A Distributed High Assurance Reference Monitor*

Extended Abstract

Ajay Chander[1], Drew Dean[2], and John Mitchell[3]

[1] DoCoMo Communications Laboratories USA, San Jose, CA 95110
[2] Computer Science Laboratory, SRI International, Menlo Park, CA 94025
[3] Computer Science Department, Stanford University, Stanford, CA 94305

Abstract. We present DHARMA, a distributed high assurance reference monitor that is generated mechanically by the formal methods tool PVS from a verified specification of its key algorithms. DHARMA supports policies that allow delegation of access rights, as well as structured, distributed names. To test DHARMA, we use it as the core reference monitor behind a web server that serves files over SSL connections. Our measurements show that formally verified high assurance access control systems are practical.

1 Introduction

One of the major landmarks in computer security research was the definition of the reference monitor as a central location for access control decisions, and the associated notion of a Trusted Computing Base (TCB)[1, 2]. The classic definition of a reference monitor has three properties: (1) It is always invoked (equivalently, is unbypassable). (2) It is tamper-proof. (3) It is verifiable. Much research over the last thirty years has been done on the first [3] and second properties, but comparatively little research has appeared on the verifiability of reference monitors. Given the importance of reference monitor correctness, this state of affairs is highly regrettable. Bugs in the reference monitor have a high probability of directly compromising the security goals of the system, as they are in the control flow path for every access control decision. While a reference monitor for a file system or other resource is often part of the TCB, recent trends in extensible operating kernels [4] and proof-carrying code [5] suggest that it may be possible to move some portions of traditional reference monitor functionality out of the TCB, as long as computations done outside the TCB can be checked within the TCB.

Formal methods can be difficult and time-consuming to use, and are considered impractical for many applications. Because of the difficulty of using formal methods, alternative assurance techniques have been developed [6]. Unfortunately, alternative techniques do not provide the same confidence in correctness as formal, machine-checked, proofs. That formal methods remain the "gold standard" can be seen, for example, in the Common Criteria. The paucity of general purpose operating systems evaluated at EAL7 (under any protection profile) shows the rarity of formally-verified reference monitors.

* This work is supported by DARPA through SPAWAR contract N66001-00-C-8015 and by DOD University Research Initiative (URI) program administered by the Office of Naval Research under Grant N00014-01-1-0795.

In earlier work, we presented the design of a distributed reference monitor, along with its formal verification [7]. In this paper, we use the PVS theorem proving system [8] to turn our formalization into a mechanically-generated *verified* distributed reference monitor *implementation*, that provides access control in credential chain check and search modes. As a result, our implementation provably meets a formally specified definition of security, namely, soundness and completeness *w.r.t.* a semantic access control model. We use the PVS compiler, which takes a substantial subset of the PVS specification language and generates Common Lisp.

To test the performance and functionality of the automatically generated reference monitor, we manually wrote the code to link the reference monitor with cryptographic libraries and administrative interfaces. These components were then integrated into a HTTP server. While building the test system around the verified reference monitor was not conceptually challenging, it was necessary to make a number of modifications and extensions to the tools and utilities we used. In addition to providing a working system that confirms our belief in the potential for verified components of a trusted computing base, we also produced improvements in a standard web server and cryptographic library interface that will be useful for other projects.

2 Distributed Reference Monitor Architecture

The two fundamental concepts in our policy language are access control lists (ACLs) and bounded delegation. Every resource has an associated ACL, which is a list of (subject, right, delegation bound) tuples. An ACL grants some subject a right on some object, and also specifies how far (if at all) that subject can delegate this right by specifying a non-negative integral delegation depth. As shown in previous work, the combination of ACLs and delegation depth provides an attractive compromise between revocability and decentralized control of rights [9]. Moreover, since delegation chains can be viewed as proofs certifying access to a resource, we will use the terms "proof search" and "proof check" for the delegation chain search and chain validity algorithms respectively.

Our policy supports structured distributed names. Names can refer to principals in policies, and are generated by a grammar that is independent of the authorization constructs. In particular we support linked local name spaces as defined in SPKI/SDSI [10, 11], such as `Alice`'s `Bob`'s `Doctor`, which we denote by `Alice.Bob.Doctor`. Names compile to keys, which may appear directly in policies and proofs submitted by clients. With the addition of names, our policy language is expressive enough to subsume the core authorization and naming primitives of SPKI/SDSI.

