

---

# **EROS: A Capabilities based and Persistent Operating System**

---

**CS384  
The Design of Operating Systems**

Submitted By: Matt Peterson  
Submitted To: Dr. Chris Taylor  
Monday, February 3, 2003

<b>ABSTRACT .....</b>	<b>1</b>
<b>INTRODUCTION .....</b>	<b>1</b>
<b>THE CONVENTIONAL APPROACH TO SECURITY.....</b>	<b>3</b>
SPAGHETTI SECURITY .....	3
ACCESS CONTROL LISTS .....	4
FAULTS OF THE ACCESS CONTROL LIST SYSTEM .....	5
<b>CAPABILITY DESIGN DECISIONS.....</b>	<b>6</b>
PURE CAPABILITIES ARCHITECTURE .....	6
EROS: A CAPABILITIES-BASED SYSTEM.....	8
DISADVANTAGES OF A CAPABILITY-BASED SYSTEM .....	9
<b>UNIVERSAL PERSISTENCE.....</b>	<b>10</b>
INTRODUCTION TO PERSISTENCE.....	10
EROS: AN ORTHOGONALLY TRANSPARENT PERSISTENT SYSTEM .....	10
<i>Advantages To Persistence.....</i>	<i>11</i>
<i>Disadvantages of Persistence .....</i>	<i>12</i>
<i>Implementation of Persistence .....</i>	<i>12</i>
<i>Global Transparent Checkpointing.....</i>	<i>13</i>
<i>Journaling .....</i>	<i>14</i>
<b>CONCLUSION.....</b>	<b>14</b>
<b>BIBLIOGRAPHY.....</b>	<b>16</b>

## **Abstract**

EROS (Extremely Reliable Operating System) is an operating system being built from the ground up in the systems research laboratories at John Hopkins University. EROS is an attempt to combine some very old ideas in the realm of operating systems with some newer ones in an attempt to create a more secure, faster, and much more reliable operating system than many operating systems to date. The primary difference between EROS and many other operating systems today is it uses capabilities to control access rights to objects and it uses global persistence as an alternative to file systems. This article gives a general overview of the EROS system and discusses in detail how capabilities and persistence make it such a secure, reliable, and quick system.

## **Introduction**

Today's computer users are constantly being bombarded with information concerning the vast amount of malicious computer viruses residing on the Internet. Reports of computer break-ins and other such illegal security-breaching activities are becoming quite common in our news. Just recently the "SQL slammer" worm has made headlines by exploiting a problem with Microsoft's SQL server. According to Marc Stiegler, "Reading the news media, one would conclude that all our computer systems are horrifically vulnerable to computer

hackers... That much is indeed true; we are all so vulnerable today it is more a joke than a question.”

Unfortunately, the technology for defeating such malevolent activity has existed for quite some time. This technology was actually developed by operating systems developers working on mainframes. “A couple of computer operating systems, notably Multics and KeyKos, were extremely resistant, indeed virtually invulnerable, to hacking and cracking. However, in the rush to the PC, the knowledge was forsaken.” (Stiegler)

KeyKos was particularly resistant because it was a persistent capabilities-based system, a simple but very secure system design. Recent attempts have been made to revive the ideas that made the KeyKos system so reliable and secure. These attempts have manifested themselves in the development of the EROS operating system.

The EROS operating system “started as a cleanroom reimplementation of KeyKos.” (Hardy and Shapiro, 26) Like the KeyKos system, the EROS’s operating system is a capability-based system. The EROS development team is also making an attempt to improve upon the KeyKos’s system design by adding some more features that have surfaced long after KeyKos was designed.

The EROS operating system initially began its development at the University of Pennsylvania towards the middle half of the 1990's. From here it migrated to John Hopkins University in 1999 along with its principle architect Jonathan Shapiro. At John Hopkins University, the EROS operating system is still in development. Because of bad experiences with estimations concerning release dates, the development team is not setting a date for EROS's release.

According to Shapiro, EROS has three primary objectives: simplification, containment, and security. EROS will give bigger systems simplification because it eliminates "as many diversionary requirements as possible." (Shapiro, "A Platform") EROS will have containment because it will allow "inevitable software failures to be caught and recovered from." (Shapiro, "A Platform") EROS will provide security because it enables "applications to safely expose sensitive information in a controlled way." (Shapiro, "A Platform") This is how EROS gets its name: the extremely reliable operating system.

