Confidence Domains for Distributed Systems

Winfried E. Kühnhauser
German National Research Centre For Computer Science (GMD)
P.O. Box 1316 D-53731 Sankt Augustin
Germany
e-mail: winfried.kuehnhauser@gmd.de

Abstract

The paper addresses the problem of trust in large computer networks that connect several independent organizations. While in such networks it is politically difficult to agree upon one single common point of trust and one single global network security policy, few networks exist in which no system trusts any other system. Thus we observe that systems in a network form clusters, based on the sharing of a common point of trust or a common security policy.

One of the major assumptions in this paper is that trust cannot be achieved on a simple technical or mechanical level alone. We introduce confidence domains as an approach to describe human belief in the trustworthiness of systems and thus make this knowledge available to the system’s security components.

The paper describes the concept of confidence domains together with the paradigms used to define and establish them. It gives examples how confidence domains can be exploited as a foundation for security policies. The paper also describes mechanisms needed to enforce confidence domains in an open network and concludes with a detailed description of an implementation for the BirliX Security Architecture.

Introduction

In large computer networks with distributed administration and systems belonging to several independent organizations, both political and technical reasons make global trust difficult to achieve.

Politically, it seems difficult to convince people with high security requirements that they may trust any alien home computer as much as their own system. Technically, although there are approaches that allow mutual system authentication [CCI88] and that allow proofs that neither the operating system nor a system’s hardware have been tampered with [GGKL89, Gro91], these proofs still assume that
- the system hardware is capable of storing and hiding a secret, and that any attempt to tamper with the hardware results in the destruction of that secret
- removable storage media that hold long-term data are encrypted in a way that makes it virtually impossible to read or modify the contents.

Since these assumptions do not hold with most of today’s systems and in general are difficult and expensive to achieve, current hardware and encryption technology cannot establish trust in a remote system without at the same time trusting the human beings that control the system. Just being able to authenticate a system does not necessarily imply that we also trust it. On the contrary, because of the reasons mentioned above, trust in a system always presupposes trust in the people controlling the system, and thus must be explicitly dealt with as a part of a security policy. Implementing security policies in a large open network thus requires a way to formally specify human assumptions of the trustworthiness of systems in a language that can be interpreted by the systems themselves.

This paper introduces confidence domains as an approach to reasoning about trustworthiness of systems. A confidence domain is a cluster of systems in a computer network that from some point of view (e.g., a security policy) share the same level of trust.

In the real world, confidence domains have existed for a long time and in many variations. “All people working for this organization may enter the building without being challenged” is a wide-spread rule putting all employees of a company in a confidence domain of humans that are generally assumed to have legal access to a building. In computer networks, “all systems in building C” is a domain based on the physical location of machines, whereas “all systems belonging to company X” may also include mobile computers. Finally, “all systems belonging to project ALPHA” defines a domain that may cover both physical locations and administrative units, as might arise in a multinational project involving several cooperating institutions.

These examples have in common that

- high level security policies use confidence domains as a paradigm to distinguish between trustworthy and non-trustworthy systems, and to apply corresponding regulations, e.g., for within-domain versus inter-domain communication or cooperation
- mechanisms on the system level must exist to enforce confidence domains in open networks, e.g., by mutual system authentication or message flow control.

Confidence domains as a notion to describe human knowledge about the trustworthiness of systems were first discussed in [HKK93] in the context of the BirliX Security
Architecture. This paper summarizes and generalizes the model described in [HKK93] and focuses on the formal definition, implementation, experience, and exploitation.

The next section describes the confidence domain model by relating it to a small set of scenarios, each one having its own simple security policy. Section three focuses on the system-level mechanisms needed to enforce the model in a large open computer network. An implementation of the mechanisms within the scope of the Birlix Security Architecture [HKK93] is discussed in detail.

The Model

We discuss our model of confidence domains in the context of a distributed system consisting of autonomous and mutually suspicious computer systems. We do not assume any central management, and we support the view that new security concepts must have minimum need for real time interaction between users and remote security services.

The entities in the system are users, computer systems, and system objects (like files and processes). Entities have names, and entities are either able to authenticate themselves or can be authenticated via certificates obtained from an authentication service like X.509 [CCI88].

Any communication between entities is controlled by local and autonomous reference monitors. More precise, an object-based communication scheme is assumed that precisely restricts communication to the invocation of methods; reference monitors control both the invocation of methods at the caller and the acceptance of a call at the callee.

