2008-06-01

Formal models of a least privilege separation kernel in alloy

Phelps, David; Levin, Timothy E.; Auguston, Mikhail

International Conference on i-Warfare and Security

Proceedings of the International Conference on i-Warfare and Security, April 2008.
http://hdl.handle.net/10945/7158

Formal Models of a Least Privilege Separation Kernel in Alloy

David Phelps, Mikhail Auguston, Timothy E. Levin
Naval Postgraduate School
Monterey, California, United States
david.a.phelps@navy.mil
maugusto@nps.edu
levin@nps.edu

Abstract: We describe the specification of the formal security policy model and formal top-level specification for the Least Privilege Separation Kernel (LPSK) in Alloy, a relatively new modeling language and analysis tool. The state of the art for the formal verification of secure software requires representation of an abstract model, and one or more refinements (to the model), in a formal specification language. These specifications are then examined for self-consistency with their properties, as well as for consistency between levels of abstraction, all of which can be time consuming, and costly. Alloy provides a simple, intuitive logic framework, in contrast to many other formal languages that are intended to support general-purpose mathematics. In order to determine whether Alloy can improve the efficiency and effectiveness of the verification of secure computer systems, we used it to specify portions of the LPSK formal security policy model and formal top-level specification, and utilized the Alloy Analyzer to examine the consistency of the specifications. The security-critical system elements and predicates for security properties were defined in terms of a state model, and system operations were represented as state transitions. While Alloy does not support induction or proofs, it can be used to find counter examples in a small scope of state transitions. We conclude that Alloy has few limitations and is suitable, as measured by utility and ease of use, to include in the toolbox for rapid high-assurance system development. The primary concern with using Alloy for industrial, versus academic, security verification is the scalability of the Alloy Analyzer with respect to the state-space of the security model and formal top-level specification. For real system verification, Alloy must support a much larger scope. We found that the translation of an existing informal LPSK security policy model to Alloy provided insight for making the model clearer. It is also apparent that Alloy allows for the beginner to formal system verification to quickly climb its learning curve.

Key Words: Software Verification; Principles, Least Privilege; Information Flow Controls, Separation Kernels; Formal Languages

1. Introduction

The current industrial paradigm of software security involves constant patching of reused software. This paradigm is untenable for security and safety critical software that require an extremely high degree of assurance that they will behave as expected. For example, lives depend the proper functioning of medical and aviation software. Lives depend on the proper containment of state secretes held on information systems. The confidentiality and integrity of large amounts of high value information (both personal and financial) in commercial-industrial databases needs better, high assurance protection.

We describe the specification of the formal security policy model and formal top-level specification for the Least Privilege Separation Kernel (LPSK) in Alloy, a relatively new modeling language and analysis tool. In order to determine whether Alloy can improve the efficiency and effectiveness of the verification of secure computer systems, we used it to specify portions of the LPSK formal security policy model and formal top-level specification (FTLS), [Bowen 1996] and utilized the Alloy Analyzer to examine the consistency of the specifications.

This work builds on and extends work of the TCX LPSK. [Irvine 2004] The TCX is a project to develop a Common Criteria Evaluation Assurance Level 7 (EAL7) [ISO/IEC 2004] system that is open to the public in the spirit of the open source movement. The TCX includes a formal specification of an LPSK. The successful dissemination and adoption of the TCX framework by the growing high assurance development community will depend not only on the utility of the framework's toolset, but also on the ease of use of the toolset. It is related to the noninterference work by Goguen and

Mesequer [Goguen 1982] and to work by David Greve, Mark Vanfleet, John McLean, et. al. [Greve 2003] [Heitmeyer 2006] who have provided formal verification of separation kernel security models.

2.1 Access Matrices and Security

Butler Lampson was the first to propose the concept of an access matrix to analyze the information flow of a system [Lampson 1971]. The matrix can then be used to study the flow of information in the system. The Bell and LaPadula model places partially-ordered security labels on system resources [Bell 1973]. The model allows subjects to write up and read down in the hierarchy, but prohibits them from writing down or reading up. The Principle of Least Privilege was first proposed by Saltzer and Schroeder [Saltzer 1975]. As its name suggests, it requires system resources to only have access privileges that are absolutely necessary. This reduces potential vulnerabilities by eliminating unnecessary functionality.

