Fourth LACCEI International Latin American and Caribbean Conference for Engineering and Technology (LACCET'2006)
"Breaking Frontiers and Barriers in Engineering: Education, Research and Practice"
21-23 June 2006, Mayagüez, Puerto Rico.

Security Patterns and Secure Systems Design

Eduardo B. Fernández, PhD
Professor of Computer Science & Engineering,
Florida Atlantic University, 777 Glades Road SE-408, Boca Raton, Florida 33431-0991, USA,
ed@cse.fau.edu

María M. Larrondo Petrie, PhD
Professor and Associate Dean of the College of Engineering & Computer Science,
Florida Atlantic University, 777 Glades Road SE-308, Boca Raton, Florida 33431-0991, USA,
petrie@fau.edu

Abstract
Analysis and design patterns are well established as a convenient and reusable way to build high-quality object-oriented software. Patterns combine experience and good practices to develop basic models that can be used for new designs. Security patterns join the extensive knowledge accumulated about security with the structure provided by patterns to provide guidelines for secure system design and evaluation. A variety of security patterns has been developed for the construction of secure systems. These patterns include Authentication, Authorization, Role-based Access Control, Firewalls, Protected Execution Environment, and others. These patterns can be combined to build more complex architectures such as Single-Sign-On architectures, web services authorization, authorized applications, and others. We can apply these patterns through a secure system development method that uses different mechanisms based on a hierarchical architecture whose layers define the scope of each security mechanism. We are building a catalog of security patterns that helps in defining the security mechanisms at each architectural level and at each development stage. In addition to their value for new system design, security patterns are useful to evaluate existing systems by analyzing if they include specific patterns or not. They are also useful to compare security standards and to verify that products comply with the standard. Finally, we have found security patterns very valuable for teaching security concepts and mechanisms.

Keywords
Security, Design Patterns, Software Architecture, Systems Design, Software Engineering

1. Introduction

There is no doubt that the appearance of design patterns [Gam94] was one of the most important developments in software engineering of the last 20 years. Design patterns embody the experience and knowledge of many designers and when properly catalogued, they provide a repository of solutions for useful problems. They have shown their value in many projects and have been adopted by many institutions, including companies such as IBM, Microsoft, Siemens, Motorola, Sun, and others. Design

patterns were extended to other aspects of software, first to architectural aspects of the design [Bus96], then to the analysis stage [Fow97]. This was followed by their use in process aspects [Cop05], organizational aspects [Kol, Man05], and pedagogical aspects.

Analysis and design patterns are now well established as a convenient and reusable way to build high-quality object-oriented software. A pattern solves a specific problem in a given context and can be tailored to fit different situations. Analysis patterns can be used to build conceptual models, design patterns can be used to make software more flexible and reusable, and security patterns can be used to build secure systems. Security has had a long trajectory, starting from the early models of Lampson [Lam71] and Bell/LaPadula [Fer06c] in the early 70s, and resulting in a variety of approaches to analyze security problems and to design security mechanisms. It is natural to try to codify this expertise in the form of patterns.

Security patterns join the extensive knowledge accumulated about security with the structure provided by patterns to provide guidelines for secure system design and evaluation. Security patterns describe a precise generic model for a security mechanism. An example is a pattern for a packet filter firewall (see Section 2), where a conceptual class model is shown that describes the architecture of packet firewalls. The model includes also sequence diagrams for some use cases. The implementation section describes how this pattern can be applied to build a new system or to describe a real product. Security patterns can be considered a type of architectural patterns.

Yoder and Barcalow wrote the first paper on security patterns [Yod97]. They included a variety of patterns useful in different aspects of security. Before them, at least three papers [Fer93, Fer94, Ess97] had shown object-oriented models of secure systems without calling them patterns or using one of the standard pattern templates. In the next year (1998), two more patterns appeared: a pattern for cryptography [Bra00], and a pattern for access control [Das98]. After that, several others have appeared and we have now a substantial collection [Ste05, Schu06], some of which are mentioned here.

We have applied these patterns through a secure system development methodology based on a hierarchical architecture whose layers define the scope of each security mechanism [Fer06a]. In addition to their value for new system design, security patterns are useful to evaluate existing systems by analyzing if they include specific patterns or not. They are also useful to compare security standards and to verify that products comply with the standard. Finally, we have found security patterns very valuable for teaching security concepts and mechanisms.

Section 2 analyzes a security pattern in detail, while Section 3 relates patterns to the architectural layers of the system. Section 4 describes our methodology for secure systems design using patterns. We end with some conclusions.

2. Anatomy of a security pattern: The Packet Filter Firewall

We look here at the structure of our patterns from [Sch06].

Every pattern starts with a thumbnail of the problem it solves and maybe a brief description of how it solves the problem.

