementation to drift from the natural language security policy goals,

By enforcing a more direct path from the natural language policy the security implementation, an organization can have a higher degree of assurance that the security implementation meets the security policy constraints at any point of time.

In this paper we outlined existing technologies that are used to validate and verify changes to the security implementation, and we explored how formal policy specifications can be used to

directly drive implementation configuration and more promisingly in the short term to drive policy validation of security configurations created through a variety of means. Network security configuration is currently distributed, complicated, and error prone. The automatic verification of formal policy on network security configuration changes can help in this domain, greatly improving an organization's assurance of security.

This approach of formal policy validation can be applied to other components of a network security architecture. Adding formal high level policy fleshes out the security policy refinement hierarchy making security expectations more explicit and the security implementation more accurate. Validation tools and formal policy specifications will enable an iterative security policy development process enabling a architecture that is assured on each configuration change reducing the number of security holes for the attackers to exploit.

References

1. *Cisco PIX Firewall 6.3 Command Reference*. Cisco Systems, Inc., http://www.cisco.com/en/US/docs/security/pix/pix63/command/reference/cmdref.html.
2. *ScreenOS CLI Reference Guide: IPv4 Command Descriptions, Release 6.0.0,* Juniper Networks, Inc., http://www.juniper.net/techpubs/software/screenos/screenos6.0.0.
3. J. D. Guttman, "Filtering Postures: local enforcement for global policies," In *IEEE Security and Privacy Symposium*, Oakland, CA, 1997.
4. Ehab Al-Shaer and Hazem Hamed , "Modeling and Management of Firewall Policies." In *IEEE Transactions on Network and System Management*, Volume 1-1, April 2004.
5. A. Mayer, A. Wool, and E. Ziskind. "Offline firewall analysis." *International Journal of Information Security*, 5(3):125-144, 2006.
6. J. Qian, S. Hinrichs, K. Nahrstedt, "ACLA: A Framework for Access Control List (ACL) Analysis and Optimization." In *Proceedings of Communications and Multimedia Security*, 2001.
7. S. Hinrichs, "Policy-Based Management: Bridging the Gap." In *Proc. of the 15th Annual Security Applications Conference*, IEEE, Phoenix, AZ, December, 1999.
8. Y. Bartal, A. Mayer, K. Nissim, and A. Wool. "Firmato: A novel firewall management toolkit." *ACM Transactions on Computer Systems*, 22(4):381-420, November 2004.
9. *Solsoft Policy Server*, http://www.solsoft.com/pages/download/index.php.
10. S. Hinrichs, "Integrating Changes to a Hierarchical Policy Model", In *Proceedings of 9th IFIP/IEEE International Symposium on Integrated Network Management*, IEEE, Nice, France, May, 2005.