2.2 Separation Kernels

The concept of a separation kernel was first proposed by Rushby [Rushby 1981] and are currently used in military avionics, military communications, and virtual machine monitors (VMM). For example, the F-35 Joint Strike Fighter, F-22, Joint Tactical Radio system, C-17 Globemaster III, and Unmanned Combat Air Vehicle involved separation kernels in their design [Marsh 2002].

A separation kernel partitions system resources and controls the flow of information between the partitions. The separation kernel ensures that no information flows between partitions other than a set of allowed flows, thereby providing the desired security policy.

2.3 Alloy

Alloy is a formal tool set that includes a specification language and a model analyzer. It is well suited for detecting security flaws in software specifications. Via what has been coined the “small scope hypothesis” [Jackson 2006], Alloy’s model analyzer can manage the complexity of software specification. The Alloy syntax defines sets and relations in such a way that it almost appears to be an object oriented programming language. The elegance and simplicity of this syntax allows for a large amount of natural expressivity.

The small scope hypothesis states that if an inconsistency in a model exists there is a high probability that it will present itself within a small scope of the model [Jackson 2006]. For example, in a model of a file system the number of files modeled is the “scope.” If no error or inconsistency is found by the Alloy Analyzer in a scope of ten, there is a very low probability that an error or inconsistency will present itself in a much larger model of a hundred files. The small scope hypothesis allows for the rapid verification of software specifications with much greater detail than other tools. Alloy’s innovation is in connecting the elegant world of specification languages with the power of the model checkers.

3. Formal Security Policy Model

This section presents a basic overview of the TCX LPSK and of Alloy. It then presents an Alloy specification of the FSPM for the TCX LPSK.

3.1 Terminology

In an access matrix a “resource” denotes both subjects and objects which are typically used to indicate which resource is the active entity performing the read and/or write on another passive resource. This model allows one subject to operate on another subject (i.e., one process communicating to another process). The model refers to a subject accessing a resource (which may be another subject).

A partition separates a set of items into distinct groupings. In a separation kernel the resources are the items that are partitioned. The word “partition” is used in the mathematical literature to refer to the set of groupings, and it refers to an individual grouping in the separation kernel literature. This paper avoids the confusion by referring to each individual grouping as a Block. The set of Blocks is a partition of the system resources.

3.2 LPSK Overview

In October 2004, the CISR group at the Naval Postgraduate School published A Least Privilege Model for Static Separation Kernels [Levin 2004]. It is a high-level design document for a LPSK and its security policy. The document contains both a description of the security policy and a FSPM

written in predicate logic that specifies the critical elements of a LPSK, and a security predicate over all possible operations that could be included in a secure system.

The specification developed in this paper contains two essential access matrices that are intended to be orthogonal to each other. One represents the flow of information between partitions which is confined so as to conform to the Bell and LaPadula model. The other represents the Principle of Least Privilege, i.e., a restriction to the set of flows allowed in the former. Once the kernel is up and operational its flow structure will remain static. This means that the flows of information as allowed by the two matrices will not change during run time.

3.3 Alloy Syntax

Alloy uses a mix of relational framework and first order predicate logic. In a join operation, the last item of one tuple is matched with the first item of another. With a brief refresher of the join operation and discrete math the Alloy user will be prepared to construct models in Alloy.

The 'sig' keyword (short for signature) is used to declare sets. The 'extends' keyword will create disjoint subsets much like inheritance in object-oriented programming. The 'abstract' keyword indicates that an item should not be created in the model, but that items extended from it may, much like the virtual keyword in C++ or the interface keyword in Java.

The 'pred' keyword declares a predicate that returns only a true or false value. The 'fun' keyword is used for functions and both functions and predicates may take parameters. The 'fact' keyword can be used anywhere in the specification to place constraints on the model. The 'assert' keyword is used to verify that certain properties hold in the specification. If the properties stated in the 'assert' do not hold the Alloy Analyzer will produce a counter example.

3.4 Alloy FSPM of the TCX LPSK

```
-- the type Resource is defined
sig Resource  {
-- Every Resource belongs to exactly one Block, no more, no less
  master: one Block
}
-- the type Subject is defined
sig Subject extends Resource {}

-- access rights type
abstract sig Mode {}

-- read and write
one sig RD, WT extends Mode {}
```

In Alloy the first set of braces after a signature definition indicates the attributes of items belonging to the set. The 'one' keyword indicates that exactly one item should be generated in the model. In the specification every Resource is assigned exactly one and only one Block. Thus the Block elements partition the Resource elements into equivalence classes as is intended.

```
-- Block  type, the set of blocks defines the partition of the systems
sig Block { }
-- each block has at least one resource - no empty blocks
{
  some r: Resource | r.master = this
}
```

The second set of braces of a type definition is used to indicate Boolean constraints placed on the set. The 'some' keyword acts as an existential quantifier. The 'this' keyword is used in Alloy to refer to the contextual entity being constrained, similar to many object oriented languages. It will not make sense to have an empty Block. Therefore, in the constraint section of the Block type definition, for every Block in the model, there exists at least one resource in the model whose master Block is the Block. The separation kernel is static. Therefore, there is no reassigning of Resource elements to a different Block once the kernel is running.