> *The Packet Filter Firewall filters incoming and outgoing network traffic in a computer system based on packet inspection at the IP level.*

Then we give an example of a problematic situation where this pattern is not yet used:

Example

Our system has been attacked recently by a variety of hackers, including somebody who penetrated our operating system and stole our clients' credit card numbers. Our employees are wasting time at work by looking at inappropriate sites in the Internet. If we continue like this we will be out of business soon.

We define the context where the pattern solution is applicable:

Context

Computer systems on a local network connected to the Internet and to other networks with different levels of trust. A host in a local network receives and sends traffic to other networks. This traffic has several layers or levels. The most basic level is the IP level, made up of packets consisting of headers and bodies (payloads). The headers include the source and destination addresses as well as other routing information, the bodies include the message payloads.

Now a generic description of what happens when we don't have a good solution: We also indicate the forces that affect the possible solution.

Problem

Some of the hosts in other networks may try to attack the local network through their IP-level payloads. These payloads may include viruses or application-specific attacks. We need to identify and block those hosts.

The possible solution is constrained by the following forces:

- *We need to communicate with other networks so isolating our network is not an option. However, we do not want to take a high risk for doing so.*
- *The protection mechanism should be able to reflect precisely the security policies of the institution. A too coarse defense may not be useful.*
- *Any protection mechanism should be transparent to the users. Users should not need to perform special actions to be secure.*
- *The cost and overhead of the protection mechanism should be relatively low or the system may become too expensive to run.*
- *Network administrators deploy and configure a variety of protection mechanisms; hence it is important to have a clear model of what is being protected.*
- *The attacks are constantly changing; hence it should be easy to make changes to the configuration of the protection mechanism.*
- *It may be necessary to log input and/or output requests for auditing and defense purposes.*

The solution section describes the idea of the pattern. A descriptive figure may help to visualize the solution.

Solution

A Packet Filter Firewall intercepts all traffic coming/going from a port P and inspects its packets (Figure 1). Those coming from or going to untrusted addresses are rejected. The untrusted addresses are determined from a set of rules that implement the security policies of the institution. A client from another network can only access the Local Host if a rule exists that authorizes traffic from its address. Specific rules may indicate an address or a range of

addresses. Rules may be positive (allow traffic from some address) or negative (block traffic from some address). Most commercial products order these rules for efficiency in checking. Additionally, if a request is not satisfied by any of the Explicit Rules, then a Default Rule is applied.

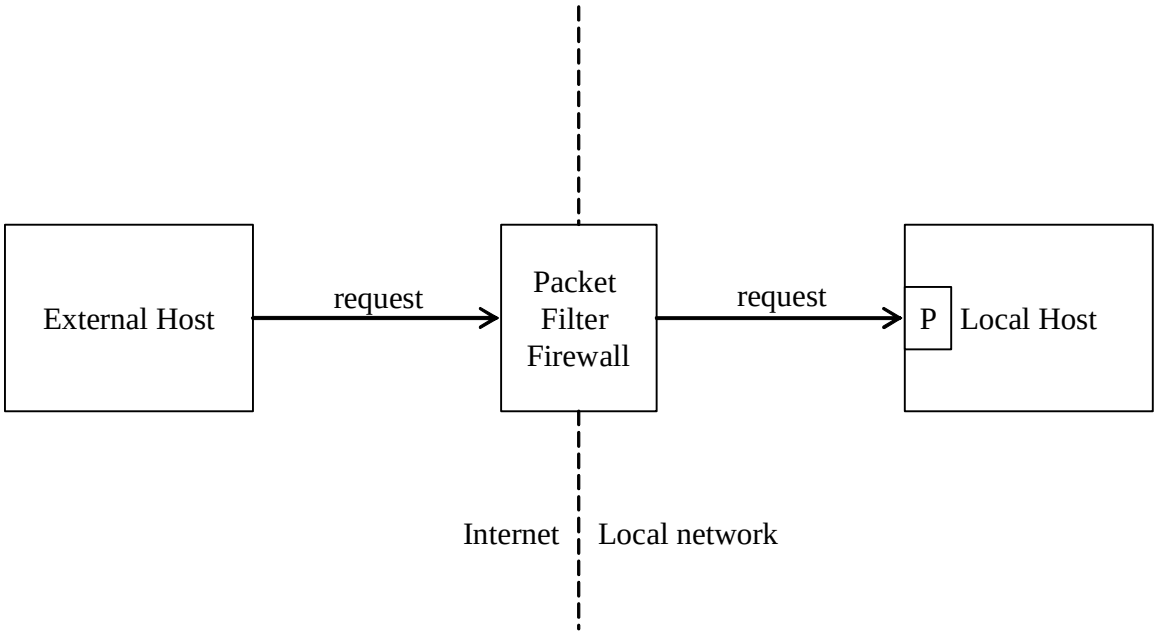

Figure 1 Idea of the packet filter firewall

We then describe the structure (static view) of the solution and some dynamic aspects in the form of sequence diagrams for a use case.