This section first describes the confidence domain model based on three widespread scenarios, each one having its own simple security policy: mobile computing, lab computers, and contract negotiations. The requirements for each individual security policy are the input for a formal definition in terms of elementary set theory.

Szenarios

Mobile Computing: A laptop is a self-contained, mobile computer that usually has a single owner. During travels, the laptop is frequently connected to a public network, e.g. for moving documents back to the owner’s company.

The documents on the laptop have very individual security requirements. Personal documents such as notebooks, have privacy as the ultimate concern. Other documents, like letters or contracts, are less private; to print them on company paper they have to be moved to the secretary’s workstation. Nevertheless, it is possible that no other
system in the company is trusted to keep these letters or contracts confidential. Other documents such as memos for all employees of the company may exist that are even less private.

The above scenario can be described by three confidence domains, one containing just the laptop, the second containing the laptop and the secretary’s workstation, and the third containing the laptop and all systems belonging to the company.

Our small privacy security policy can be enforced if

- confidence domains can be defined individually for each object, similar to access control lists
- the secretary’s and company’s systems can be reliably authenticated without relying on the trustworthiness of public routers and communication media
- access is denied to a caller if the accessing system is either not properly authenticated or is not in the object’s confidence domain

Figure 1: Laptop Objects and Confidence Domains

In this example the confidence domains are strictly hierarchical, ranging from the most private personal notebook via the secret contracts down to the just company-confidential memos; the first set is included in the second set which again is a subset of the third set. Nevertheless, a simple extension of the example shows that confidence domains are not partially ordered in general: perhaps the owner of the laptop is just negotiating with a potential new employer, and he also keeps a contract proposal from the new employer on his laptop. An adequate model for the privacy requirements of the contract proposal is a confidence domain that contains the laptop and some system
of the future employer. This confidence domain is not a subset of any of the confidence domains defined above, nor does it contain them.

By not assuming a partial order relation between confidence domains, we avoided unnecessarily restricting the power of the model.

Lab Computers: Lab computers are stationary systems which are less private than laptops. A lab computer is used by many people. Examples are machines in a public workstation pool of a university, or measurement/control computers in a chemistry analysis laboratory. A lab computer is used by everyone who currently needs its services; therefore it is usually not considered to be very trustworthy.

Lab computers normally are connected to server pools that are - in contrast to the lab computer - trusted to a higher degree. The typical way of working in such a scenario is that any object of value is maintained by the servers. When a user logs into a workstation, he may migrate (or copy) his valued objects from the servers to the workstation, because he trusts it as long as he is logged in and controls the workstation. No object of value stays on the lab computer when the user logs out.

This scenario can easily be described in terms of confidence domains. Before the user logs in, the confidence domains of his objects contain the server machines only. After a mutual authentication between user and lab workstation, the user relaxes the confinement of his objects to the servers by adding his current login-machine to the confidence domains of his objects, which allows him to move his objects to the workstation. On logout, the confidence domains of his objects are reset to the original which forces the objects back to the server pool.

This example shows that

- confidence domains control the location of an objects
- confidence domains associated to objects may change dynamically

A major security threat in this scenario is the operating system on the lab workstation itself. As workstations are generally under full control of their current user, any user may also install a modified operating system that pretends to be the real thing and actually is a major threat to security. Thus we need two basic mechanisms for the implementation of confidence domains:

- a user must be able to authenticate the lab workstation system to make sure no other user has manipulated the machine and installed a malicious operating system
- the servers must be sure that the user has logged into the workstation and the workstation now is working on his behalf. This trust can only be established by a mechanism that allows the servers to authenticate the workstation’s system [GGKL89, Gro91].
Contract Negotiations: When two parties agree on a business transaction they usually have conflicting goals: both want to make a maximum profit. To make the transaction secure and precisely fix the rights and obligations of both parties, written contracts are negotiated. If large amounts of money are involved, a notary is consulted.

When the notary and the parties use a computer network for the contract negotiations, it is highly desirable that the security gained by the consultation of the notary is not sacrificed for the speed and comfort of the negotiations. The two major obligations of a notary are:

- to guard the privacy of the contract by granting access only to the contract parties
- to guard the integrity of the contract (and, by that, the binding force) by allowing no single party to modify the contract without agreement of the other party.

As a notary cannot trust systems that are not under his own physical control, it is a very basic part of the security policy to keep the contract on the notary’s own system, and to provide access only via well-defined, audited operations that are under absolute control of the notary.