Shared reference monitor: operational modes Our verified reference monitor DHARMA is parameterized over a finite set of rights whose meanings are not interpreted by the reference monitor. The monitor can be used both in proof search and proof check mode. In proof check mode, the reference monitor ensures that the given evidence is sufficient to allow the requested access. In proof search mode, the reference monitor tries to construct a proof of access from the given policy.

A system deployed in a distributed setting will not know the identities of its users in advance, or have all the evidence required to generate the needed proof of access.

A proof search procedure needs to access policy statements made by several resource owners (or principals). For this, we need guarantees on the validity of local policy statements, when used elsewhere in the system. While some systems use signed certificates to authenticate policy statements, DHARMA supports the authentication of principals through SSL, and allows for the collection of policy statements through secure HTTP connections. Finally, name resolution for linked local names is done through remote procedure calls over SSL. These calls are handled by "nameobjects", which contain local name definitions for the principal owning the nameobject. For example, to resolve the name `Alice.Bob` in the current context, a secure RPC call to the nameobject for `Alice` returns the key for `Alice.Bob`. Nameobjects and the name resolution function are specified within PVS, from which executable versions are automatically generated.

Access graphs: a general semantic model The semantic model of access for this architecture is a forest of weighted, directed graphs, whose nodes are principals and edges capture policy statements (credentials). Each graph is rooted at the associated resource. The weights represent delegation depths. Given an access graph, a subject s can access a resource (o, r) if a path p exists in the graph from the node corresponding to o to the node corresponding to s, labeled with policy statements for the right r, such that the length of each subpath from a node in p down to s is no larger than the weight associated with that node. The weight condition must be satisfied at every node in the path, meaning that if d_i and d_{i+1} are weights of two consecutive nodes, then $d_{i+1} \leq d_i - 1$.

Proof engines for access We implement the access control model using a proof system that is sound and complete for the weighted graph semantics [7]. We *prove* that subject s can access the resource (o, r), by constructing a *proof* of the assertion $\mathsf{Access}(s, o, r, d)$. Proofs start from the access policy and proceed using the proof rules in Table 1, where $\vdash$ denotes provability in the system. Access policies are expressed with similar assertions. The policy assertion $\mathsf{ACL}(A, B, r, d)$ adds A to the ACL for right r on object B, with further delegation ability d. The assertion $\mathsf{Del}(A, B, r, C, d)$ allows A to delegate a right r on object B, to principal C with further delegation depth d.

The inference rules are essentially straightforward. For example, rule (*Delegation*) can be read as: if A can access right r on resource B with the ability to delegate it further at most $(d+1)$ times, and delegates the same right to C with a delegation bound of d, then C can access right r on B and can delegate it further up to a depth of d. The last two rules are needed for completeness: if you have the ability to delegate a right n times, you can also delegate that right k times, for any $k < n$.

(*RootACL*)	$\mathsf{ACL}(A, B, r, d) \vdash \mathsf{Access}(A, B, r, d)$
(*Delegation*)	$\mathsf{Access}(A, B, r, d+1), \mathsf{Del}(A, B, r, C, d) \vdash \mathsf{Access}(C, B, r, d)$
(*Ord1*)	$\mathsf{Access}(A, B, r, d+1) \vdash \mathsf{Access}(A, B, r, d)$
(*Ord2*)	$\mathsf{Del}(A, B, r, C, d+1) \vdash \mathsf{Del}(A, B, r, C, d)$

Table 1. Logical rules for access judgments

Structured, distributed names Our policy language also provides support for linked local names, in the manner of SPKI/SDSI. In earlier work, we showed that soundness and

completeness for an access control system that includes naming follows logically from a commutativity theorem [7] between authorization and naming constructs. Linked local names are a special case of a general approach to composition, captured by the commutativity theorem, that supports other policy constructs such as groups, roles, etc.

Our PVS effort includes a specification of linked local names and name resolution, using which we formally prove the commutativity theorem. This allowed us to formally demonstrate the soundness and completeness of the name-extended proof system, increasing confidence in an implementation that uses them directly. We note also that as a result of the commutativity theorem, policies that contain names can first be compiled into a policy containing only keys, to which the rules in Table 1 can then be applied.