Structure

*Figure 2 shows an **External Host** requesting access to a **Local Host** (a server), through a **Packet Filter Firewall (PFFirewall)**. The institution policies are embodied in the objects of class **Rule** collected by the **RuleBase**. The RuleBase includes data structures and operations to manage rules in a convenient way. The rules in this set are ordered and can be **Explicit** or **Default**.*

Dynamics

We describe the dynamic aspects of the Packet Filter Firewall using a sequence diagram for one of its basic use cases. There is a symmetric use case, Filtering an outgoing request, which we omit for briefness. We also omit use cases for adding, removing, or reordering rules because they are straightforward.

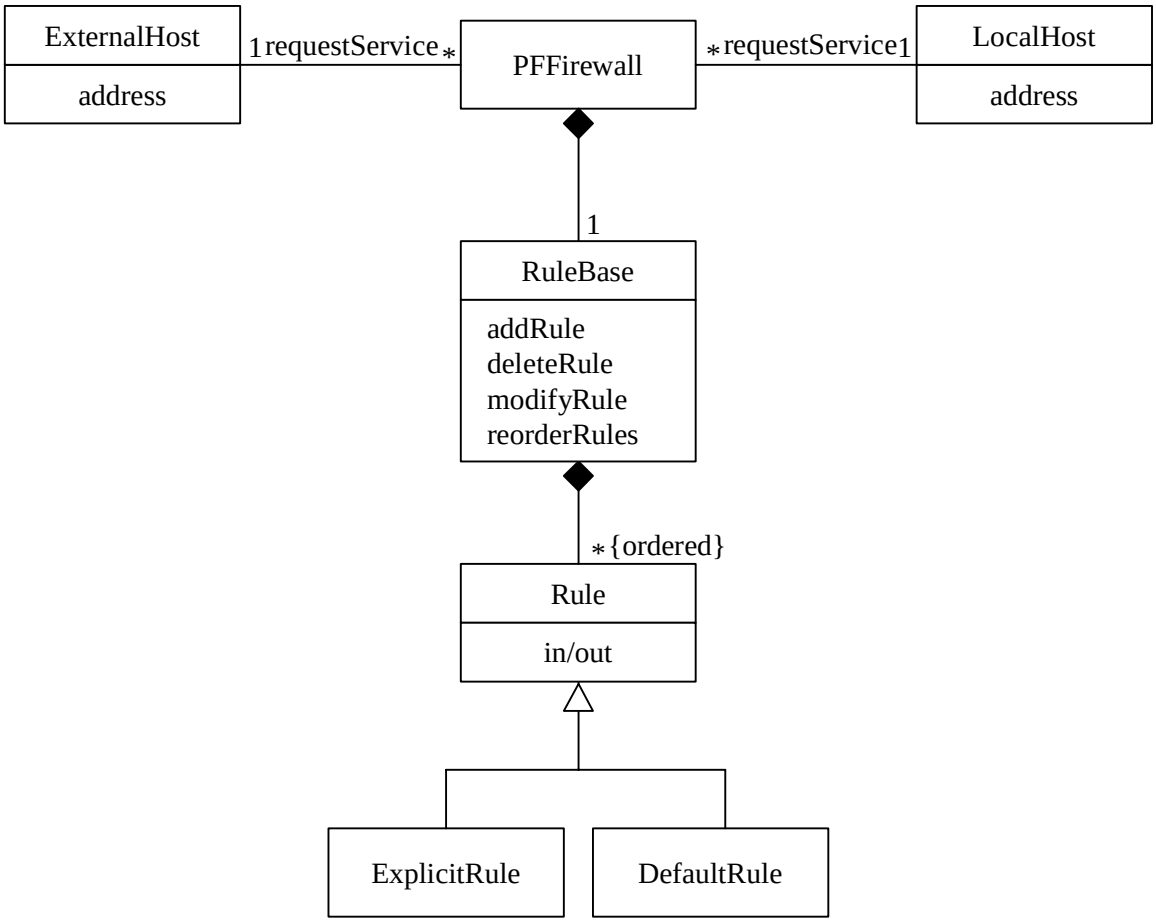

Figure 2: Class diagram for Packet Filter Firewall pattern

Filtering a Client's Request (Figure 3).