## **The Conventional Approach to Security**

### ***Spaghetti Security***

According to Stiegler, today's security systems work similar to a defensive stonewall. In order to be completely safe, one must have the wall completely sealed to ensure that systems that they access on the Internet do not harm them.

But in order for many of the systems that need to save specific information to a person's hard disk to fully function, there must be some holes in this wall to allow this functionality. For example if a user was working with a java applet on the Internet to generate some sort of data, and that user desired to save that data then there would have to be some sort of hole in the wall to allow the applet to save such data.

In order to get around this, the user is forced to make certain compromises (holes) in his/her wall to allow certain processes to perform these operations. To prevent malicious software designed to exploit these holes, certain verification systems need to be installed to insure that only certain applications can have access through these holes. Occasionally after placing these "verifiers" it is discovered that they can be circumvented. So then secondary verifiers are implemented into the system. Eventually more problems arise after this system has been set up and the entire verification system eventually becomes what Stiegler calls spaghetti, the antithesis of security.

### ***Access Control Lists***

This is the basic problem of computer security, access control. According to Shapiro ("what is a capability") there are four main areas to access control:

1. You need to make sure that one user cannot damage or obtain access to another user's private information or data.

2. You must ensure that a program or user cannot do more than you intend them to do.
3. A user should be able to give other users access to specified files when desired.
4. A user's access rights should be revoked when that user is no longer authorized to have access to that particular document.

Many systems use access control list based systems inside of the verification systems that they use to identify which particular users have access to what. According to MSDN, an access control list is a list held by an accessible object containing a number of access control entries that each identify a user and specifies the access rights that are allowed, denied, or audited for that user. When accessing an object for a user, the system simply checks the access control entries in that object's access control list, if there exists an access control entry for that user and it says that the user is allowed access, then the system gives access. Otherwise if there is no access control entry, or if the entry claims that the user should be denied access, then no access is allowed.

### ***Faults of the Access Control List System***

According to Shapiro ("what is a capability"), access control lists only manage to meet two of his four previously mentioned main areas of access control. They prevent users from gaining access that they should not have, and they allow the

revoking of authorization (by systems administrators). Unfortunately, they do not necessarily limit what a process can do, nor do they allow users to grant read/write permissions on their files.

The access control list method of security can potentially create a tremendous breach in security as any program running under the name of a specified user has access to anything the user can access. So any program that is downloaded and run by a specific user off the Internet has access to all the same files that the user does. This is how computer worms such as the “I love you” virus are able to access a person’s address book and send themselves out to all the addresses in that address book.

## **Capability Design Decisions**

### ***Pure Capabilities Architecture***

One solution to these problems is to avoid access control lists altogether and use a completely different implementation. One such implementation that was used in some older operating systems like the KeyKos system and is now being used in EROS is the use of capabilities.

In a capabilities system programs are allowed access to objects in that system if they hold a token known as a capability. An object in the system can be anything

including files, other processes, or even events. In this way a capability is somewhat similar to a house or a car key. Only persons holding that key may get into a specific house, and they may only get into that particular house. They cannot, for instance, use that key to get into their neighbor's house. In other words, capabilities only allow access to one specific object in the system. Unlike keys however capabilities cannot be forged or modified. The operating system itself, along with certain hardware measures ensures this protection.

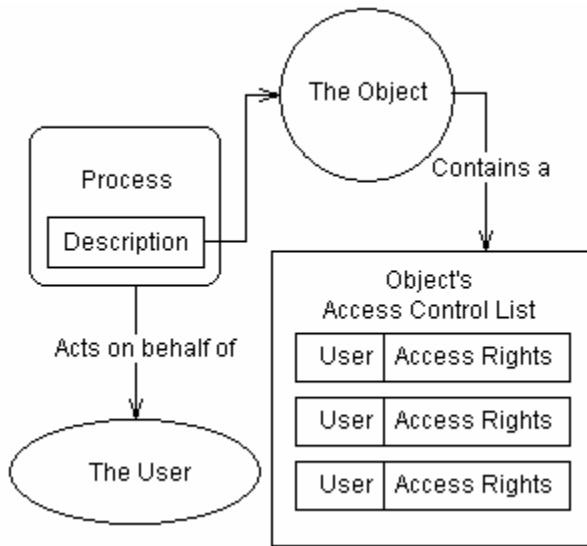
Capabilities do not only represent objects in the system that a particular process has access to. They also represent the access rights that a process may have when accessing an object. For instance, there may be two or more capabilities relating to a specific file, but one of those capabilities may allow its owner to have read-only privileges of the file, while another capability used by a different process but relating to the same file may allow that owner the ability to read and write to the file.