This scenario cannot be modelled using the simple approach of one single confidence domain attached to an object. We now need the notion of confidence domain lists that attach more than one confidence domain to an object, together with rules of their interaction. In the contract example, two confidence domains attached to the contract and two rules for their interaction allow modelling the privacy and integrity requirements. One confidence domain contains the notary and the first party, the other one the notary and the second party.

The notary’s security requirements are satisfied if:

- the contract may be accessed only from systems in the union of all attached confidence domains; in this case, only from the notary’s system and the systems of the two contract parties (guarantee of privacy)
- the contract is physically confined to the intersection of all attached confidence domains. This locks the contract on the notary’s system and ensures that the contract is modified only via well-defined, audited operations that are under absolute control of the notary’s system (guarantee of integrity).

In standard systems where documents are accessed via common document processing tools like text editors, not even the parties’ systems may be trusted to access the contract directly. Here, the contract’s confidence domain just contains the notary’s system. To allow the contract parties to change the contract, access is provided by an intermediate program on the notary’s system that ensures that only well-defined,
In object-based systems (such as the BSA [HKK93]), access via intermediate objects is obsolete due to the type-specific and fine-grained object interfaces. Instead, contracts can be implemented by a dedicated abstract data type that defines exactly what the contract parties and the notary may do and that additionally provides logging or auditing functionality.

Summary: The examples demonstrated the use of confidence domains. In summary, they implied basic qualities as well as requirements for basic mechanisms needed for their implementation.

(a) As objects even on the same host may have quite different security requirements, confidence domains are defined individually for each object.

(b) The confidence domain of an object defines the set of hosts that are trusted to correctly enforce the security requirements that are considered a precondition for any communication with the object. A confidence domain thus restricts both an object’s visual range and its visibility to other objects.

(c) The confidence domain of an object defines the set of hosts that are trusted to safeguard the physical integrity and privacy of the object and thus may become a storage site for the object.

Any object may have associated more than a single confidence domain. While the concept of confidence domains does not itself imply rules for the evaluation of multiple domains, our interpretation of the model uses the following simple evaluation scheme.

Given a set of confidence domains (that are themselves sets of systems), either the union or the intersection of these sets can be used to determine the object’s overall communication range. Unfortunately, there are drawbacks for both of these constructors. Using the intersection for communication operations implies that any partner the object communicates with must be member of every associated confidence domain, making domain handling clumsy. On the other hand, the intersection provides a simple way to definitely prevent communication with a known set of malicious systems without the need for modifying every associated domain, while the union constructor makes adding a new system easy.

Fortunately, we may combine the virtues of both alternatives using negative system entries in a domain similar to the concept of negative access rights that make access control lists both powerful and easy to handle [KH90, HKK93]. Together with the positive entries a confidence domain may contain also negative entries that denote systems that are not trusted under any circumstances, although they may be part of another associated domain.
A confidence domain then is a pair of sets \((CD^+, CD^-)\), denoting the set of positive and the set of negative system names. A set of confidence domains \(CD_i = (CD^+_i, CD^-_i)\) is evaluated by \(\bigcup CD^+_i \setminus \bigcup CD^-_i\) (\(\setminus\) is the "set minus" operator defined by \(A \setminus B := \{a | a \in A \land a \notin B\}\)).

Considering point (c) there is a significant difference in trust between allowing an object just to communicate with objects on another system and allowing an object to migrate to that system. As the violation of an object’s physical integrity or privacy can always be used to fake a communication, trust in a storage site covers trust in the same site with respect to communication. As a consequence, in our interpretation of the model a second and more restrictive evaluation expression exists that guards operations that change the location of an object’s physical representation. This evaluation expression denotes that an object may migrate to a host only if all of the associated confidence domains agree: \(\text{target host} \in \bigcap CD^+_i \setminus \bigcup CD^-_i\).

Last but not least, the two following basic qualities of confidence domains were implied by the examples:

- There are no rules in the model that imprint a structure on the confidence domains in a network (e.g. hierarchical nesting).
- Confidence domains change dynamically; there are both political reasons (a former partner who broke a contract) and technical reasons (some system that was discovered to be a security hole) for making alterations.

To enforce confidence domains in a security architecture, the architecture must provide basic mechanisms for

- a user to authenticate a system. This allows users of shared systems to make sure no previous user has manipulated the system and installed a malicious operating system.
- a system to authenticate another system. This allows server systems to make sure that a user has logged into a client station and the client station now is working on its user’s behalf. System authentication makes sure that the client station runs a trustworthy operating system.