In Alloy, a 3-tuple or 3 item relation can be used to indicate an access matrix or set of flows. The model of the system will contain three access matrices, bb, sr_flow, and MM.

```
-- Block-To-Block Flow Matrix
sig BB {
  flow:            Block -> Block -> Mode,

-- secondary, derived from the flow
  basic_flow:   Block -> Block,
  FLOWS:       Block -> Block
} {
-- definition of basic flow
  basic_flow =  {
    a, b: Block |
      WT in flow[a][b] or
      RD in flow[b][a]
   }
-- transitive closure of basic_flow
  FLOWS = ^ basic_flow
}
sig System {
  bb:          BB,
  sr_flow:   Subject->Resource->Mode,
  MM:        Subject -> Resource -> Mode
}
```

The '->' characters are used to indicate a tuple. Therefore, sr_flow is the relation Subject->Resource->Mode. The sr_flow and bb matrices are intended to be orthogonal, that is, there may be flows allowed in one that are not allowed in the other. These matrices only determine what flows are allowed, that is they represent the security policy. Technically sr_flow is not required to be a subset of bb. That is, it may allow flows that are prohibited by bb, however, in a secure system those flows would never be realized, because what is allowed is only the intersection of the two matrices.

The MM matrixrepresents the flows that are actualized by the system in operation. MM should be orthogonal to sr_flow and bb. That is sr_flow/bb is the policy of what is allowed, and MM is what the program wants to do. A secure system is a system in which all flows in MM are allowed by both access matrices bb and sr_flow. Conversely, an insecure system is a system that contains at least one flow in MM not allowed by either bb or sr_flow. The security predicate states that a secure system allows only the program actions (MM) that conform to the policy (sr_flow/bb).

Harrison et. al. [Harrison 1976] proved that in the general case the flow of information in an discretionary (i.e., dynamic) access matrix is undecidable. Dorothy Denning used the lattice framework and security labels as constraints on a static access matrix which allow for the determination of information flow a priori [Denning 1976]. However, a partial ordering is sufficient. Denning also demonstrated that any mandatory policy of interest could be mapped to lattice policy.
Thus, a partial ordering ("PO") between the Blocks is imposed, without loss of generality in the access hierarchy.

The predicate PO is defined as a relation over a set that is transitive and antisymmetric. Antisymmetric means there is no circularity in the relation. That is, if information leaves a block, there is no sequence of flows that will lead back to its self.

```
pred PO(b_b: Block->Block){
  all  i,j,k: (b_b.Block + Block.b_b) |

-- reflexive
    (
      i->i in b_b
    ) &&
```

```
-- antisymmetric
   (
     ((i->j in b_b) && (j->i in b_b)) => (i=j)
   ) &&
-- transitive
   (
     ((i->j in b_b)  && (j->k in b_b)) =>
                                   (i->k in b_b)
   )
}
```

The set of all blocks in the block to block relation b_b passed into the predicate can be constructed using the join operation and set addition. Block is the set of all Blocks in the model. Joining Block with b_b like so, b_b.Block, will produce all the blocks of the domain of the b_b relation and Block.b_b will produce the range of the relation. Adding the set of blocks on the lefthand side and the right hand side of the relation will produce the set of all blocks in the relation.

Over time an item of information may no longer be sensitive and so it may need to be downgraded. To do so requires violating the partial ordering. Therefore the notion of a trusted subject is introduced. A trusted subject is a subject (i.e., process) that has undergone rigorous analysis and is trusted not to downgrade information other than the information it is intended to downgrade [Landauer 1989]. Verification of this trust is outside the scope of this paper. However, we desire the model to provide a means of identifying the trusted subjects of the system as well as their flows. This is critical to ensure that they receive the required analysis.

```
sig Trusted_Subject extends Subject {}
```

There may be flows in the system that involve trusted subjects and violate the partial ordering. The challenge here is to identify which subject, trusted or otherwise, caused a flow in bb. Since, the matrices sr_flow and bb are not syntactically linked this is not straight forward.