- *Summary: A host in a remote network wants access to a local host to either transfer or retrieve information. The access request is made through the firewall, which according to its set of rules determines whether to accept or deny the request, i.e., it filters the access request.*
- *Actors: A host in an external network (client).*
- *Precondition: An existing set of rules to filter the request must be in place in the firewall.*
- *Description:*
  a. *An external host requests access to the local host.*
  b. *A firewall filters the request according to a set of ordered rules. If none of the explicit rules in the rule set allows or denies the request, a default rule is used for making a decision.*
  c. *If the request is accepted, the firewall allows access to the local host.*
- *Alternate Flow: The request is denied.*
- *Postcondition: The firewall has accepted the access of a trustworthy client to the local host.*

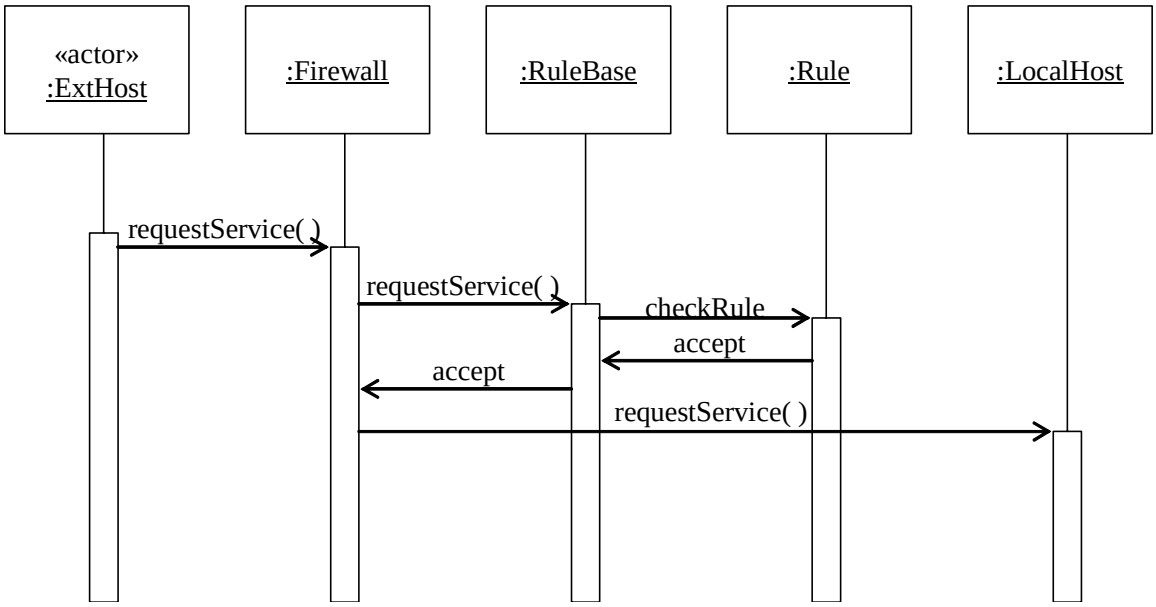

Figure 3: Sequence diagram for filtering a client's request

What one should consider when implementing the pattern is the objective of the next section. This can be a set of general recommendations or a sequence of what to do to use the pattern. It may include some sample code if appropriate.

Implementation

1. *Define an institution policy about network access, classifying sites according to our trust in them.*
2. *Convert this policy into a set of access rules. This can be done manually, which may be complex for large systems. An alternative is using an appropriate commercial product. .*
3. *Note that the idea of a single point of access is virtual, there may be several physical firewalls deployed at different places. This means it is necessary to install firewalls at all external boundaries (routers or gateways).*
4. *Write the rules in each firewall. Again, products such as Solsoft and others automatically propagate the rules to each registered firewall.*
5. *Configure the corresponding firewalls according to standard architectures. A common deployment architecture is the Demilitarized Zone (DMZ) [Sch06].*

Now we see what happens in the example after the pattern solution has been applied.

Example resolved

We were able to trace the addresses of our attackers and we got a firewall to block requests from those addresses from reaching our system. We also made a list of addresses of inappropriate sites and blocked access to them from the hosts in our network. All this reduced the number of attacks and helped control the behavior of some employees.

The Consequences section indicates the advantages and disadvantages of the solution embodied in this pattern. The advantages should match the forces in the Problem section.

Consequences

The Packet Filter Firewall Pattern has the following advantages:

- *A firewall transparently filters all the traffic that passes through it, thus lowering the risk of communicating with potentially hostile networks.*
- *It is possible to express the institution filtering policies through its filtering rules, with different levels of protection for different parts of the network.*
- *It is easy to update the rule set to counter new threats.*
- *Because it intercepts all requests, a firewall allows systematic logging of incoming and outgoing messages. Because of this, a firewall facilitates the detection of possible attacks and helps to hold local users responsible of their actions when interacting with external networks.*
- *Low cost, it is included as part of many operating systems and simple network devices such as routers.*
- *Good performance. It only needs to look at the headers of IP packets, not at the complete packet.*
- *It can be combined with Intrusion Detection Systems (IDS) for greater effectiveness. In this case, the IDS can tell the firewall to block suspicious traffic. This can also be useful to control Distributed Denial of Service (DDoS) attacks.*