**Definitions**

This section defines a confidence domain as a finite set of system names and a confidence domain list as a finite set of confidence domains. One of the major principles in building new concepts for secure systems is simplicity. Guided by this principle, we defined confidence domains in the very elementary terms of basic set theory.
A confidence domain (CD) is a pair of two finite sets of system names:

\[ CD = (CD^+, CD^-) \]

\[ CD^+ = \{ s_1, \ldots, s_i \}, \quad s_i : \text{positive system names}. \]

\[ CD^- = \{ t_1, \ldots, t_m \}, \quad t_i : \text{negative system names}. \]

A confidence domain list (CDL) is a finite set of confidence domains

\[ CDL = \{ CD_1, \ldots, CD_n \}, \quad CD_i : \text{cd names}. \]

Two evaluation expressions for CDLs are defined, one liberal and one restrictive. The liberal expression returns true if and only if the partner host \( H \) is in at least one of the positive confidence domains attached to the object and not in any of the negatives:

\[ \text{evalCDL}_{\text{liber}}(CDL, H) ::= H \in (\bigcup CD_i^+ \setminus \bigcup CD_i^-) . \]

The restrictive expression returns true if and only if the partner host \( H \) is in all positive confidence domains attached to the object and in none of the negatives:

\[ \text{evalCDL}_{\text{restrict}}(CDL, H) ::= H \in (\bigcap CD_i^+ \setminus \bigcup CD_i^-) . \]

In our interpretation of the confidence domain model, we apply these expressions according to the potential severeness of the security violations. Communication operations with the associated object (both incoming or outgoing) are controlled by the liberal expression, while operations that migrate the associated object are controlled by the restrictive expression. Note that this classification is not part of the model but a deliberate interpretation based on practicality issues.

**CDLs and ACLs**

CDLs and ACLs are orthogonal. While ACLs restrict method invocation on an object with respect to the identity of the invoking subjects, CDLs restrict method invocation with respect to the identity of the hosts of the invosing subjects. The same subject invoking a method on the same object may experience different access rights on the object depending on the system the subject invokes the method from.

Thus CDLs support ACLs by making them more reliable: As any host \( H \in (\bigcup CD_i^+ \setminus \bigcup CD_i^-) \) is trusted to authenticate its subjects properly, then, if \( H \) is authenticated, a message from \( H \) saying "the caller is subject S" is believed. Thus the authenticity of a subject on such a system can be assumed when the ACL is evaluated.
Implementation

The implementation of confidence domains is discussed within the context of the BirliX Security Architecture (BSA) [HKK93]. Although the implementation is based on the paradigms of the BSA, the concepts may easily be integrated in similar environments supporting an object-based view in communication, supporting the concept of local and autonomous reference monitors, and providing authentication mechanisms as mentioned above. The minimum platform requirements are described in more detail at the end of this chapter.

The BirliX Security Architecture

The BirliX Security Architecture is a framework for integrating applications, security policies, and security mechanisms. A prototype of the framework has been embedded in the BirliX Operating System [HKL+92, HKLR92]. Similar to many of today’s state-of-the-art operating systems [Yok92, LAJ91, DJ, SGH+89], the BirliX Operating System supports an object oriented paradigm. Each individual object on the programming language layer can be embedded into an object infrastructure. The object infrastructure encapsulates a language object and provides persistency, transparent method invocation, protection, and migration. With respect to protection, the object infrastructure supports access control lists (ACLs), subject restriction lists (SRLs), and confidence domain lists (CDLs). In each of these lists, objects are identified by unique object identifiers. In contrast to systems having symbolic names in ACLs (e.g. DSSA [GGKL89]), this avoids the need for reliable naming services when one of these lists is evaluated. For more details concerning ACLs and SRLs in the BSA see [KH90, HKK93].

Figure 2: The BSA Object Infrastructure

ACLs, SRLs, and CDLs strongly depend on the idea that objects interact by method invocation only. With respect to a single object, incoming and outgoing method invocations are well-defined points where object interaction may be intercepted. With respect to the definition of a reference monitor given in [TCS83], this feature exactly provides requirement (b) that allows a reference monitor to enforce access control on every object access.
Method invocation is implemented via a basic remote procedure call communication mechanism. While local calls are handled directly by a communication nucleus, non-local calls are reflected to a network assistant. The network assistant implements remote procedure calls by running a corresponding network protocol. With respect to confidence domains, the assistant is also responsible for the authentication of remote hosts and the bookkeeping of encryptions keys that are exchanged during the host authentication procedure. To avoid performance bottlenecks, the network assistant is multi-threaded.