3 Formal Verification of Reference Monitor using PVS

The first set of proofs within PVS ensure the soundness and completeness of our proof rules *w.r.t.* the semantic model of access graphs. These theorems relate *provability* within the proof system, denoted by $\vdash$, to *truth* in the access graph semantic model, denoted by $\models$. Formally, we capture the set of local policies relevant to the access control decision by the *world state*. We denote the minimal access graph model corresponding to a world state w by $M(w)$. Given a world state w, access graph G, subject s, object/right pair (o, r), we use the following notation:

$$w \vdash s \rightarrow (o, r) \equiv \text{there exists a proof of } s \text{ accessing } (o, r) \text{ using the logical rules}$$
$$w \models s \rightarrow (o, r) \equiv s \text{ can access } (o, r) \text{ in the access graph model}$$
$$G \models w \equiv \text{all accesses provable in } w \text{ are also true of the access graph } G$$

Given this, soundness and completeness can be stated as follows:

$$\text{(S)} \quad w \vdash s \rightarrow (o, r) \Rightarrow \forall G.(G \models w) \Rightarrow G \models s \rightarrow (o, r)$$
$$\text{(C)} \quad \forall G.[G \models w \Rightarrow G \models s \rightarrow (o, r)] \Rightarrow w \vdash s \rightarrow (o, r)$$

In other words, soundness states that for any given world state w, subject s, and resource (o, r), if one can *prove* that s can access the right r on object o within the proof system, then in all semantic models (access graphs) that satisfy the world state, subject s can access the resource (o, r). Completeness can be understood as follows: let us assume that for all access graphs G, whenever G satisfies the given world state w, subject s can access resource (o, r) in the access graph. Then there exists a *proof* of $s \rightarrow (o, r)$ within the proof system. These formally verified properties increase confidence in any implementation of a reference monitor that uses these rules.

Correctness of algorithms We obtain a verified implementation of our verified model by specifying the proof search and check algorithms in PVS, from which PVS extracts equivalent Lisp code. Thus, we avoid the implementation errors often associated with hand-coded implementations of verified models (e.g., [12]). Either algorithm can be invoked by the server or the client. Ideally, the client would use proof search (verified or otherwise) to ensure that it can access the desired resource, and the server would validate this proof using the verified proof check algorithm. This setup provides the

same efficiency benefits as proof-carrying code [13, 14]; instead of executable code sent by the client, we have a verifiable proof corresponding to an access request.

Soundness for the proof check (equivalently, chain validation) algorithm ensures that the logical proof of an allowed access will correspond to a valid path in the access graph. Completeness for this algorithm ensures that whenever there exists a valid path in the access graph, the proof check algorithm will validate the corresponding logical proof. Formally, given an access graph G that is a model of the world state w, $(G \models w)$, if there exists a path t of G that is equivalent to a logical proof $p \equiv w \vdash s \rightarrow (o, r)$, then soundness and completeness for this algorithm can be stated as:

(PC_S) $\texttt{check}(w, p, s, o, r) \Rightarrow \forall G.(G \models w) \Rightarrow \exists t.G \models_t s \rightarrow (o, r)$
(PC_C) $\forall G.(G \models w) \Rightarrow \exists t.G \models_t s \rightarrow (o, r) \Rightarrow \exists p.\texttt{check}(w, p, s, o, r)$

where $G \models_t s \rightarrow (o, r)$ denotes that s can access the resource (o, r) in the access graph G via the well-defined path t.

The proof search algorithm returns a proof-tree that can be checked against Table 1, thereby ensuring soundness. Completeness of this algorithm shows that if the procedure fails, then no path exists in the access graph for the access request. Using the same terminology as above, soundness and completeness for this algorithm can be stated as:

(PS_S) $\texttt{search}(w, s, o, r) = \texttt{witness}(p) \Rightarrow \forall G.(G \models w) \Rightarrow \exists t.G \models_t s \rightarrow (o, r)$
(PS_C) $\texttt{search}(w, s, o, r) = \texttt{failed} \Rightarrow \forall t.\neg(G(w) \models_t s \rightarrow (o, r))$

Soundness and completeness imply that the PVS algorithms will match our expectation of when an access should be allowed. We refer the reader to [7] for more details.

Code extraction from verified specification A large subset of the PVS specification language can be seen as a functional language, supporting common data types such as lists and records, higher order functions, parametrization, and additional features such as dependent types, that greatly simplify proof construction. From version 2.3, PVS can generate Lisp code from the corresponding functional specification, which is guaranteed to meet the specifications if all proof obligations have been discharged. We use this capability to generate implementations of the proof check and proof search algorithms in DHARMA from their verified PVS specification. In addition to soundness and completeness, type correctness condition proofs ensure the termination of the recursively defined proof check and search functions.

