

Alternative protection systems for OO Environments: Capability-Based Protection and the SSCLI-Rotor

Darío Álvarez Gutiérrez
Dept. of Informatics,
University of Oviedo
c/ Calvo Sotelo s/n
33007, Oviedo, Spain
darioa@uniovi.es

María Ángeles Díaz Fondón
Dept. of Informatics,
University of Oviedo
c/ Calvo Sotelo s/n
33007, Oviedo, Spain
fondon@uniovi.es

Iván Suárez Rodríguez
Dept. of Informatics,
University of Oviedo
c/ Calvo Sotelo s/n
33007, Oviedo, Spain
banisr@telecable.es

ABSTRACT

Protection (access control) is a crucial issue in modern software systems. There are many different protection mechanisms, including Access Control Lists and the Code Access Security included in .NET. Capabilities are other well-known protection mechanism that has many merits. This paper describes a form of capability-based protection specially suited for Object-Oriented environments based on OO Virtual Machines that compares favorably with the .NET CAS mechanism in many contexts. The implementation of this protection model into the Microsoft SSCLI-Rotor implementation of the .NET platform is shown (RotorCapa), involving modification of core VM structures and behaviour of instructions. Besides other benefits, the early performance results of the RotorCapa system compared with .NET CAS protection are very encouraging, as it does not suffer from the exponential degradation of performance imposed by the security stack walking mechanism of .NET.

Keywords

Security, protection, access control, capabilities, performance, virtual machine, code access security, SSCLI, Rotor, .NET security.

1. INTRODUCTION

A protection mechanism (access control mechanism) is a security measure in computer systems that restricts access from a piece of code (subject) to a resource (object). An example is the well-known Access Control List protection mechanism: its variants are used in operating systems such as Unix.

Object-Oriented environments based on OO Virtual Machines (Java and .NET being prominent examples) need a protection mechanism, too. Subjects are here objects (instances of a class), and the resource to protect is also an object (calls to a method of an instance of other class). .NET is shipped with a protection mechanism called Code Access Security [Wat02], part of a more

comprehensive security system. The mechanism is based on a form of stack introspection [Wal97] (stack walking of the internal VM stack holding security information).

But there are other protection mechanism, such as capabilities. Our research focus on the application of capability-based protection to object-oriented environments. We have implanted capability-based protection into the SSCLI-Rotor (the RotorCapa system¹). SSCLI-Rotor [Stu03] is Microsoft's Shared Source implementation of the Common Language Infrastructure (.NET).

This paper describes briefly our model of capability-based protection and its advantages (in general and compared to the .NET CAS mechanism). Then, the implementation of this model into the SSCLI-Rotor is presented with more detail, involving modifications to the core of the Rotor VM (Just in Time Compiler, object layout, addition and

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

.NET Technologies'2004 workshop proceedings,
ISBN 80-903100-4-4

Copyright UNION Agency – Science Press, Plzen, Czech Republic

¹ RotorCapa development was supported by Microsoft Research through the RFP Rotor Awards. Home page is at <http://www.di.uniovi.es/~darioa/rotorcapa/> and at the SSCLI Community Site <http://rotorcapa.sscli.net/>

modification of instructions, etc.). Some early and encouraging performance results are presented in the next section. Another section draws some conclusions about the SSCLI-Rotor as a research platform gained while developing the implementation. The paper ends with a comparison with related work, and the conclusions and future work section.

2. CAPABILITY-BASED PROTECTION

Pure Capabilities [Den66] are a well-known protection mechanism that can be used to implement a comprehensive set of flexible security policies. A capability is basically a ticket which names an object (resource) and a set of permitted operations on that object (permissions) (Figure 1). The only requirement for an object (subject or client) in order to use another object (object or server) is to hold a capability pointing to the server object with adequate permission to use the intended operation. Consequently, an object will hold just the minimum protection information relevant to it: the rights to just the objects it will use.