Figure 3: Local and Remote Object Communication

Confidence Domains in the BirliX Security Architecture

From an abstract point of view, a confidence domain is an object maintaining a list of host names, with methods to lookup, add and delete names. A confidence domain has a name, is persistent and protected against unauthorized manipulation.

Within the BirliX Security Architecture, these requirements are provided by the object model of the underlying BirliX Operating System. Implemented by an ordinary BirliX object, a confidence domain is denominated via the standard naming services. Persistency and protection are provided by the encapsulating object infrastructure (see figure 2). With respect to protection, confidence domains themselves have ACLs, SRLs, and CDLs that control the invocation of methods such as AddHost or RemoveHost. Note that the same recursiveness as known from ACLs protected by ACLs can be observed (and the same solutions apply, too).

As confidence domain lists are sets of CDs and CDs are ordinary objects, CDLs are object groups that can be implemented by any efficient group management mechanism, e.g. as found in newer work rooting in the ISIS project [Bir91]. As our implementation platform still lacks such a mechanism, our prototype implementation maintains a CDL as a list of CD names that is attached to an object in the same way as an ACL. In fact, the same code within the object infrastructure is used for accessing ACLs and CDLs.
As a CDL is a group object, access to a CDL is controlled by ordinary group access protection. In our current makeshift implementation, a separate ACL controls access to an object’s CDL. Currently, a CDL has neither a SRL nor a CDL.

Figure 4: Structures for Confidence Domain Implementation in the BSA

For convenience, two predefined standard confidence domains exist on each node: one set (MyHost), containing just the local host, and one set (AllHosts), being an alias for a dynamic set containing every host in the universe. Also for convenience, whenever an object is created the initial value of a CDL is inherited from the creating object.

As CDLs control object communication and migration, they are by definition part of the security mechanisms that are embedded within the local reference monitor [TCS83]. The reference monitor of the BirliX Security Architecture exactly has two components for the implementation of CDs.

One component maintains a small data base of successfully authenticated systems. This data base grows on demand. Whenever a communication with a not yet authen-
ticated host occurs, the host authentication protocol is run, provided the host is in the confidence domain of the involved local object. As a result from the authentication protocol, the data base is updated with a symmetric key for encrypting/decrypting messages to/from that host. The data base is kept fresh by periodical reauthentication of the contained hosts. The data base itself is embedded within the network assistant; for details of the host authentication protocol see [Gro91].

The second reference monitor component intercepts incoming/outgoing messages for each individual object and checks their validity against the object’s CDL. As object communication is by method invocation only, a well-defined interception point within the object infrastructure exists where also ordinary ACL checks are performed.

Outgoing Method Calls: When a subject invokes a method in an object, the invocation is intercepted within the subject’s infrastructure and is checked with respect to the subject’s CDL and the object’s location.

Figure 5: Method Call Interception

A CDL is a hash-table, allowing for an efficient implementation of invocation interception. The check consists of just a hash-lookup (within $evalCDL_{lib}$ or $evalCDL_{restrict}$, see below) and a single $IF$ - statement.

In the following, let $S$ (subject) and $O$ (object) be identifiers for the caller and the callee in a communication; $M$ is a method identifier and $P$ are the corresponding method invocation parameters. $ReceiverHost$ is an information obtained from the naming system when $O$’s symbolic name was transformed to a pair (unique id, location). The $CDL$ is located within the subject’s infrastructure.

The procedure that implements method interception within the subject’s infrastructure is $MethodCallIntercept$; it is called by that part of the infrastructure that implements method invocation, right after the method parameters have been packed into a representation that matches the rpc interface of the communication system.
Infrastructure.MethodCallIntercept(ReceiverHost, O, M, P) ::= 
    IF evalCDL(ReceiverHost, CDL, R) 
    THEN 
        NetAssistant.WithinCDCall(ReceiverHost, (S, O, M, P)) 
    ELSE 
        reject call 
    END 

WithinCDCall is implemented either directly within the local communication system (for local calls, see figure 3) or reflected to the network assistant. As the network assistant authenticates remote hosts on demand, the first step is to lookup the authentication database (called the GoodGuyDatabase), whether the host of the object has already been authenticated. If the lookup fails, the net assistant tries to run the authentication protocol described in [Gro91]. On successful completion, the authentication database is updated with a mapping of the authenticated host to a (symmetric) encryption key obtained as one result of the authentication protocol.