4 A Policy-driven Web-based File Server

Our PVS specification allowed us to automatically generate the verified reference monitor DHARMA. To study its applicability, we instrumented a HTTP server with DHARMA. The enhanced server allows for editing of local policies, remote management of policy databases, and access control via proof check and proof search.

Infrastructure We chose Franz's Allegro Common Lisp and AllegroServe HTTP server [15] as our research infrastructure. Due to AllegroServe's provision of a hook for the authorization of each HTTP request, integrating DHARMA was easy to do. We enhanced the Allegro Common Lisp OpenSSL binding to support client authentication.

Enhancing the server with DHARMA We use AllegroServe's authorizer framework to integrate DHARMA with AllegroServe. Files are published in AllegroServe by associating one or more entities with the file. An entity is a Common Lisp object which handles *all* requests for the file, and can have a number of attributes related to pre-loading, caching, the time-out for serving the file, etc. Each entity also has a hook for an authorizer object, which can be any Lisp function. When a client connects to the port on which AllegroServe is listening, AllegroServe passes that connected socket to a free worker thread that has access to the local policy. The worker thread locates the entity that is associated with the request; if an entity is found and has an authorizer, it calls the authorizer function to decide if this client should be allowed to access the selected file.

We instantiated the authorizer functions for the entities of each published resource with our verified DHARMA proof search and proof check functions. Based on whether or not the client has presented a proof, the appropriate function (proof check and proof search, respectively) is invoked. The client's public key, name, and URI of the requested resource are passed to the function. We also need to authenticate the client, and perform distributed name resolution when the presented proof contains linked local names.

Leveraging AllegroServe's request protocol in this manner allows us to express the security guarantees provided by the web server in terms of the security properties of the reference monitor DHARMA. The fact that AllegroServe is programmed in a functional style in Lisp allowed us to quickly audit its source code and assure ourselves that DHARMA is invoked before *any* resource is served by the enhanced web-server. This source code audit is, of course, far short of a formal proof of security for the DHARMA-enhanced AllegroServe. Such a proof would be important for production use of a high-assurance system; however, it is outside the scope of this work. We also note that the context in which AllegroServe operates may be vulnerable to other attacks; for example, an attacker may gain root access to the machine and replace the DHARMA-enhanced web server with an insecure server. The research described here is focused on providing assurance for a critical software component, the reference monitor, even as we remain cognizant of other attacks on the larger system containing DHARMA.

Managing local policy databases A resource owner may not have all the certificates required to decide on an access request. We provide the ability to add delegation policy statements to a *local* policy database after authenticating the signer of the statement against the subjects mentioned in the body of the statement. For example, to add the statement which says that A adds B to its ACL for a particular right, it must be signed by A. Checking signatures in this manner is equivalent to implementing the logical operator "says" used in other treatments [16] of distributed authorization. DHARMA supports this by using a POST method over a SSL-secured HTTP connection. In addition to such peer exchange of credentials, a resource host can edit its own local policy as reflected in the files and their authorizers published through AllegroServe.

Distributed name resolution As mentioned earlier, our policy language supports linked local names. Local name resolution is provided via nameobjects that are generated automatically from their PVS specification. Since PVS doesn't have native support for remote procedure calls, we provided this by means of a *semantic attachment* to the PVS specification. A semantic attachment is an external piece of code that is declared but not

defined within a PVS specification. The hand-written implementation of the semantic attachment is linked with the PVS code generated for the nameobjects at the different peer sites. This hand-written code was a few lines that set up Allegro Lisp RPC sockets, connecting the PVS-generated local name resolution function on one nameobject to the same function running on another nameobject. Our handling of nested RPC calls, such as those involved in resolving the name $A.B.C$, is iterative; this allows us to detect if a specific nameobject is unresponsive, and provides a way of caching keys corresponding to intermediate names such as $A.B$.

5 Measurements

To measure the cost of adding DHARMA to a HTTP server, we performed a series of measurements. First, to get a baseline, we compared the performance of AllegroServe to Apache, both with and without SSL support. Then, we measured the cost of adding either proof check or proof search while handling file requests. Finally, we measured the cost of resolving linked local names in a distributed setting, as part of checking the proof presented with the access request. This series of measurements allowed us to identify the costs of individual additions to the original web server, and point to performance limits as well as areas where optimization would produce the most benefit.