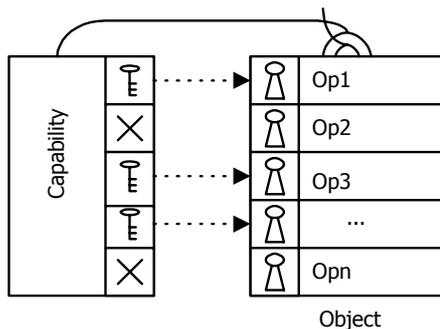


Figure 1. A capability.

Capabilities for OO Environments based on OO Virtual Machines

A big advantage of capabilities over other protection mechanisms such as the before mentioned access control lists, stack introspection, etc. [Wal97] is that they can be smoothly and easily integrated with the object model.

In our version of capabilities [Dia99], the protection information (permissions) can be integrated with object references in the machine, and the mechanism

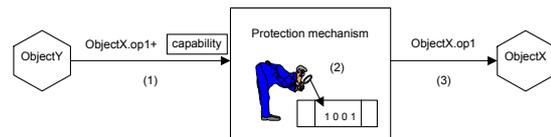


Figure 2. Checking permissions in a capability.

for testing the permissions can be integrated with the method call process (Figure 2. If the reference does not hold a permission for the method called in the destination object, the call fails, and an exception is raised).

Modifications to instructions (and structures) dealing with references must be also done accordingly. Just a new instruction to restrict the permissions a given capability is holding (to follow the principle of least privilege) is needed.

In fact, there are no conceptual changes to the Object Model, and the protection can be (and should be) seen as another property of the Object Model (encapsulation, inheritance, ...and protection).

We have previously worked with this model with our own OO environment with OO VM [Alv98] and have found advantages [Dia99] such as:

- Flexibility and adaptability
- High performance
- Integration with the object model
- Fine granularity of protection
- Reduced Trusted Computing Base, as a simple mechanism is implemented with a small code.
- More Hardened Systems, as the principle of least privilege can be followed with no restrictions.
- Compatibility with existing applications, as capabilities are used as normal references in applications.
- Scalability. Managing capabilities for thousands of objects is not a problem, as they are managed and stored as normal references in the objects themselves.

Capabilities have some drawbacks, most notably revocation problems, although there are solutions such as facades and reference monitors in case they were needed.

3. WHY CAPABILITIES FOR .NET?

The .NET security model is very complete and has many advantages. It includes a Code Access Security mechanism. So, why using other protection mechanism?

Capabilities in general have its own merits. They are clearly superior to Access Control Lists in terms of confinement [Har02]. Besides, there are some points in .NET security for some applications where our model of capabilities is a best fit:

- **Complexity.** The .NET security system is comprehensive and thus complex: evidence, policies, permission sets, stack walking mechanisms... For many applications that just need the base form of protection (such as the one provided by capabilities) this is overkill:
 - **Footprint and overhead.** The code, data, and runtime overhead needed by the .NET security system is present, although just a fraction of its power is used.
 - **Big Trusted Computing Base.** For the same reason, the trusted computing base of the system is big, and the probability of security bugs increases.
- **Access to source code needed.** To add protection to a given class, access to the source code of the class (to demand permissions) is needed, and the code to represent the permissions has to be created, too. With capabilities, any binary object can be protected anytime without effort (it just requires setting permission bits in the references).
- **Protection at the level of the class, not at the level of individual instances.** Since permissions are assigned based (roughly speaking) on the class of a client instance, not on an instance-by-instance basis. With capabilities, permissions are assigned on a reference-by-reference basis. Two objects of the same class can hold different permissions when calling a third object.

And finally, another reason is that .NET can be used just as a platform to research on other protection mechanism.