Instead, a list of trusted subjects is specified, from which it can be determined whether or not the system's flows are partially ordered if the trusted subjects' flows are removed.

The manual translation of the original documentation's [Levin 2004] trusted partial ordering predicate (viz., TPO) in the Alloy specification language produced the following counter example from the Alloy Analyzer: Two block-flows, which individually do not upset the partial ordering, when taken together upset the partial ordering. To resolve this conflict an alternative translation of the TPO was constructed which more clearly matched the intended security policy.

```
-- dom is a library function which returns the domain of the relation
pred TPO(sys: System){

  let Nontrusted_Subs_in_SR =
        dom[sys.sr_flow] - Trusted_Subject,

      Nontrusted_Block_Flow =
      {
        b1, b2: Block , m: Mode |
        (
          some sub: Nontrusted_Subs_in_SR,
                 r:     Resource |
          (
          -- sub is a non-trusted subject in the subject part of sr_flow that when combined with some
          -- resource and the mode of Nontrusted_Block_Flow is also in sr_flow

          (sub -> r -> m) in sys.sr_flow and

          -- and the corresponding blocks of  that subject and resource comprises the blocks of  ---------
          -- Nontrusted_Block_Flow
```

```
            b1 = sub.master and b2 = r.master
            )
          )
      } |
  -- The transitive closure of the intersection of the Nontrusted_Block_Flow and bb.flow with the Mode
removed should be a partial ordering  -- Alloy 4.0 uses brackets [] to pass parameters to a predicate

  PO[ ^((Nontrusted_Block_Flow &
          sys.bb.flow).Mode) ]
}
-- Defines the state of a secure system
pred Secure(sys: System)
{
  TPO[sys]  &&
  all sub:  Subject,
      res:  Resource,
      mod: Mode |
  (
    (sub -> res -> mod) in sys.MM
    =>
    (
      (sub -> res -> mod) in sys.sr_flow &&

      mod in
          sys.bb.flow[sub.master][res.master]
    )
  )
}
```

4. Alloy and Refinement

In the formal methods process the formal top level specification (FTLS) is a high level design specification that represents a refinement of the abstract concepts and properties of the policy model (FSPM) in the form of a general blue print towards implementing the actual system. A major step of the formal methods development process is to prove that the FTLS accurately represents the system in its entirety while preserving the properties of the FSPM.

The separation kernel model that was built in the previous section represents the FSPM of the separation kernel. Alloy allows for a gradual incremental development style. The separation kernel FSPM Alloy specification can be reused unaltered in a new augmented Alloy specification. This augmented specification corresponds to a FTLS. Since the elements and relations of the previous level (FSPM) model are preserved the corresponding properties should also be preserved.

4.1 State Transitions

Demonstrating that the FSPM properties have been preserved in the FTLS presents a challenge. Discovering insecurities in the design and implementation of the operating system is the principle interest. A traditional way to formally represent the security of a system is with a state transition model, where certain inputs on a state define a transition to a new state. By defining a set of possible transitions and an initial state, all reachable states can be modeled. Additionally, by defining what makes a state valid or invalid, any of the reachable states can be checked if they are invalid states, and invalid state-to-state transitions can be identified.

In the FSPM model the signature System and matching predicate Secure naturally correspond to states and a definition of state validity, respectively. An informal system interface document defines the operations that can occur in the TCX LPSK. The FTLS represents these interface operations as commands that effect transitions. With the first state declared secure any of the commands can be checked to see if they lead to an insecure state. If all inputs result in secure states, the system is said to be “Secure”.

This use of automata theory to prove security properties of a system is essentially an inductive proof of properties in a sequence of state transitions. Alloy is not capable of induction or proofs. However, Alloy can be used to find counter examples in a small scope of state transitions. If no counter examples are found the security of the system has been demonstrated for that scope.

```
-- List of commands
abstract sig Command {}
one sig no_op_com,
        read_com,
        write_com
extends Command {}

-- An error for each command
abstract sig Error { }
one sig read_err,
        write_err
extends Error {}

sig Transition
{
  error_message: lone Error,
  last_command:  Command,
  arguments:       set univ,

  sys: System
}
```

The Command signature characterizes an abstract operation and a subset of the actual operations is extended from it. All the system operations have the ability to return error codes, as do the corresponding commands. The Transition signature encapsulates the state and the inputs to the commands. This increases the ability of the model to be visualized in the Alloy Analyzer. Alloy has a built-in type to represent the universal set (univ) which permits any type in the model to belong to the arguments of the transition.