All measurements were performed on a dual Xeon machine, with 2GB of RAM, running at 2.8GHz with hyperthreading enabled. The machine was running Redhat Linux 7.3, kernel version 2.4.18. We used AllegroServe version 1.2.26, and Apache version 2.0.47 with the `mod_ssl` module. Measurements used version 0.8 of the httperf tool [17] that was slightly modified to support sending client certificates as per [18].

Distributed name resolution We found AllegroServe to be a well-architected, high-performance web-server that performed well on our tests. We made three representative measurements, for file sizes of 8Kbytes (text page), 100Kbytes (image), and 1byte (minimal HTTP payload processing time). Table 2 shows the throughput for the different servers, for serving files of different sizes over HTTP and HTTPS connections.

To estimate the relative cost of adding a Lisp reference monitor to AllegroServe with a C-based one, we compared the performance for DHARMA-enhanced AllegroServe that does proof search over a policy database containing only ACLs, and ACL-based access control in Apache using SSLRequire directives. For a set of 100 credentials, we measured a 2.4% performance hit for AllegroServe, and 3.8% for Apache. These numbers suggest that our measurements in the following sections reflect the intrinsic cost of the algorithms, rather than the choice of Lisp for the reference monitor.

Adding proof check Figure 1 measures the cost of verified proof check in DHARMA-enhanced AllegroServe. This is a crucial measurement, as it is a fundamental guarantee of assurance. A proof presented by a client will have to be validated by the verified proof check code, irrespective of the algorithm used for proof search.

For our measurements, we varied the number of principals and the number of credentials (policy statements) in the generated policy database. The proof check algorithm checks every step of the presented proof against this database, in addition to checking

		Size of file		
Protocol	Server	1B	8KB	100KB
HTTP	AllegroServe	613	430	90
	Apache	431	417	104
HTTPS	AllegroServe	128	101	38
	Apache	101	95	54
ACL	AllegroServe	125	99	38
	Apache	96	92	52

Table 2. Baseline performance (requests served / second) for AllegroServe and Apache

whether each pair of successive steps represents a valid application of one of the rules in Table 1. The policy database is represented as lists within PVS; we can see the effect of searching through longer lists as the number of credentials increases.

Our measurements show that verified proof check using DHARMA scales very well. For access control scenarios in corporations with up to 5000 credentials, the expected performance hit is only 5% for an average file size of 8KB. If the average resource size is higher, then even more credentials can be supported with the same penalty. An optimization is to specify the policy database as a hashtable within PVS, which would allow for nearly constant proof check time.

Adding proof search Proof search is the most expensive operation in DHARMA. Figure 1 measures the times for verified proof search for different combinations of policy and file sizes. Our proof search algorithm is equivalent to a breadth first search on the size of the access graph, and thus is linear in the number of credentials which may grow as $\Theta(N^3)$ in an access graph with N principals. In our measurements for both proof search and proof check, we generated a policy database by limiting the degree of nodes in the access graph to between 3 and 20, depending on the total number of principals. The degree estimates how many other principals a given principal would add to an ACL for a resource controlled by itself, and to how many principals it would pass on a delegated access. In other words, a principal may have upto 20 other principals on its ACL, *and* pass on rights to 20 principals for resources it doesn't own.

In contrast with proof check, proof search over a database of 5000 credentials has an expected performance penalty of about 40%. The contrast is more pronounced with higher policy sizes; 15000 credentials corresponds to only a 17% performance penalty for proof check in DHARMA, but 110% for proof search. These numbers for verified proof search correspond to the best known algorithm for proof search in policies without extended names [19], and thus reflect the complexity of the underlying algorithm, as opposed to the performance of the generated code.