4. SSCLI-ROTOR IMPLEMENTATION OF CAPABILITIES: STRUCTURAL CHANGES

Since we had previous experience implementing capabilities in a VM, we expected to follow a similar path to implement capabilities into the SSCLI-Rotor 1.0. However, due to the constraints and the architecture of SSCLI-Rotor, we had to resort to a different approach, which is described in this and the following section.

Representation of capabilities in objects

Each (reference) attribute in an object can have a set of access permissions attached. A capabilities table holding these permissions is attached to every object (in the OBJECT structure), with an entry for each reference held in the object (Figure 3). A lazy-creation strategy is used so that objects that do not use protection (i.e. do not apply the operation to restrict permissions to a reference) do not have this necessary protection overhead.

The same is done for array references.

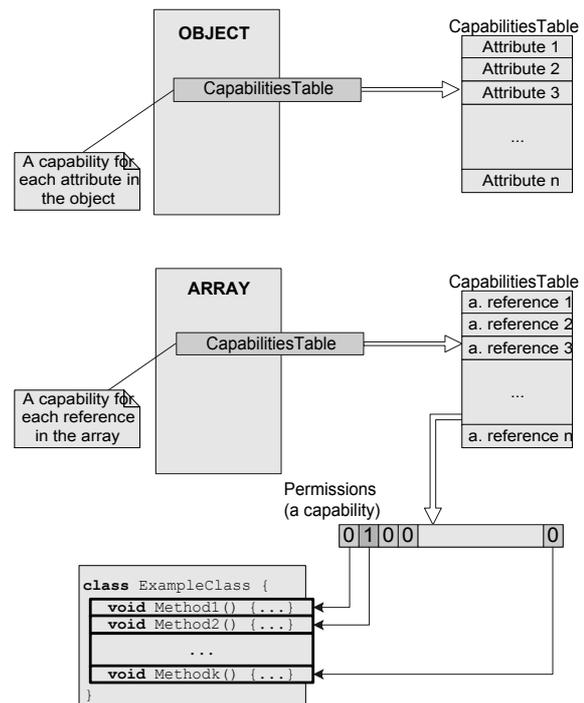


Figure 3. Representing capabilities in objects and arrays.

Support in behavioural structures

Implementing capabilities also needs support in structures used for behavioural purposes (execution): the execution stack (activation records) and the operation (evaluation) stack.

An activation record (stack frame) for a method can have references to other objects in local variables and method attributes (parameters). These references can have an associated set of permissions. A scheme using capabilities tables similar to the one used with objects is applied.

The permissions for these references are stored using one capabilities table for variables and one for attributes. These tables are organized into stacks that grow in parallel with the activation records (main stack).

The operation stack also holds references that can have permissions (for example, a reference to an object and references to object parameters are stacked prior to a method call to the first object. These references have associated permissions. A capabilities stack that mimics this operation stack holds the permissions for the references.

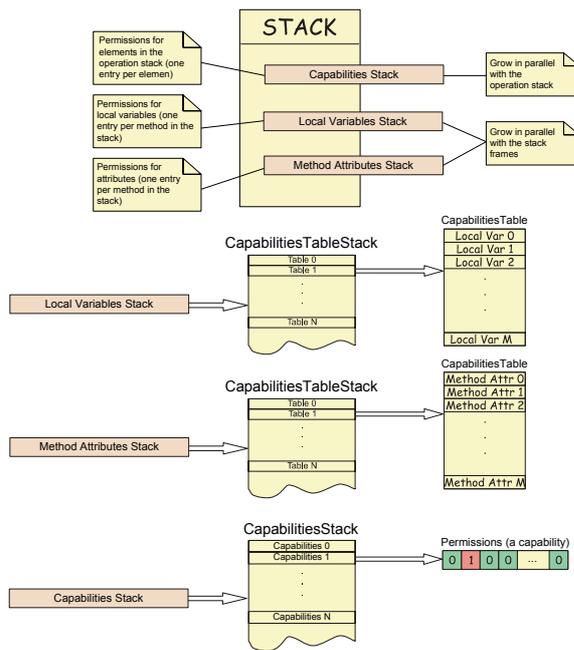


Figure 4. Representing capabilities in behavioural structures.