4.2 Two separate models

Due to limits on the size of the model this experimentation divided the FTLS into two distinct but connected sub-systems: One that represents the system initialization sequence and another that represents the system during runtime. This experimentation developed two Alloy specifications to represent the FTLS as distinct sub-systems. The connectedness of the two sub systems is demonstrated by having the last state of the initialization FTLS be the first state of the runtime FTLS. Together they cover the FSPM in its entirety.

The major difference between the two FTLS specification parts is the declaration of the first state of the transition sequence. The initialization model has empty matrices in its first state and the runtime model has established matrices in its first state. Another fact in each model defines a transition in the system as any of the allowed operations occurring.

```
-- First fact in the initialization model, state -- transition starts with empty matrices
fact init_set_up_seq{

  -- first is a function in the sequence library indicating the first item in the sequence
  let t = Seq::set_up_seq/first[] |

    no t.error_message &&
    no_op_com = t.last_command &&
    no t.arguments &&

    no t.sys.resources &&
    no t.sys.bb.flow   &&
    no t.sys.sr_flow   &&
```

```
    no t.sys.MM
}
-- First fact in the runtime model, state -- transitions start with completed matrices that are required
-- not to change
fact init_run_time
{
  let t = Seq::run_time_seq/first[] |

    no t.error_message &&
    no_op_com = t.last_command  &&
    no t.arguments              &&

    Secure[t.sys]
}
```

There are separate predicates for each operation, which allows for either success or failure of the operation. Alloy has a construct for if then else which is used in the transition specification to articulate the system response whether or not the operation was successful.

While the system is being initialized the allowed information flows are configured. Thereafter, in the runtime model, none of the matrices are allowed to change. This restriction is replicated in each operation.

4.3. The Initialization Model

At the beginning of the init specification all the matrices are declared to be empty. The initialization operations that affect the matrices are allowed to occur in any order, which allows the matrices to be gradually filled by one operation at a time.

The signatures for ResourceID and PartitionID refine the spec towards the interface document. Processes are extensions of subjects. A Partition_Flow_Vector, Resource_Vector, and Partition_Resource_Vector were added to represent parameters of the operations. There are seven operations in the initialization model. The set_partition_flows operation is give as an example below.

```
-- Used by set_partition_flows
sig Partition_Flow_Vector {
  v: Partition_ID -> Partition_ID -> Mode
}

pred set_partition_flows(sys, sys': System, pvec: Partition_Flow_Vector)
{
  let new_bb =
  {
    b1, b2: Block, m: Mode |
    (
      (b1.partition_id -> b2.partition_id -> m)
                                    in pvec.v
    )
  } |

  sys'.bb.flow = sys.bb.flow + new_bb &&

  -- only bb matrix is modified
  sys'.sr_flow = sys.sr_flow        &&
  sys'.MM = sys.MM                  &&
  sys'.resources = sys.resources
}
```

4.4. The Runtime Model

The TCX design has various operations dealing essentially with memory movement which is a fundamental security concern. The most basic operation in the interface is reading or writing a byte of

memory. In order to simplify the modeling of reading and writing the signature Memory_Segment and a relation of resources to memory segments were added to the System signature.

Only subjects are allowed to initiate a memory flow. When a subject reads a byte of memory from another resource it is overwriting one of its own bytes of memory (e.g., an internal register) with one of the resources bytes of memory. Likewise, when a subject writes a byte of memory it is overwriting one of the resource's bytes of memory with one of its own.

```
sig Memory_Segment { }

-- This is added to the System signature for
-- modeling reading and writing.
  RM: resources -> Memory_Segment

-- This is a read memory modification
sys'.RM = ((sys.RM)-(sub->Sreg)) +
            (sub->Rreg)

-- This is a write memory modification
sys'.RM = ((sys.RM)-(res->Rreg)) +
            (res->Sreg)
```

In the read op, (sys.RM)-(sub->Sreg) removes the subject/memory segment mapping from the relation, and + (sub->Rreg) adds the new memory segment associated with the resource to the subject relation.

5. Conclusion

This paper has demonstrated that Alloy was effective for representing and reasoning about the security characteristics of a separation kernel. Our conclusion is that Alloy is a useful tool to include in the toolbox of rapid high assurance development. Its low learning curve allows for the beginner to formal methods and high assurance systems to quickly hone their development skills.