Adding name resolution via RPC Finally, we considered the cost of resolving linked local names through remote procedure call, when they appear in the proof presented by the client. Setting up an SSL-secured channel over RPC incurred significant cost; for scenarios in which the presented proof contained *only one* linked local name, where each name resolution request is handled by a different server, DHARMA was only able to handle 11 requests per second initiated by an individual client. The other extreme, when all name resolution requests are handled by a single server using the same secured

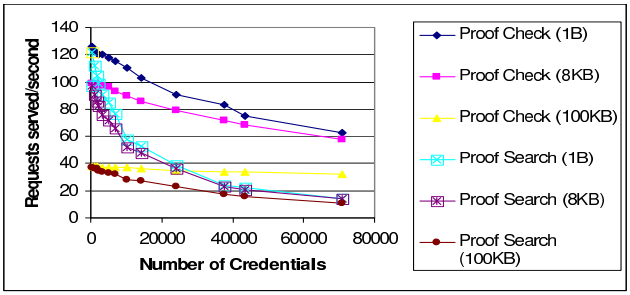

Fig. 1. Requests per second for DHARMA-enhanced AllegroServe

RPC channel, results in DHARMA being able to handle 80 requests per second. Recall that in the absence of names, DHARMA can handle 125 requests per second.

Our measurements show that name resolution during proof check imposes a significant penalty, in the absence of a central name server. Thus, when using DHARMA with a different proof search algorithm that can handle names in credentials, we suggest compiling the final proof containing names to one that has only keys. This compiled proof can then be validated against a database that contains the equivalent name-resolved credentials, without compromising security.

Discussion Our experience with DHARMA shows that the TCB can be augmented with a verified reference monitor that provides proof check functionality, without significant penalties. In addition to intranet scenarios discussed earlier, this benefit scales well to wide-area services such as electronic subscription-based publishing as only the sizes of local policy tables *in the presented proof* affect the cost of proof check. However, adding proof search to the TCB has significant cost, as a result of the complexity of the proof search algorithm. For the case of credentials that contain names, the optimal search algorithms are cubic in the number of *credentials* [19, 20], which would further worsen performance if included in the TCB. If the result returned by the search algorithm is validated by the verified proof checker, then the search algorithm itself doesn't provide any additional assurance. Given the lower bounds on these search algorithms, we suggest implementing the search algorithm as efficiently as possible outside the TCB, and validating the result by the verified proof checker inside the TCB.

We notice from the name resolution measurements that there is a significant penalty for making remote calls for name resolution. A system with greater delegation depth in policies may have longer proofs of access on the average, which would multiplicatively increase this name resolution cost. Since search algorithms with named credentials are already more expensive, we suggest providing name-resolved proofs to the server while accessing a resource. One option for the infrastructure to support this is for the principals in the system to attach name resolution credentials together with the authorization credentials as part of the proof of access presented with the request. SPKI/SDSI-style name credentials that attest to the (local name, key value) binding in a nameobject, can be directly used for this purpose. Servers may also advertise "recency requirements" on

credentials that they are willing to accept, which can guide the rate at which principals refresh their credentials. Another observation is that performance is much improved by having a single name server, since the cost of secure channel setup is shared amongst several name resolution requests. This points to an architecture with a few back-end "resolution servers" available to all DHARMA-enhanced web servers.

6 Related Work

Early work in formal methods, such as the Stanford Pascal Verifier [21], and Ina Jo [22] employed separate specification and implementation languages. Gypsy [23] was an early system that provided separate specification and implementation languages, and a compiler from its implementation language to executable code. Approaches such as Sannella's Extended ML system [24] integrate axiomatic specifications with Standard ML programs. The Design/CPN tool has been used to compile a colored Petri net specification of a centralized access control system into ML code [25], but lacks full verification support. Some recent work in model checking system code [26, 27] has approached the problem by extracting specifications from actual code.

There is a rich history of security kernel designs from the late 1970s and early 1980s, including the provably secure operating system PSOS [28] and the kernelized secure operating system (KSOS), which was implemented in Modula-2,and specified *separately* as a finite state machine. The PSOS-inspired KSOS went through several future generations, and begot SCOMP [29] (KSOS-6), KSOS-11 [30], VIKING [31], and the Secure Ada Target [32]. Gutmann presents the verification of a reference monitor based kernel [33] using assertion-based testing tools and hierarchical design methods; in comparison, we provide provable compliance to a formal definition of security.

Several access control systems with support for decentralized policies have been built, though none provide strong assurance guarantees or report performance studies. The two best known are PolicyMaker [34] and KeyNote [35], with SPKI/SDSI [11, 10] being the most analyzed. Clarke [36] integrated an existing C language implementation of SPKI/SDSI into the Apache HTTP server, but didn't provide actual system measurements. We note that a very early version of linked local names appears in Reed's Ph.D. thesis [37]; his implementation is very similar to our implementation of nameobjects and name resolution (see Section 4.) Proof-carrying authentication [14, 38] provides a general-purpose higher-order *undecidable* logic for distributed access control, and is implemented in the proof-checker Twelf[39], but is not formally verified.