5. SSCLI-ROTOR IMPLEMENTATION OF CAPABILITIES: BEHAVIOURAL CHANGES

New instruction: Restrict <method>

This new instruction acts upon a reference (top of the stack), and denies access to the method specified, restricting the set of methods that can be called using the reference.

This is the primitive operation for security. Initially, the creator of an object holds a reference with all the permissions. The creator object can duplicate this reference, restrict some methods, and then pass the reference to others for secure computation (the set of available operations for these objects is restricted).

Call and callvirt now check permissions

The other pillar of capability-based protection is that method calls to an object should only be allowed if the reference (capability) used for making the call has the permission (bit) for the method set active.

Thus, **call** and **callvirt** instructions are modified accordingly. The instruction check that the reference to the object called (top of stack) has an asserted permission for the method being called (the bit for the method is “1” in the implementation). If the reference does not hold a permission (bit “0”) a protection exception is raised.

Modifications to many other instructions

Although capabilities only affect the semantics of the “call” instruction (now a security exception might be thrown), MANY other instructions are indirectly affected. With capability-based protection, ALL references, including local variables, references in the stack, etc., have an attached set of permissions (conceptually, that is the philosophy of capability-based protection). The behaviour of the instructions that deal with references must take this into account, “manually” copying, deleting, etc. the set of permissions when dealing with references, as represented in structural and behavioural structures as shown before.

Some of the instructions that had to be modified are:

- Creation of new objects: **newobj** (when a new object is created a reference is returned. This reference has an associated set of permissions, initially set to “1” for the creator).
- Storing from the stack: **starg**, **stlocs**, **stfld**, **stsfld** (when storing a reference from the stack, the permissions associated to the reference must also be copied to the destination reference)
- Loading: **ldarg**, **ldloc**, **ldnull**, **ldelem.x**, **ldelem.ref**, **ldfld**, **ldsfld**, **ldsfla**, **ldstr** (symmetrically, permissions associated to a reference must be copied when the reference is pushed in the operation stack)
- Various: **dup**, **isinst**, **box**, **unbox**

Example of CLI code with capabilities

The following is a small example of the use of capabilities in the SSCLI (the **restrict** instruction):

```
...
// An object is created and a reference
// (capability) is left on the stack
newobj instance
    void Test::ctor()
// A method is restricted in the
// capability in the top of the stack
restrict instance
    void Test::Message()
// Now the method is invoked using the
// reference in the top of the stack
// The reference can be stored, cloned,
// passed as an argument to other
// objects, etc.
callvirt instance
    void Test::Message()
// The call will not succeed and an
// exception is raised at this point,
// as the reference used has not the
// permission to call “Message” set
...
```

6. PERFORMANCE

Preliminary tests were made, comparing the SSCLI-Rotor capabilities system (RotorCapa) with a “normal” SSCLI-Rotor with the security system active. Access control to a method was checked by a very simple test program.

The test program involved a class with a given method. An instance of the class was created and then the method was repeatedly called using the reference

to the object created. To protect the call in SSCLI-Rotor, a .NET permission protecting the method was created and granted to the original class. In RotorCapa, the permission for the method in the reference remained set to achieve the same effect.

The same test program was run with different stack depths before calling the method.

As the .NET security mechanism relies on stack walking, it was expected that the time taken by SSCLI-Rotor to execute the same program would increase with the stack size, as and domain intersections and stack walks are getting longer. The exponential degradation of performance shown in figure 5 confirms this.