The Packet Filter Firewall Pattern has the following (possible) liabilities:

- *The firewall's effectiveness and speed may be limited due to its rule set (order of precedence). Addition of new rules may interfere with existing rules in the rule set; hence, a careful approach should be taken in adding and updating access rules.*
- *The firewall can only enforce security policies on traffic that goes through the firewall. This means that one must make changes to the network to ensure that there are no other paths into its hosts.*
- *An IP-level firewall cannot stop attacks coming through the higher levels of the network. For example, a hacker could put malicious commands or data in header data not used for routing and in the payload.*
- *Each packet is analyzed independently, which means that it is necessary to analyze every packet. This may reduce performance.*
- *A packet filter cannot recognize forged addresses (IP spoofing) because it only examines the header of the IP packet. This can be corrected (at some extra cost) using Link Layer filtering, where each IP address is correlated to its hardware address [Fra01].*

To accept this solution as a pattern we should find at least three examples of its use in real systems.

Known Uses

This model corresponds to an architecture that is seen in commercial firewall products, such as: ARGuE (Advanced Research Guard for Experimentation), which is based on Network Associates' Gauntlet Firewall, OpenBSD Packet Filtering Firewall, which is the basic firewall architecture for the Berkeley Software Distribution system; and, the Linux Firewall, which is the basic firewall architecture used with the Linux operating system. The Packet filter firewall is used as an underlying architecture for other types of firewalls that include more advanced features.

Finally, we relate our pattern to other known patterns. Those may be complementary patterns, variations of our pattern, or extensions of it.

Related Patterns

The Authorization pattern [Fer01] defines the standard security model for the Packet Filter Firewall Pattern. This pattern is also a special case of the Single-Point-of-Access [Sch06] and it is the basis for other, more complex, types of firewalls (described in the other patterns in this language). The DMZ pattern [Sch06] defines a way to configure this pattern in a network. This pattern can also be combined with the Stateful Inspection Firewall [Sch06].

Some researchers think that one should also indicate explicitly what attacks the security pattern can prevent or mitigate. We prefer to indicate them as part of the regular sections.

3. Patterns and architectural layers

We can think of a computer system as a hierarchy of layers, where the application layer uses the services of the database and operating system layers, which in turn, execute on a hardware layer. In fact, this structure is a pattern in itself [Bus96]. Two basic principles of security are:

- Security constraints should be defined at the highest layer, where their semantics are clear, and propagated to the lower levels, which enforce them.
- All the layers of the architecture must be secure.

Our use of patterns is guided by these principles. We can define patterns at all levels. This allows a designer to make sure that all levels are secured, and also makes easier propagating down the high-level constraints. We survey now some of the patterns that apply to each layer, starting from the application layer.

At the abstract level we have patterns that describe security models. These models can be applied to define application constraints or policies that are then propagated down. Some of these patterns include:

- **Authorization** [Sch06]. How do we describe who is authorized to access specific resources in a system? Keep a list of authorization rules describing who has access to what and how.
- **Role-Based Access Control (RBAC)** [Fer01, Sch06]. How do we assign rights to people based on their functions or tasks? Assign people to roles and give rights to these roles so they can perform their tasks. **The Role-Rights Definition Pattern** [Sch06]. Least privilege is a fundamental principle for secure systems. Roles can directly support the least privilege principle, but there must be some methods to assign only the needed rights to each role. This pattern provides a precise way to assign rights to implement least privilege.
- **Reference Monitor** [Fer02, Sch06]. How to enforce authorizations when a process requests access to an object? Define an abstract process that intercepts all requests for resources from processes and checks them for compliance with authorizations.
- **Multilevel Security pattern** [Fer01, Sch06]. How to decide access in an environment with security classifications.

The Layers pattern, one of the fundamental patterns of [Bus96], was reinterpreted as a security pattern in [Fer01] and [Yod99].

Patterns for operating systems were developed in [Fer02] and [Fer03a] (combined in [Sch06]). Figure 4 shows a pattern diagram that relates some of these patterns (the ones with double lines are described on [Fer06c].