7 Conclusions

Building on our previous work on the formal analysis of access control policies, we used PVS to generate a *verified* reference monitor. We then *integrated* this monitor into a web server that checks SSL certificates, resolves names against a set of hierarchical local namespaces, and determines whether to allow an access request. The timing results in Section 5 show that the verified system has acceptable performance in proof check mode when certificate chains are supplied as input. For greater efficiency, we can replace the verified proof search function with hand-coded C, since the output of the

search process is a certificate chain that is checked by our verified reference monitor. The results of Section 5 suggest that an architecture with a few public name resolution servers mitigates the cost of setting up per-name secure connections in a distributed access control system, making the performance of the verified chain validation acceptable in the presence of linked local names.

References

1. Lampson, B.: Protection. In: Proc. of the 5th Annual Princeton Conference on Information Sciences and Systems, Princeton University (1971) 437–443
2. U.S. D.O.D.: Trusted Computer System Evaluation Criteria ('Orange Book'). (1983)
3. Rushby, J.: Noninterference, transitivity, and channel-control policies. Technical Report SRI-CSL-92-02, SRI International (1992)
4. Engler, D.R., Kaashoek, M.F., O'Toole Jr., J.: Exokernel: an operating system architecture for application-level resource management. In: Proc. of the 15th ACM Symposium on Operating Systems Principles (SOSP '95), Copper Mountain, CO (1995) 251–266
5. Necula, G.C., Lee, P.: Safe kernel extensions without run-time checking. In USENIX, ed.: 2nd Symposium on Operating Systems Design and Implementation (OSDI '96), October 28–31, 1996. Seattle, WA, Berkeley, CA, USA, USENIX (1996) 229–243
6. Vecellio, G., Thomas, W.: Issues in the assurance of component-based software. In: Proc. of the 2000 Workshop on Continuing Collaborations for Successful COTS Development (ICSE2000), (Limerick, Ireland) `wwwsel.iit.nrc.ca/projects/cots/icse2000wkshp/Papers/14.pdf`.
7. Chander, A., Dean, D., Mitchell, J.C.: Reconstructing trust management. Journal of Computer Security **12** (2004) 131–164
8. Owre, S., Shankar, N., Rushby, J.M., Stringer-Calvert, D.W.J.: PVS Language Reference, Version 2.3. SRI International. (1999) `http://pvs.csl.sri.com/`.
9. Chander, A., Dean, D., Mitchell, J.: A state-transition model of trust management and access control. In: Proc. of the 14th IEEE Computer Security Foundations Workshop. (2001) 27–43
10. Ellison, C., Frantz, B., Lampson, B., Rivest, R., Thomas, B., Ylonen, T.: SPKI certificate theory. RFC 2693 (1999)
11. Rivest, R., Lampson, B.: SDSI–A Simple Distributed Security Infrastructure. `http://theory.lcs.mit.edu/~rivest/sdsi11.html` (1996)
12. Anderson, J.P.: Computer security technology planning study. Technical Report ESD-TR-73-51, U.S. Air Force, Electronic Systems Division, Deputy for Command and Management Systems, HQ Electronic Systems Division (AFSC), L. G. Hanscom Field, Bedford, MA 01730 USA (1972) Volume 2, pages 58–69.
13. Necula, G.C.: Proof-carrying code. In: Conference Record of POPL '97: The 24th ACM Symposium on Principles of Programming Languages, Paris, France (1997) 106–119
14. Appel, A.W., Felten, E.W.: Proof-carrying authentication. In: ACM Conference on Computer and Communications Security. (1999) 52–62
15. John Foderaro: AllegroServe – A Web Application Server (Franz. Inc.). (`http://allegroserve.sourceforge.net/`)
16. Abadi, M., Burrows, M., Lampson, B., Plotkin, G.: A calculus for access control in distributed systems. TOPLAS **15** (1993) 706–734
17. Mosberger, D., Jin, T.: httperf: A tool for measuring web server performance. In: First Workshop on Internet Server Performance, ACM (1998) 59–67
18. Rescorla, E.: An introduction to OpenSSL programming, Part I. Originally appeared in the Linux Journal; `http://www.rtfm.com/openssl-examples/part1.pdf` (2001)