Let Encrypt and Decrypt be matching functions that encrypt respectively decrypt a 4-tupel (S,O,M,P) based on a single (symmetric) key.

\[ \text{Encrypt} : (\text{Key}, (S, O, M, P)) \rightarrow \{S, O, M, P\}_{\text{Key}} \]
\[ \text{Decrypt} : (\text{Key}, \{S, O, M, P\}_{\text{Key}}) \rightarrow (S, O, M, P) \]

Further, let:

\[ \text{Authenticate} : \text{Host} \rightarrow \text{Key} \]

be a function that authenticates a pair (operating system, central processing unit) as described in [Gro91] and that results in a symmetric key for encrypting/decrypting future communication with the authenticated system.

NetAssistant.WithinCDCall(ReceiverHost, (S, O, M, P)) ::= 
    IF Key ← GoodGuysDatabase.Lookup(ReceiverHost) 
    THEN 
        SendMsg(ReceiverHost, Encrypt(Key, (S, O, M, P))) 
    ELSE 
        IF Key ← Authenticate(ReceiverHost) 
        THEN 
            GoodGuysDatabase.Insert(ReceiverHost, Key) 


SendMsg(ReceiverHost, Encrypt(Key, (S, O, M, P)))
ELSE
  communication failure: unable to authenticate remote host
END
END

SendMsg is provided by the underlying communication protocol. It delivers the message to the network assistant of ReceiverHost as a part of the rpc implementation.

Incoming Method Calls: All incoming messages in a host are received by the network assistant. Within the assistant, one or more threads are accepting messages via the operation ReceiveMsg. SenderHost is provided by the underlying communication system.

NetAssistant.ReceiveMsg(SenderHost, \{S, O, M, P\}_Key) ::= 
  IF Key ← GoodGuysDatabase.Look up(SenderHost)
  THEN
    (S, O, M, P) ← Decrypt(Key, \{S, O, M, P\}_Key)
    ForwardWithinCDCall(O, SenderHost, (S, M, P))
  ELSE
    IF Key ← Authenticate(SenderHost)
    THEN
      GoodGuysDatabase.Insert(SenderHost, Key)
      (S, O, M, P) ← Decrypt(Key, \{S, O, M, P\}_Key)
      ForwardWithinCDCall(O, SenderHost, (S, M, P))
    ELSE
      communication failure: unable to authenticate remote host
    END
  END
END

The call is forwarded to the object via ForwardWithinCDCall. Within the object’s embedding infrastructure, incoming method calls are received via WithinCDAccept.

Infrastructure.WithinCDAccept(SenderHost, (S, M, P)) ::= 
  IF evalCDLiberal(SenderHost)
  THEN
    Object.M(P)
  ELSE
    reject call
  END
The subject identifier $S$ is used by the object's infrastructure for checking the subject's rights to invoke the specified method $M$ with parameters $P$. The checking can either be as simple as based on an ACL or be more complex using the custodian paradigm of the BSA [HKK93]. The authenticity of $S$ is trusted here, because the object's CDL denotes that the sender host can be trusted (not to forge $S$) and encryption is used to maintain integrity for host-to-host communication.

**Performance**

While no experience has yet been gained with the overall performance of a large-scale system with security policies based on confidence domains, those components that implement confidence domains within the local infrastructure of an object have been carefully measured.

As mentioned above, all objects within the BSA are encapsulated by a personal environment that provides the object with transparent method invocation, persistency, and security. A CDL is hooked into this infrastructure in the same way as any other security mechanism, e.g. an access control list. Like the ACL, the CDL is evaluated on every method invocation; additionally, a CDL is accessed to check outgoing method calls issued by active objects (e.g. processes).

If no CDL is attached to an object, the checking overhead of within the infrastructure is pretty small, taking less than 0.1 $\mu$sec on a 50MHz 486 machine.

If a CDL exists, the time for checking depends - in our current implementation - on the total number of domains, on the size of each domain, and on the size of a system name. In our environment (the BSA), system names are unique identifiers of 128 bits each.

While hash tables scale better, we prefer in our current implementation the simplicity of an unordered linear list for representing a single confidence domain, as we use a very simple sequential lookup procedure. This procedure takes about 3.8 $\mu$sec for a full scan of a set of ten elements, which is typical for our current applications.