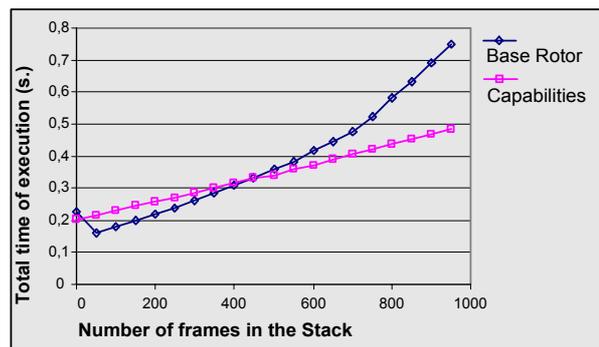


Figure 5. Performance of RotorCapa and SSCLI-Rotor (simple test).

RotorCapa, on the other hand, does not rely on stack walks. The mechanism always checks that a given permission is set on a reference or not, and this, ideally, is independent of stack size, and the number of permissions needed in the system. Thus, ideally, the time taken by RotorCapa should be constant. Actually, the figures show a linear increase of the time (much better than exponential). This can be related to the way method calls are handled and a non-optimal implementation of capabilities in this first version.

With modest stack sizes, RotorCapa performance is very similar to SSCLI-Rotor performance.

These performance results are very encouraging. On the one hand performance is equal or better than SSCLI-Rotor, and our implementation is barely optimized in this version. The test program was very simple. In fact, there is only one domain and one custom permission used. With more domains, and more custom permissions (as would be the case with real applications), the burden of stack walks, domain intersections and searches of permissions should be more apparent, and the performance of the capabilities system would become even more obvious.

7. PROS AND CONS OF THE SSCLI-ROTOR AS A RESEARCH PLATFORM

A very important bonus of working with SSCLI-Rotor is that we can now make further research on capabilities without having to resort build a complete VM to test the mechanism.

.NET is a system that has many “real” applications already done, and these applications can be directly used in SSCLI-Rotor (and therefore in RotorCapa). Thus, we can use real applications to research the cost of capability-based protection. To build similar applications in number and complexity for a custom system is something that is out of the picture. Obviously, the code to use capabilities must be added to the applications, but this is an incremental and relatively small process.

However, the implementation of capabilities in the SSCLI-Rotor source code was not as easy as we would have liked. Base Rotor security structures and code are deeply intermingled with the core of the Rotor VM. We did not try (yet) to delete them and avoid its overload. The nature of our project and the architecture of Rotor obliged us to “touch” almost every part of the Rotor VM: JIT, dozens of helpers for the JIT (assembly generation), memory layout, stack, threads, metadata, etc.

8. RELATED WORK

Capabilities as a protection mechanism are almost as old as computers, and many projects have used this protection mechanism, especially in the OS area. With the recent spread of security breaches in commercial Access Control List-based Operating Systems, there is a renewed interest in them, as shown in the EROS operating system [Har02], or the JX [Gol02] Operating System.

Capabilities were also used in object-oriented systems, for example in the Hidden Capabilities model [Hag96]. They were also used in OO systems based in Virtual Machines, such as the J-Kernel [Haw98] project for the Java platform.

All these projects use the basic philosophy of capabilities for protection. However, the specific variants differ in many aspects with our approach, mainly in how protection is smoothly integrated with the object model and the virtual machine structures and mechanisms in our system.

With respect to .NET and the SSCLI-Rotor, there are some projects that use the SSCLI to test or implement different protection mechanisms, such as

the implementation of the Delegent authorisation system for SSCLI-Rotor [Ris03], but none (as far as we now) related to capabilities.

9. CONCLUSIONS AND FUTURE WORK

Protection (access control) is a crucial issue in modern software systems. There are many different protection mechanisms, including Access Control Lists and the Code Access Security included in .NET.

Capabilities are other well-known protection mechanism that has many merits. We have developed a form of capability-based protection specially suited for Object-Oriented environments based on OO Virtual Machines. Our system compares favorably with the .NET CAS mechanism in many contexts, as it is much simpler, with a smaller overhead, footprint, and trusted computing base, does not require access to the source code of a class (or additional coding) in order to protect it, and grants protection at the level of individual instances instead of at the level of classes.