Capabilities are in no way dependent upon who owns them. Capabilities can be both given away and copied. If there is another process that, for some reason, a process wishes to give a capability to the "giving process" can either give the "receiving process" the capability if it is no longer needed, or it can make a copy of that capability and hand it over to the "receiving process." Capabilities are only handed out to trusted processes. Because there is no way for capabilities to

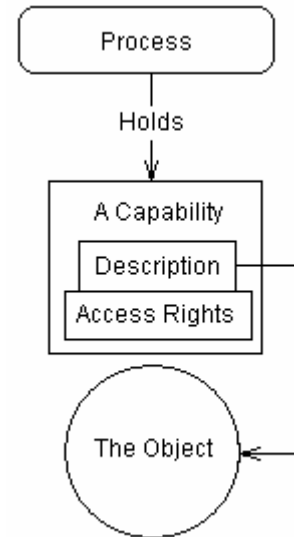
be forged, there is no way for a process to get a capability unless it is given one. Furthermore, it is impossible for a process to have any sort of access to an object in the system unless it has a capability that gives it a specific access privilege to that object.

### ***EROS: A Capabilities-based System***

The EROS development team chose the capabilities system because of its strict security. There is no way for a process to access information that the user does not want it to, unlike in an access control list system. A worm such as the “I love you” virus would have no means of accessing your address book, unless the user of the system gave it access. Capabilities are handed out upon installation of the process. Figures 1 and 2 below diagram the primary difference in functionality between the access control list system and the capabilities system.



**Figure 1:** An Access Control System. The process acts on behalf of the present user. To access an object, the process must first obtain a descriptor to the object. The descriptor can only be obtained if the process is included in the object's access control list.



**Figure 2:** A Capabilities System. The process holds a capability to an object, the descriptor to the object is held in the capability. The process may only access the object to which the capability refers, and it may only perform what the access rights in the capability suggest.

## ***Disadvantages of a Capability-Based System***

After learning about the difference between capability based systems and access control list based systems, one may ask why we have not always been using capability systems since they appear to be so much more secure. Part of the reason is because their performance was terrible in the early operating systems that they were implemented in. These systems that were developed with capabilities were very inefficient because at the time of their development very little was known concerning computer architecture (Shapiro, "What is a Capability"). These systems used capabilities to access places in main memory, which caused them to be quite slow and complex. Unfortunately, these initial

systems had a bad impact upon operating systems designers for quite some time, and many disregarded the system.

## **Universal Persistence**

### ***Introduction to Persistence***

In order for capabilities to be useful, they must be remembered across system restarts. Not only is it important for the capabilities to be saved, but just as in any other operating system there is a whole host of other information must be stored as well. EROS, unlike a multitude of other operating systems, does not use a file system for long-term storage. Instead, EROS uses what is known as persistence.

### ***EROS: An Orthogonally Transparent Persistent System***

Persistence in operating systems terms is defined as the saving of the “state” of the entire system. The “state” of the system is any data present in the system including the user’s data and any data needed by processes and/or the operating system itself. EROS uses a form of persistence known as orthogonal transparent persistence.

Transparent persistence is defined as the saving of the entire state of the system periodically.(Shapiro, “Issues”) In EROS’s case, the state of the system is

recorded every five minutes. According to Shapiro (EROS FAQ), “this seems to be often enough to prevent a major loss.” Orthogonal persistence means that running applications do not need to take any action to become part of the image that is saved. The *entire* system is saved every five minutes.

### **Advantages To Persistence**

The beauty of persistence presents itself when one realizes that they will no longer have to worry about saving their work. If for instance, the power cord to your computer was accidentally knocked out of the computer, or if you lost power via some other means, the next time your computer booted up it would be in relatively the same state as when you left it.

According to Jonathan Shapiro (“Issues”), one big reason why persistence was chosen by the EROS development team was because transparent persistence is the preferred design for capability systems. This is true primarily because persistence eliminates the need for programs to re-obtain their capabilities when the system is started up again. This also eliminates the need for capabilities to be saved when the system is shutdown. (“Issues”)

## **Disadvantages of Persistence**

Upon initial observation, it appears that there could potentially be a multitude of problems that could arise when saving information in this fashion, particularly problems concerning efficiency and implementation. The use of persistence and a complete lack of a file system mean that the development team cannot use many common and accepted implementation processes.

There are specific exceptions to the persistence system in EROS however. Some examples include network connections that must not be retained through restarts due to security reasons and databases that need to be written down explicitly to record transactions.