In our application environment (program development and document processing within a research institute of 30 people within a company of 1000) a typical confidence domain configuration for a document consists of each document having 4 confidence domains (private, file server, project, institute) with one, three, five, and thirty elements. To check a CDL, each of the four domains in the list has to be looked up. As most accesses to a document are local, in most cases the private confidence domain already matches. Thus the typical average lookup time here is 1.016 $\mu$secs. The worst case using the sequential lookup procedure costs 14.86 $\mu$secs.

These figures show that even with very simple implementation structures (linear lists) confidence domains can be implemented extremely efficient. Using hash tables will give us scalability; a preliminary hash function currently is tested that finds an
element in about 0.89 μsecs, including already 20% collision handling. This hash function already reduces the above worst case cost from 14.86 μsecs to 3.56 μsecs.

For considerations on the performance in-the-large, see the comparative discussion in the chapter on Related Works.

**Confidence Domains on Other Platforms**

The object-oriented paradigm and the reference monitor properties of the BirliX Security Architecture allowed for a straight-forward implementation of confidence domains. This section discusses the minimum properties any other platform must provide to support confidence domains.

First, a mechanism is needed for authenticating host names. While from the point of view of authentication, host names can be treated as special object names, an authenticated host is more fundamental: hosts store and execute objects, and even an authenticated and trustworthy object cannot be trusted if it resides on a non-trustworthy host with a non-trustworthy operating system. Both the operating system and the hardware may successfully try to violate the object's integrity, e.g., by stealing its private key. Mechanisms for host authentication can be found in the DSSA [GGKL89] and more formally in the BSA [Gro91].

Second, the underlying system must be able to control all incoming and outgoing communication with any object. This is, basically, the property of a reference monitor as defined in [AGS83, TCS83]. Being able to control all incoming communication with an object is a feature of any rigorous ACL-based protection, and a similar mechanism has already been proposed for outgoing communication, too, in [KH90].

Third, CDLs and the trusted hosts database that keeps the keys of authenticated hosts have to be maintained. While the trusted hosts database can be maintained as volatile data base in the operating system kernel (e.g., similar to arp-information within the TCP network protocol), CDLs can be attached to objects similar to ACLs. The BSA implementation makes use of the existing ordinary ACL access operations; it implements a CDL by type-overlaying an ACL entry an appending CD entries at the end of the ACL.

```c
/* ACL entry type definition */
typedef
  struct {
    UniqueId Subject;
    Methods PositiveAccessRights;
    Methods NegativeAccessRights;
  } ACLEntry;
```
typedef
   UniqueId CDLEntry;

typedef
   union {
      ACLEntry AccessControlListEntry;
      CDLEntry ConfidenceDomainName;
   } ACL [ACLSize+CDLSize];

The conventional protection checking routines based on the ACL have been modified to include a call to one of the evalCDL procedures.

**Related Works**

In several ways our work is influenced by the Digital Distributed System Security Architecture (DSSA) [GGKL89] and the Kerberos Authentication Service [SNS88, Koh91].

Within the scope of the DSSA, Gasser and McDermott describe in [GM90] a delegation scheme for distributed systems. Like Gasser and McDermott we assume that a distributed system consists of a collection of mutually suspicious and autonomous computer systems, each one with its own local reference monitor. We also support the view that new concepts in such systems must have minimum need for online interaction between users and remote security services. We agree on the basic assumption that all entities in the system must either be able to authenticate themselves or be authenticated with the help of an authentication service like X.509.

Gasser and McDermott focus on delegation, which is not a primary aim of confidence domains. Nevertheless, a comparison to a well known concept may help to assess a new one.

The delegation scheme of Gasser and McDermott uses cryptographically sealed tickets for authorizing intermediates to act on behalf of a principal, thus allowing servers to decide whether to grant a request that is not directly issued by a client. In terms of confidence domains, the delegation of a right to speak for to a collection of intermediates is similar to the definition of a confidence domain consisting of systems that are permitted to act on behalf of a client. By associating this domain to an object we restrict the communication of that object to the systems in just that domain. If we
then use the restrictive and the liberal evaluation expressions to distinguish between ordinary communication and delegation, we already have the foundations of a simple delegation scheme. Any service request that arrives via an intermediate at a server can be considered genuine if the original request is authentic, because the server knows that delegation can only happen within the borders enforced by the domain.