The number of operations in the model increased the computational load on the Analyzer. The most complex configuration that could be processed in the initialization model was a scope of three (out of a possible seven). In this respect the current version (4.0) of the Alloy toolset was inadequate. A much larger scope would be necessary to reasonably establish a high assurance of security, but within the limited scope, the security of the model was demonstrated.

5.1 Future Work

In a layered system of specifications, each layer must be shown to preserve the security properties of the layer from which it is refined. Morphisms, from category theory, provide a means for demonstrating the preservation of such properties. [Specware 2004] A property of morphisms is that if the types and operations are mapped from the source to the target layer, then the properties of the source are shown to be preserved in the target, under the conditions of the mapping.

Alloy is suited to demonstrate morphisms by reusing specifications. The nature of extending a signature creates a mapping of each type symbol. However, there remains the effort of. However, Alloy does not yet have built-in constructs, libraries, or templates for proving that the operations of the target specification preserve the properties of the source specification operations. In the current work, a state transition model was used to look for counter examples. We believe that a more direct approach for representing morphisms in Alloy is possible, and further investigation is recommended.

5.2 Acknowledgements

This work has been performed as a master's thesis.[Phelps 2007] The complete source of Alloy models developed for the project is available at <http://alloy.mit.edu/alloy4/>.

This material is based on work supported by the National Science Foundation under Grant No. DUE—0414102 and the Office of Naval Research. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

-- (2004) *ISO/IEC 15408 - Common Criteria for Information Technology Security Evaluation*, Number CCIMB-2004-01-001, International Organization for Standardisation, version 2.0 edition.

-- (2004) "Specware documentation," [online], Kestrel Institute, http://www.specware.org/doc.html

Bell, D. E. and LaPadula, L. J. (1973) *Secure computer systems: Mathematical foundations and model*, Technical Report M74-244, The MITRE Corp., Bedford MA.

Bowen, J. (1996) *Formal Specification and Documentation using Z: A Case Study Approach*, International Thomson Computer Press.

Denning, D. (1976) "A lattice model of secure information flow," *Communications of the ACM*, 19(5):236–243.

Goguen, J. and Meseguer, J. (1982) "Security policies and security models," In *Proc. of 1982 IEEE Symposium on Security and Privacy*, pages 11–20, Oakland, CA, IEEE Computer Society Press.

Greve, D., Wilding, M., and Vanfleet, W. M. (2003) "A separation kernel formal security policy," In *Fourth International Workshop on the ACL2 Theorem Prover and Its Applications*, Boulder, Colorado.

Harrison, M., Ruzzo, W., and Ullman, J. (1976) "Protection in operating systems," *Communications of the ACM*, 19(8):461–471.

Heitmeyer, C. L., Archer, M., Leonard, E. I., and McLean, J. (2006) "Formal specification and verification of data separation in a separation kernel for an embedded system," In *Proc. of the 13th ACM Conference on Computer and Communications Security*, pages 346–355, Alexandria, Virginia.

Irvine, C. E., Levin, T. E., and Nguyen, T. D. (2004) *Trusted computing exemplar 2004 developments*, Technical Report NPS-CS-05-001, Naval Postgraduate School, Monterey, CA.

Jackson, D. (2006) *Software Abstractions*, The MIT Press, Cambridge, MA.

Lampson, B. W. (1971) "Protection," In *5th Princeton Symposium on Information Sciences and Systems,* Princeton University, Reprinted in Operating Systems Review 8,1 January 74.

Landauer, J., Redmond, T., and Benzel, T. (1989) "Formal policies for trusted process," In *Proc. 1989 IEEE Computer Security Foundations Workshop II*, pages 31–40, IEEE Press.

Levin, T. E., Irvine, C., and Nguyen, T. D. (2004) *A least privilege model for static separation kernels*, Technical Report NPS-CS-05-003, Naval Postgraduate School, Monterey, CA.

Marsh, C. (2002) "Real-time operating systems and hardware support," *Avionics Magazine*, June

Phelps, D. A. (2007) *Alloy experiments for a least privilege separation kernel*, Masters thesis, Naval Postgraduate School, Monterey, CA.

Rushby, J. (1981) "Design and verification of secure systems," In *Proceedings of the 8th ACM Symposium on Operating Systems Principles*, pages 12–21.

Saltzer, J. H. and Schroeder, M. D. (1975) "The protection of information in computer systems," *Proceedings of the IEEE*, 63(9):1278–1308.