19. Clarke, D., Elien, J.E., Ellison, C., Fredette, M., Morcos, A., R.L.Rivest: Certificate chain discovery in SPKI/SDSI. Journal of Computer Security **9** (2001) 285–322
20. Li, N., Winsborough, W., Mitchell, J.C.: Distributed credential chain discovery in trust management. Journal of Computer Security **11** (2003) 35–86
21. Luckham, D.C., German, S.M., von Henke, F.W., Karp, R.A., Milne, P.W., Oppen, D.C., Polak, W., Scherlis, W.L.: Stanford Pascal Verifier user manual. CSD Report STAN-CS-79-731, Stanford University, Stanford, CA (1979)
22. Locasso, R., Scheid, J., Schorre, D.V., Eggert, P.R.: The Ina Jo Specification Language Reference Manual. System Development Corporation, Santa Monica, CA. (1980)
23. Good, D.I., London, R.L., Bledsoe, W.W.: An interactive program verification system. IEEE Transactions on Software Engineering **1** (1975) 59–67
24. Karhs, S., Sannella, D., Tarlecki, A.: The definition of Extended ML: a gentle introduction. Theoretical Computer Science **173** (1997) 445–484
25. Mortensen, K.H.: Automatic code generation method based on coloured petri net models applied on an access control system. In: Lecture Notes in Computer Science: 21st International Conference on Application and Theory of Petri Nets (ICATPN 2000), Aarhus, Denmark, June 2000. Volume 1825., Springer-Verlag (2000) 367–386
26. Chen, H., Wagner, D.: MOPS: An infrastructure for examining security properties of software. In: Proc. of the 9th ACM Conference on Computer and Communication Security, Washington D.C. (2002) 235–244
27. Chen, H., Wagner, D., Dean, D.: Setuid demystified. In: Proc. of the 11th USENIX Security Symposium, San Francisco, CA (2002) 171–190
28. Neumann, P.G., Boyer, R.S., Feiertag, R.J., Levitt, K.N., Robinson, L.: A provably secure operating system: The system, its applications, and proofs. Technical Report CSL-116, 2nd Ed., SRI International (1980)
29. Fraim, L.J.: SCOMP: A solution to the multilevel security problem. IEEE Computer **16** (1983) 26–34
30. Berson, T., Barksdale, G.: KSOS: Development methodology for a secure operating system. In: National Computer Conference, AFIPS Conference Proc. (1979) 365–371 Vol. 48.
31. Hartman, B.: A Gypsy-based kernel. In: Proc. of the 1984 IEEE Symposium on Security and Privacy, Oakland, CA (1984) 219–225
32. Boebert, W.E., Kain, R.Y., Young, W., Hansohn, S.: Secure Ada target: Issues, system design, and verification. In: Proc. of the 1985 IEEE Symposium on Security and Privacy, Oakland, CA (1985) 176–190
33. Gutmann, P.: The Design and Verification of a Cryptographic Security Architecture. PhD thesis, Department of Computer Science, University of Auckland (2000)
34. Blaze, M., Feigenbaum, J., Lacy, J.: Decentralized trust management. In: Proc. of the 1996 IEEE Symposium on Research in Security and Privacy, Oakland, CA (1996) 164–173
35. Blaze, M., Feigenbaum, J., Keromytis, A.D.: KeyNote: Trust management for public-key infrastructures. In: Proc. of the 1998 Security Protocols Workshop. Volume 1550 of Lecture Notes in Computer Science. (1999) 59–63
36. Clarke, D.E.: SPKI/SDSI http server / certificate chain discovery in SPKI/SDSI. Master's thesis, Massachusetts Institute of Technology (2001)
37. Reed, D.P.: Naming and synchronization in a decentralized computer system. Technical Report MIT/LCS/TR-205, Massachusetts Institute of Technology (1978) Also Ph.D. thesis.
38. Bauer, L., Schneider, M.A., Felten, E.W.: A general and flexible access-control system for the web. In: Proc. of the 11th USENIX Security Symposium, San Francisco, CA (2002)
39. Pfenning, F., Schürmann, C.: System description: Twelf — A meta-logical framework for deductive systems. In: Proc. of the 16th International Conference on Automated Deduction (CADE-16), Trento, Italy, Springer-Verlag LNAI 1632 (1999) 202–206