Delegation in the DSSA also covers delegation via several intermediates, so-called **delegation chains**. A delegation chain is a sequence of tickets in the DSSA, and a sequence of confidence domains in our model. While in the DSSA the server has to evaluate all tickets of a chain, confidence domains allow a server to immediately accept a request. Again the argument is that there must exist a confidence domain path between the client and the server, because otherwise no delegation could have taken place.

Nevertheless, confidence domains are not for free; they require authentication and delegation evaluation *en route* between client and server, carried out by every delegating system. Both, system authentication and the checking of a ticket, are based on public key encryption and cost a similar price. The difference lies in the frequency of the operation. A system within the delegation path is authenticated only once (refreshes not considered), whereas tickets need to be evaluated for every request or user. Although usage patterns exist where ticket-based schemes perform similar or better to the system authentication approach, results presented in [AFRS87] indicate that system authentication in general has a performance advantage.

A second difference lies in the objects of trust. Confidence domains are based on trust in *systems*, while the DSSA delegation scheme is based on trust in *objects* acting as intermediates. This implies a different granularity as well as a different paradigm. While in the confidence domain model an intermediate is no longer trusted when it migrates to a non-trustworthy host, the DSSA delegation allows a client to specify delegation in the granularity of individual operations.

Finally, a significant difference exists in the support of security policies. A confidence domain list associated to an object is protected by an individual access control list. Thus the object may be denied access to its own CDL, and mandatory security policies may be enforced.

Another example where confidence domains provide a solution to a yet unsatisfactorily solved problem can be observed within the Kerberos system. Kerberos is a widely-know authentication and ticket-granting system based on the client/server model. Kerberos supports the notion of a *realm*, a domain in which all clients and servers trust the same authentication server and the same ticket granting server. In several ways, the need for a central ticket granting service might be considered a weakness. Besides general availability and scalability problems, any change to the ticket granting policies requires a major action that involves the security administration personnel. Besides taking its time, the reconfiguration of trusted servers is an operation that is critical for the security of an entire realm.
In the notary scenario, the notary trusts his two clients to make changes to the contract. He will also trust the machines of his clients to authenticate his clients correctly (because his clients told him so). In Kerberos, setting up this scenario requires

- the notary to reconfigure his local ticket granting server to accept tickets from the clients’ local ticket granting servers and to issue tickets to the clients for accessing the contract server
- each client to reconfigure his local ticket granting server to grant tickets for the notary’s ticket granting server

Using confidence domains, the notary creates two new domains, each containing one of the client’s machines, and attaches both domains to the contract.

A noticeable difference is that Kerberos needs three ticket granting servers to be reconfigured, whereas confidence domains are established by a local operation on the involved object, similar to modifying its ACL. Another difference is that confidence domains are enforced by the local system alone; no trust in a remote security server is needed, a feature that becomes essential in large networks.

On the other hand, Kerberos and confidence domains work together well: confidence domains are attached to individual objects (contracts, documents, etc.) and control on every object access whether the caller has been authenticated within a Kerberos realm that is trusted by the owner of the object. Only then the caller’s ticket is accepted to check his individual rights.

Figuratively speaking, whereas Kerberos realms enforce the borders of a country, confidence domains control the doors of buildings and vehicles.

**Conclusions**

Confidence domains are an approach to formally specify human belief in the trustworthyness of computer systems. The formal specification is used by the security architecture to impose restrictions on the communication and mobility of system entities. Based on three scenarios we developed a simple model in terms of elementary set theory. We implemented the model within the context of the BirliX Security Architecture, outlined the basic performance and distilled the minimum properties any platform must provide to support confidence domains.

We discussed the model’s relations to two other topics in computer security: delegation and authentication. As examples, we used the delegation scheme in the DSSA and the authentication scheme in the Kerberos system. We concluded that confidence domains minimize the number of concepts that must be understood to manage security, that they work together well with existing approaches, that they perform well,
that they make few assumptions about the underlying platform, and that they can be implemented by small, simple, and well-confined code within the reference monitor.

**Acknowledgements**

Many of the results presented here originate from discussions within the Security Architectures Group at the Institute for Software Design Technology at GMD, including Hermann Härtig, Oliver Kowalski, Jochen Liedtke and Wolfgang Lux.

A part of this work was funded by the German Ministry of Research and Technology under contract number 01 IS 202 DO.

**References**


[Gro91] M. Gross. Vertrauenswürdiges Booten als Grundlage authentischer Basis-

systeme. In Tagungsband der GI-Fachtagung 'Verlässliche Informationssys-