## **Implementation of Persistence**

According to David Hulse and Alan Dearle “if persistent operating systems are supposed to be anything other than research vehicles, they must both be stable and resilient.” Many early systems developed with persistence were both stable and resilient, unfortunately they faltered elsewhere. Some early implementations of persistence used too simple of an approach to persistence. Whenever the state of the system needed to be saved, they saved everything all at once.

These systems made no attempt to decide between what had changed since the last save in the system and what had not. Consequently these systems were

very slow when saving their data, forcing the user of the system to have to sit and wait while the system completed its saving duties.

EROS takes a much different approach to saving the system state. Rather than saving the entire state all at one time it uses a much more common-sense approach. EROS copies only the dirty pages by marking them as needing to be written. These pages are then written in the background as other processes run.

User data is not the only thing that is saved between restarts, operating system data needs to be stored as well. There is a small problem that is associated with this. While tracking down the operating system data is not very hard to do, because it is constantly changing it must be copied very quickly.

### **Global Transparent Checkpointing**

A major problem with transparent persistence is what to do if the system crashes while you are in the midst of making a copy. EROS handles this by using what its designers call a write-ahead log. Rather than overwriting the previous stored state as the current state is being saved, EROS writes the present state to this log. As soon as the log is finished, then the write-ahead log is transferred to where the state is stored. According to Shapiro, this checkpointing overhead accounts for only 0.3 percent of the overhead in the system, which is hardly even noticeable.

## **Journaling**

The storage of transaction based data, like the data stored in a database must be dealt with in a different way other than by using persistence. There is a possibility that the data could get lost (upon a system crash) before it gets saved when the system's state is recorded. Therefore, the EROS designers decided to provide a slightly different mechanism to handle this. This mechanism is known as journaling.

Journaling allows a process to give a page a specific mark that tells the operating system that the page holding the data should be written down as soon as possible. This allows the operating system to remember such data in case of a system failure.

## **Conclusion**

In his conclusion of the article Understanding the EAL4 evaluation, Jonathan Shapiro explains why operating systems such as EROS are such a good solution to the security problems we face:

EROS... is expected to provide total defense against viruses and malicious code. It won't be compatible, because the most important security problems in Windows and UNIX are *design* problems rather than

implementation problems. In fact, *none* of the viable research efforts toward secure operating systems are compatible with existing systems.

It is because of the design of present operating systems that so many systems can easily be taken advantage of by computer viruses and crackers. By using operating systems with capability-based systems like EROS, such problems would no longer exist. Unfortunately, for an operating system such as EROS or one similar to it to ever become popular, a large amount of software conversions of legacy code would have to be made. Software designed for access control list based systems would not be directly compatible with a capabilities-based persistent system such as EROS.

Because of such incompatibility issues, the EROS development team has decided to create a UNIX environment to run on top of EROS able to run UNIX binaries. Hopefully such environments will give systems like EROS more popular appeal. It is most likely however, that because of such incompatibility issues and conversion difficulties that access control list-based systems will continue to be the standard operating systems for quite some time.

## Bibliography

Hardy, Norm and Shapiro, Jonathan “EROS: A Principle-Driven Operating System from the Ground Up” IEEE Software Jan. 2002: 26 – 33.

Hulse, David and Dearle, Alan. Report on the efficacy of Persistent Operating Systems in supporting Persistent Application Systems. 1 Feb 2003.  
<<http://citeseer.nj.nec.com/353999.html>>

MSDN Library. 1 Feb. 2003. <<http://msdn.microsoft.com/library/>>

Shapiro, Jonathan. EROS: A Platform for Reliable Applications. 1 Feb. 2003.  
<<http://www.eros-os.org/essays/reliability/paper.html>>.

Shapiro, Jonathan. EROS FAQ. 1 Feb 2003. <<http://www.eros-os.org/faq/faq.html>>.

Shapiro, Jonathan. Issues in Persistence. 1 Feb 2003. <<http://www.eros-os.org/essays/Persistence.html>>.

Shapiro, Jonathan. What is a Capability. 1 Feb 2003. <<http://www.eros-os.org/essays/capintro.html>>.

Shapiro, Jonathan. Understanding the Windows EAL4 Evaluation.  
<<http://eros.cs.jhu.edu/~shap/NT-EAL4.html>>.

Stiegler, Marc. Introduction To Capability Based Security. 1 Feb. 2003.  
<<http://www.skyhunter.com/marcs/capabilityIntro/index.html>>.