We have successfully implanted this capability-based protection mechanism into the Microsoft SSCLI-Rotor implementation of the CLI (.NET) standard (the RotorCapa system). This involved modifications to structural and behavioural structures of the VM to represent the permissions associated to the capabilities, as well as modifications to the implementation of instructions that deal with references.

As expected, one advantage of capability-based protection is visible in the early performance results of the RotorCapa system. The performance is at least as good or much better than the .NET CAS figures, that show an exponential degradation of performance with stack size. Since the test program was very simple and the RotorCapa version is not much optimized, it is expected that the results would be better with test conditions similar to the ones had with real applications.

SSCLI-Rotor has proved to be a good platform for research, as we did not have to build a complete commercial-like VM to test our protection mechanism. Besides, we had a direct access to the vast array of existing applications created for the .NET platform. However, the nature of our work, and because of the Rotor architecture, involved modifying the source code of the many of the parts of the core Rotor system (and that was not as easy as expected).

Future work will be precisely in the area of performance testing. In a first phase, we will develop a more comprehensive benchmark of test programs, to exercise different elements of the system, and to represent conditions more similar to actual applications. In a second phase, we will instrument real .NET applications that are readily ported to Rotor, to measure the performance of the capability-based protection mechanism in real production conditions.

10. REFERENCES

- [Alv98] Álvarez Gutiérrez, D., Tajés Martínez, L., Álvarez García, F., Díaz Fondón, M.A., Izquierdo Castanedo, R., and Cueva Lovelle, J.M. An Object-Oriented Abstract Machine as the Substrate for an Object-Oriented Operating System. *Lecture Notes in Computer Science*, 1357, pp. 537-544, 1998.
- [Den66] Dennis, J., and van Horn, E. Programming Semantics for Multiprogrammed Computations, *Comm. of ACM* 9, No 3, 1966.
- [Dia99] Díaz Fondón, M.A., Álvarez Gutiérrez, D., García-Mendoza Sánchez, A., Álvarez García, F., Tajés Martínez, L., and Cueva Lovelle, J.M. Integrating Capabilities into the Object Model to Protect Distributed Object Systems. *Proceedings of the International Symposium on Distributed Objects and Applications (DOA'99)*, IEEE Computer Society Press, pp. 374-383, 1999.
- [Gol02] Golm, M., Felser, M., Wawersich, C., Kleinöder, J. A Java Operating System as the Foundation of a Secure Network Operating System. Technical Report TR-I4-02-05, University of Erlangen-Nuremberg, 2002.
- [Hag96] Hagimont, D., Mosière, J., Rousset de Pina, X., and Saunier, F. Hidden Capabilities. 16th International Conference on Distributed Computing Systems, May 1996.
- [Har02] Hardy, N., and Shapiro, J. EROS: A Principle-Driven Operating System from the Ground Up. *IEEE Software*, pp 26-33, January 2002.
- [Haw98] Hawblitzel, C., Chang, C., Czajkowski, G., Hu, D., and von Eicken, T. Implementing Multiple Protection Domains in Java. 1998 *Usenix Ann. Tech. Conf.*, Louisiana, June 1998.
- [Ris03] Rissanen, E. Server based application level authorisation for Rotor. *IEE Proceedings Software*, 150(5), pp 291-295, October 2003.
- [Stu03] Stutz, D., Neward, T., and Shilling, G. *Shared Source CLI Essentials*. O'Reilly, 2003.
- [Wal97] Wallach, D.S., D. Balfanz, D. Dean, and Felten, E.W. Extensible Security Architectures for Java. 16th *Symp. on Operating Systems Principles*, Saint-Malo, France, 1997.
- [Wat02] Watkins, D. An Overview of Security in the .NET Framework. January 2002. <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnnetsec/html/netframesecover.asp>