- **Controlled Virtual Address Space** [Fer02]. How to control access by processes to specific areas of their virtual address space (VAS) according to a set of predefined rights? Divide the VAS into segments that correspond to logical units in the programs. Use special words (*descriptors*) to represent access rights for these segments.
- **Controlled-Process Creator** [Fer03]. How to define the rights to be given to a new process? Define rights as part of its creation. Give it a predefined subset of its parent's rights.
- **File access control**. How do you control access to files in an operating system? Apply the Authorization pattern to describe access to files by subjects. The protection object is now a file component that may be a directory or a file.
- **Controlled Execution Environment**. How to define an execution environment for processes? Attach to each process a set of descriptors that represent the rights of the process. Use the Reference Monitor to enforce access.
- **Controlled-Object Factory**. How to specify rights of processes with respect to a new object? When a process creates a new object through a Factory, the request includes the features of the new object. Among these features include a list of rights to access the object.
- **Controlled-Object Monitor**. How to control access by a subject to an object? Use a reference monitor to intercept access requests from processes. The reference monitor checks if the process has the requested type of access to the object.
- **Operating system architectures** [Fer05a]. Four patterns describe possible ways to structure an operating system.
- **Secure Process** [Fer06c]. How do we make the execution of a process secure? A process is a program in execution and the unit of execution in some operating systems. A secure process is also a unit of execution isolation as well as a holder of rights to access resources.
- **Secure Thread** [Fer06c]. How do we make the execution of a thread secure? A thread is a lightweight process. A secure thread is a thread with controlled access.
- **Administrator Hierarchy** [Fer06c]. How do we restrict access for administrators? Defines a hierarchy of system administrators with controlled rights using a Role-Based Access Control (RBAC) model.

Patterns for firewalls are discussed in [Sch06]:

- **Packet Filter Firewall**. Filter incoming and outgoing network traffic in a computer system based on network addresses.
- **Application Proxy Firewall .** Inspect (and filter) incoming and outgoing network traffic based on the type of application they are accessing.
- **Stateful firewall** Filter incoming and outgoing network traffic in a computer system based on network addresses and the state information derived from past communications.

Patterns for distributed systems include the Bodyguard [Das98], a framework for access control and filtering of distributed objects, combining several patterns [Hay00]. A pattern for a Remote Secure Proxy is given in [Amo01]. Authentication in distributed systems is considered in:

- **Authenticator**.[Bro99, Fer02]. How to verify that a subject is who it says it is? Use a single point of access to receive the interactions of a subject with the system and apply a protocol to verify the identity of the subject.
- **Remote Authenticator /Authorizer** [Fer03b]. Provide facilities for authentication and authorization when accessing shared resources in a loosely-coupled distributed system.

Pattern languages for cryptography are described in [Bra00] and [Let01].

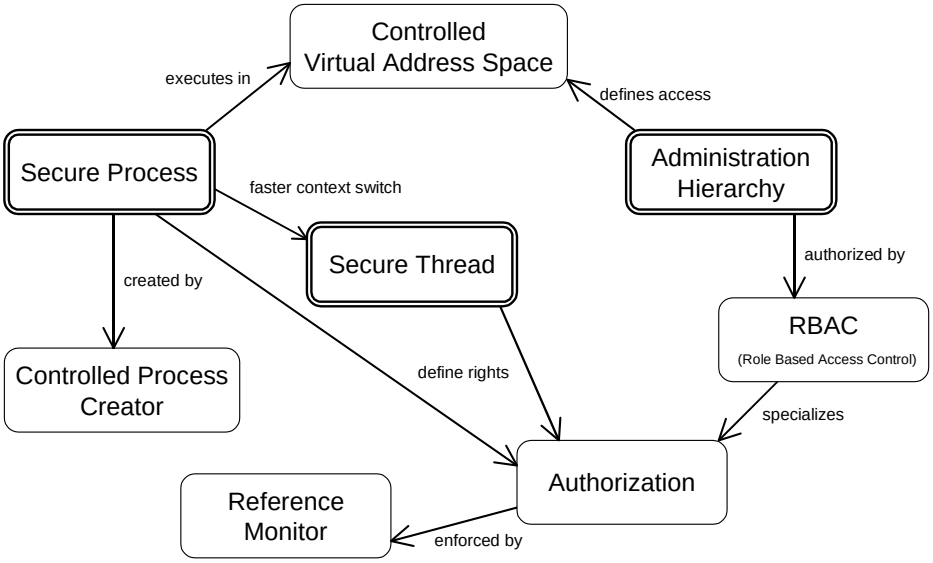

Figure 4. Pattern diagram for some of the operating system patterns

4. A methodology for secure systems design

A main idea in the proposed methodology is that security principles should be applied at every stage of the software lifecycle and that each stage can be tested for compliance with security principles. Another basic idea is the use of patterns to guide security at each stage. This project proposes guidelines for incorporating security from the requirements stage through analysis, design, implementation, testing, and deployment. It considers the following development stages:

Domain analysis stage: A business model is defined. Legacy systems are identified and their security implications analyzed. Domain and regulatory constraints are identified. Policies must be defined up front, in this phase. The suitability of the development team is assessed, possibly leading to added training. Security issues of the developers, themselves, and their environment may also be considered in some cases. This phase may be performed only once for each new domain or team.

