Member Article: Z. Cliffe Schreuders

Why do we still trust applications with all our authority? - A new usable solution for restricting Applications

Currently one of the greatest threats to security is attacks against client-side applications. For example, PDF readers, and multimedia applications are popular targets; attackers exploit vulnerabilities in applications run by users to gain control of a process that is associated with the identity of a local user. A different yet related threat is that of Trojan horse malware, where a malicious program assumes the identity of a legitimate user, and misuses the user's identity to carryout malicious actions. In both cases, the underlying issue is that applications can not be trusted to act on behalf of the user whose identity the application is associated with. However, most operating systems, such as Windows, Linux and Mac OS, make access control decisions primarily based on the user identity associated with processes, meaning that a process is authorised with all of the user's privileges, and is free to misuse the authority of the associated user to act maliciously.

In a sense this is sometimes considered an authentication problem; that is, the problem is that the wrong applications are being associated with users' identities. However, the very notion of "trusted software" should be considered tenuous at best. Zero-day vulnerabilities and malware are identified at an incredible rate, and users are increasingly executing mobile code and processing untrustworthy data. Relying on reactive approaches, such as patching known vulnerabilities and detecting known malware, is clearly a stop-gap measure (which is regularly circumvented in the current threat climate) in an attempt to address the inadequacies of user-oriented access controls.

Methods do exist for restricting what each application is authorised to perform. Yet these security solutions are rarely deployed. Application-oriented access controls can limit the damage a misbehaving application can cause, by defining the privileges each application is authorised to perform. Examples of models and mechanisms which provide application-restrictions include: chroot, FreeBSD Jails, Solaris Zones, Domain and Type Enforcement (DTE), Role Compatibility (RC), Bitfrost, CapDesk, Polaris, Plash, TRON, Virtual Machines (VMs), POSIX Capabilities, Janus, Systrace, SELinux, and AppArmor. These can be divided into broad categories including: those based on isolation or restriction. The term sandbox is often used to describe isolation-based application-oriented access controls. Systems which completely isolate applications can improve security by strictly confining applications to a limited name-space.

Unfortunately, isolation systems often require extensive redundancy of resources (as each application requires its own copy of particular 'shared' resources), and they do not accommodate typical user workflows; that is, it is not unusual for a user to use many applications to work with a single file: creating, editing, viewing and sharing that file using separate applications. Each of these activities requires different privileges, and this makes it impractical to model using an isolation approach.

Restriction-based application-oriented access controls allow precise policies to define the types of actions and resources which are made available to each application. However, the finer the granularity of privilege assigned, the more detailed and complex policies become. A typical application can require a myriad of resources including various files and types of network access. Two of the most widely deployed application-restriction systems are the Linux security mechanisms SELinux and AppArmor. Systems such as SELinux and AppArmor can enforce least privilege, by permitting programs to only access the resources they require to carry out their legitimate functions, and therefore blocking access to any resources that are superfluous to requirement. However, this comes at the expense of usability, as policies are typically incredibly complex. Both of these systems provide mandatory controls; that is they are required to be configured by administrators. Developing policies for these systems requires expertise beyond that of typical users, and arguably beyond that of the typical system administrator, as policies expose the complexity of the underlying platforms and applications.

This author has devised a unique approach to developing application-restriction policies. This new approach builds policies using reusable abstractions representing the functions that applications are authorised to perform. These policy abstractions are known as *functionalities*, the access control model is known as *Functionality-based Application Confinement (FBAC)*, and the Linux prototype implementation is known as *FBAC-LSM* (Linux Security Module). This approach allows users *and/or* administrators to build policies for applications using easy to understand abstractions. Functionalities which correspond to the legitimate features applications perform are assigned to application policies. Parameters are supplied (mostly automatically using various analysis techniques) to adapt the reusable abstractions to the needs of specific applications. Subsequently these application policies authorise restricted programs to utilise

certain privileges. For example, the Opera application can be assigned the Web_Browser, Email_Client, and Bittorrent_Client functionalities, and parameters are used to define application-specific details. Subsequently, regardless of the presence of vulnerabilities Opera cannot act beyond the privileges required to carry out the legitimate tasks which functionalities authorise the application to perform.

The FBAC model provides discretionary and mandatory controls. The security goals of users and those of administrators are simultaneously enforced. Users can protect their own files from potentially malicious application processes, and administrators can limit which programs users can use and what they can do with them, and can restrict highly privileged services and processes (such as system daemons and setuid programs) from misbehaving.

Another of the unique features provided by FBAC-LSM is the ability to construct policy without first running the application to be confined. SELinux and AppArmor, like most application-restriction systems, rely on learning modes to generate detailed policy, and users must vet these rules to ensure the policy which is created is in accordance with their security goals. On the other hand, although they are available, FBAC-LSM does not rely on learning tools to create policies to confine applications.

A usability study was conducted to evaluate the new approach. The study compared SELinux, AppArmor, and FBAC-LSM. Each of the 39 participants used all three systems in a randomly allocated order. Amongst the factors measured were user satisfaction and the resulting overall exposure to risk left after participants attempted to use each of the systems to restrict two applications. Satisfaction was measured using the System Usability Scale (SUS) which gives a score out of 100. At the conclusion of the usability study, SELinux received a SUS score of 34.5, AppArmor 54.9, and FBAC-LSM 70.2. Exposure to risk was measured by the number of certain security-sensitive resources the programs could access after participants had attempted to confine them. SELinux on average did not restrict access to 43 of these security sensitive resources, AppArmor 30, and FBAC-LSM 14. In both cases FBAC-LSM showed statistically significant improvements compared to both alternative systems. These results demonstrate that the new approach resulted in improved usability and security. Amongst other things, the usability study also showed that many users (including some Linux system administrators and security professionals) do not have the expertise required to accurately vet rules generated through learning modes. A functionality-based approach for restricting applications has shown potential for improving the usability of application-oriented access controls, and providing increased security to end-users and systems. Perhaps one day soon the norm won't be simply to trust the applications that we run with all of our authority.

Z. Cliffe Schreuders
Z. Cliffe Schreuders is a PhD candidate and casual lecturer at Murdoch University, Western Australia. Recently Cliffe has presented at academic and Linux conferences in England, Portugal, New Zealand and Australia. His current research aims to provide more useable application restrictions.

FBAC-LSM is in development and is available as free open source software: http://schreuders.org/FBAC-LSM

More information:
http://schreuders.org
Contact:
z.cliffe@schreuders.org

Acknowledgements: Thanks to my supervisors Tanya McGill and Christian Payne. Thanks also to Steve Simpson and Steve Schupp for soliciting and reviewing this article.

Do you have any thoughts on this article? Send them to the "Letters to the Editor"; news@aisa.org.au