Requirements stage: Use cases define the required interactions with the system. Applying the principle that security must start from the highest levels, it makes sense to relate attacks to use cases. We study each action within a use case and see which threats are possible (this paper). We then determine which policies would stop these attacks. From the use cases we can also determine the needed rights for each actor and thus apply a need-to-know policy. Note that the set of all use cases defines all the uses of the system and from all the use cases we can determine all the rights for each actor. The security test cases for the complete system are also defined at this stage.

Analysis stage: Analysis patterns can be used to build the conceptual model in a more reliable and efficient way. Security patterns describe security models or mechanisms. We can build a conceptual model where repeated applications of a security model pattern realize the rights determined from use cases. In fact, analysis patterns can be built with predefined authorizations according to the roles in their use cases. Then we only need to additionally specify the rights for those parts not covered by patterns. We can start defining mechanisms (countermeasures) to prevent attacks.

Design stage: Design stage: when we have the possible attacks to a system, design mechanisms are selected to stop these attacks. User interfaces should correspond to use cases and may be used to enforce the authorizations defined in the analysis stage. Secure interfaces enforce authorizations when users interact with the system. Components can be secured by using authorization rules for Java or .NET components [Ste05]. Distribution provides another dimension where security restrictions can be applied. Deployment diagrams can define secure configurations to be used by security administrators. A multilayer architecture is needed to enforce the security constraints defined at the application level. In each level we use patterns to represent appropriate security mechanisms. Security constraints must be mapped between levels.

Implementation stage: This stage requires reflecting in the code the security rules defined in the design stage. Because these rules are expressed as classes, associations, and constraints, they can be implemented as classes in object-oriented languages. In this stage we can also select specific security packages or COTS, e.g., a firewall product, a cryptographic package. Some of the patterns identified earlier in the cycle can be replaced by COTS (these can be tested to see if they include a similar pattern).

5. Conclusions

We considered the use of security patterns and looked in detail at one of them. We surveyed some patterns previously developed by us and others. In addition to their value for designing new systems, patterns are also useful when selecting a system or an application. Possible candidates can be compared according to having or not a pattern embodying a given function or capability. For example, the presence of a Role-Based Control pattern in a system indicates its support for specific features of this model, e.g. sessions or groups.

Finally, we have used patterns for teaching and explaining security aspects [Fer05b]. The abstraction present in patterns eliminates the effect of implementation details and is very valuable to make a complex structure more understandable. As compared with formal methods for the same purposes, we can see numerous advantages, including their improved ability to represent structural properties, their intuitiveness, and the fact that published patterns are almost guaranteed to be error free. Future work will include completing our methodology and the development of further patterns.

Security patterns are now accepted by many companies, Microsoft. Sun, and IBM have books, papers, and web pages on this subject. A general page for security patterns also exists [sec].

Acknowledgements

This work was supported through a Federal Earmark grant from Defense of Information Systems Agency (DISA), administrated by Pragmatics, Inc.

References

[Amo01] M.B. d'Amorim, "Proxy-to-Proxy, a structural pattern for leveraging security on highly distributed Internet applications", *Procs. of SugarLoafPLoP*, 2001. http://www.cos.ufrj.br/~sugarloafplop/

[Bra00] A. Braga, C. Rubira, and R. Dahab, "Tropyc: A pattern language for cryptographic object-oriented software", Chapter 16 in *Pattern Languages of Program Design* 4 (N. Harrison, B. Foote, and H. Rohnert, Eds.). Also in Procs. of PLoP'98, http://jerry.cs.uiuc.edu/~plop/plop98/final_submissions/

[Bro99] F. Lee Brown, J. DiVietri, G. Diaz de Villegas, and E. B. Fernandez, "The Authenticator pattern", *Procs. of PLOP'99*, http://st-www.cs.uiuc.edu/~plop/plop99.

[Bus96] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerland, and M. Stal., *Pattern- oriented software architecture*, John Wiley & Sons, 1996.

[Cop05] J.O.Coplien and N.B.Harrison, *Organizational patterns of agile software development*, Prentice Hall, 2005.

[Das98] F. Das Neves and A. Garrido, "Bodyguard", Chapter 13 in *Pattern Languages of Program Design 3*, Addison-Wesley 1998.
[Ess97] W. Essmayr, G. Pernul, and A.M. Tjoa, "Access controls by object-oriented concepts", *Proc. of 11th IFIP WG 11.3 Working Conf. on Database Security*, August 1997.
[Fer93] E.B.Fernandez, M.M.Larrondo-Petrie and E.Gudes, "A method-based authorization model for object-oriented databases", *Proc. of the OOPSLA 1993 Workshop on Security in Object-oriented Systems* , 70-79.
[Fer94] E. B. Fernandez, J. Wu, and M. H. Fernandez, "User group structures in object-oriented databases", *Proc. 8th Annual IFIP W.G.11.3 Working Conference on Database Security*, Bad Salzdetfurth, Germany, August 1994.
[Fer01] E.B. Fernandez and R.Y. Pan, "A pattern language for security models", *Procs. of PLoP 2001*, http://jerry.cs.uiuc.edu/~plop/plop2001/accepted_submissions/accepted-papers.html
[Fer02] E.B.Fernandez, "Patterns for operating systems access control", *Procs. of PLoP 2002*, http://jerry.cs.uiuc.edu/~plop/plop2002/proceedings.html
[Fer03a] E.B.Fernandez and J.C.Sinibaldi, "More patterns for operating systems access control", *Procs. EuroPLoP 2003*, http://hillside.net/europlop
[Fer03b] E.B. Fernandez and R. Warrier, "Remote Authenticator /Authorizer", *Procs. of PLoP 2003.*
[Fer05a] E.B.Fernandez and T. Sorgente, "A pattern language for secure operating system architectures", *Procs. of the 5th Latin American Conference on Pattern Languages of Programs*, Brazil, August 16-19, 2005.
[Fer05b] E.B.Fernandez and M. M. Larrondo-Petrie, "Teaching a course on data and network security using UML and patterns", *Procs. of the Educators Symposium of MoDELS/UML 2005*, Montego Bay, Jamaica, October 2-7, 2005.
[Fer06a] E. B. Fernandez, M.M. Larrondo-Petrie, T. Sorgente, and M. VanHilst, "A methodology to develop secure systems using patterns", in *Integrating security and software engineering: Advances and future vision*, H. Mouratidis and P. Giorgini (Eds.).
[Fer06b] E. B. Fernandez, E. Gudes, and M. Olivier, *The Design of Secure Systems*, Addison-Wesley 2006.
[Fer06c] E.B.Fernandez, T. Sorgente, and M.M. Larrondo-Petrie, "Even more patterns for secure operating systems", (to appear).
[Fow97] M. Fowler, *Analysis patterns -- Reusable object models*, Addison- Wesley, 1997.
[Gam94] E. Gamma, R. Helm,R. Johnson, and J. Vlissides, *Design patterns –Elements of reusable object-oriented software*, Addison-Wesley 1994.
[Hay00] V. Hays, M. Loutrel, and E.B.Fernandez, "The Object Filter and Access Control framework", *Procs. Pattern Languages of Programs (PLoP2000) Conference*, http://jerry.cs.uiuc.edu/~plop/plop2k
[Kol] M. Kolp1, P. Giorgini, and J. Mylopoulos, "Organizational Patterns for Early Requirements Analysis", http://www.cs.toronto.edu/~mkolp/caiseorgpat.pdf
[Kon03] S. Konrad, L.A. Campbell, B.H.C. Cheng, and M. Den, "A Requirements Patterns-Driven Approach to Specify Systems and Check Properties" *10th International SPIN Workshop*, T. Ball and S.K. Rajani (eds). LNCS Vol. 2648, Springer Verlag, 2003. http://www.cse.msu.edu/~konradsa/Publications/spin03.pdf
[Lam71] B.W. Lampson, "Protection", *Procs. 5th Annual Conf. on Info. Sciences and Systs.*,1971, 437-443. Reprinted in *ACM Operating Systs. Review*, 8, 1 (January 1974), 18-24.
[Let01] S. Lehtonen and J. Parssinen, "A pattern language for key management", *Procs. of PLoP 2001*, http://jerry.cs.uiuc.edu/~plop/plop2001/accepted_submissions/accepted-papers.html
[Man05] M.L. Manns and L. Rising, *Fearless change: Patterns for introducing new ideas*, Addison-Wesley, 2005.
[Sch06] M. Schumacher, E.B.Fernandez, D. Hybertson, F. Buschmann, and P. Sommerlad, *Security Patterns*, J. Wiley & Sons, 2006.
[sec] "The Security Patterns page", maintained by M. Schumacher, http://www.securitypatterns.org
[Ste05] C. Steel, R. Nagappan, and R. Lai, *Core Security Patterns: Best Strategies for J2EE, Web Services, and Identity Management*, Prentice Hall, Upper Saddle River, New Jersey, 2005.
[Yod97] J. Yoder and J. Barcalow, "Architectural patterns for enabling application security". *Procs. PLOP'97*, http://jerry.cs.uiuc.edu/~plop/plop97 Also Chapter 15 in *Pattern Languages of Program Design*, vol. 4 (N. Harrison, B. Foote, and H. Rohnert, Eds.), Addison-Wesley, 2000.

Authorization and Disclaimer

Authors authorize LACCEI to publish the papers in the conference proceedings. Neither LACCEI nor the editors are responsible either for the content or for the implications of what is expressed in the paper.