Exu - A System for Secure Delegation of Authority on an Insecure Network

Karl Ramm - Massachusetts Institute of Technology
Michael Grubb - Duke University

Exu – A System for Secure Delegation of Authority on an Insecure Network

Karl Ramm – Massachusetts Institute of Technology
Michael Grubb – Duke University

ABSTRACT

Administration of a large and complex system poses several problems: Usually, some tasks must be delegated due to lack of qualified or trusted staff, and some tasks must be automated. In many cases, some parts of the task might need special credentials, such as Kerberos tickets or AFS tokens, that may not necessarily be easily available to the person executing the task. The problem is that most systems divide users into two groups: haves and have nots, and provide no mechanism for finer-grained access control. In addition, the tasks executed must be carefully recorded for possible later auditing. Earlier solutions, such as the setuid bit, Moira, ADM, and sysctl, can be used to accomplish this, either in a limited or dangerous (in the case of setuid) fashion. Exu proposes to solve the problem via secure, authenticated connection to a server with full authentication that can cause things to happen in real time.

Problem

In a system with a very large number of users and a very small number of administrators, "superuser" access is of necessity hoarded. If such a system is critical to the operation of the enterprise (in this case, a university) the need to be careful is even more crucial. On the other hand, work-a-day system administration tasks such as changing forgetful users' passwords for them, new user account creation, and the maintenance of mailing lists are nice to farm out when you spend your days putting out fires and placating angry users. Additionally, certain tasks often end up being performed by hand or with ugly kludges involving frightening authentication structures, because authorization tends to be an all-or-nothing proposition, dividing users into haves and have-nots.

Some systems, such as TransArc AFS [AFS], partially address this problem with access control lists (ACLs), allowing several different entities to have varying amounts of access to different services; there's an ACL for "superuser" filesystem access, an ACL for doing fileserver maintenance, and another ACL for altering the kaserver (Kerberos) database. This begins to break down when you want a certain person to be able to do certain fileserver operations such as moving files between certain disks, or back them up, without giving them carte blanche for the rest of the system.

In the situation at Duke University's Office of Information Technology, with three system administrators, fourteen server machines, over one hundred workstations, and over nineteen thousand user accounts, there was very little time to develop the infrastructure for the "right solution" to the problem, and things needed to be implemented very quickly. Thus, any labor saving device that was employed had to be very flexible and quickly extensible.

This situation led to the development of Exu, a service designed to delegate authority in a controlled fashion and smoothly automate system administration tasks requiring authentication to multiple services. A third, emerging motivation is the elimination of IP-address based authentication to counter emerging security threats on the Internet. [IPSecurity] Exu rejects the traditional all-or-nothing type of security, and the not quite as traditional ACL based security, for procedural security, where a completely configurable piece of code determines the level of authorization.

Other Approaches

Setuid Bit

One of UNIX's more interesting innovations was a per-file bit that allowed executables to assume the authentication of their owner. This is used by utilities such as the local-file password-changing program to provide controlled write access to security-critical files. On a typically less than totally secure network, the setuid bit leads to a variety of problems: in cases involving remote file systems, one's trust of the file server must be contemplated, given that it probably could be easily spoofed, and if the local machine has sufficient privileges, the security of all the machines that trust the file server (which probably includes the file server) is predicated on the local machine's security. Also, applications of the setuid bit do nothing to address the problem of monitoring. Given that most workstations are interconnected via bus networks such as Ethernet, encryption and a method of authentication that does not require secret information transmitted in the clear is necessary. Such a system is MIT's Kerberos [Kerberos], which uses shared secrets to authenticate connections and securely transport randomly-generated session keys.

MIT Athena SMS

The Athena Service Management System [Moira] is fundamentally a database driver: It uses a commercial database (RTI Ingres) to store information concerning workstations, server configurations, users, and printers, with an input driver that handles access control, information queries and updates, as well as an output driver that actually updates the configurations and password files every night. Authentication and wire security are handled via Kerberos. Although there are many interesting applications that talk to it, the server itself doesn't do anything more than maintain the database. The program itself is not easily extensible; if you want to control a new set of files, you need to write the output stages and the input stages in C.

CMU ADM

The CMU ADMinistration server [FlexAdmin] is on the opposite end of the spectrum. It essentially consists of a Scheme interpreter that can run two modes: user mode and privileged mode. The privileged mode, which is normally restricted to system administrators, has access to a variety of primitives, in the canonical example being AFS server control operations, and can define functions which can execute these dangerous primitives *even in user mode*. The catch is, of course, that the interpreter in user mode cannot modify these functions. The idea is that these functions are written in a manner so as to guarantee their security – that is, check who's calling them before they perform the operation. Authentication and wire security is, as usual, provided by Kerberos. Unfortunately for our situation, an incomplete implementation and lack of provision for a database package hampered extensibility and usefulness, although it does have procedural access control.

Sysctl

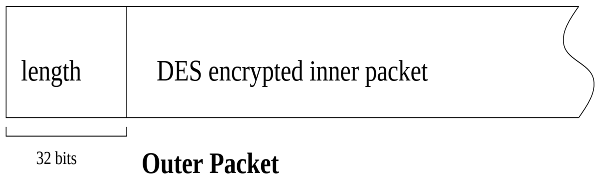
The sysctl package from IBM's Project Agora has a similar idea; however, it uses Tcl [Embeddable], a scripting language that takes syntactical concepts from the Bourne shell, C, and LISP, giving each procedure either one of three security levels: unauthenticated (for people with no Kerberos authentication), authenticated (for people with), and trusted (for people who are authenticated and on the list of trusted users), or a specific ACL. Although it should be possible to do procedural security in this context, it is not directly supported by the paradigm. Sysctl handles authentication and wire security via Kerberos.

Exu

Exu (pronounce *ee-shoo*) consists of a single-threaded server process, running out of *inetd*(8). A server will run for each client program in order to simplify security concerns at the expense of a possible performance bottleneck. Exu is extensible via Tcl. Tcl is also easily extensible; there are a variety of packages that add database functionality, access to most of the UNIX system call interface [XTcl], and even an extension called expect [Automate], which permits extensive control and management of subprocesses, even those that think they're interactive. The Exu implementation also adds commands to manage various AFS entities. However, the most important extension to Exu is SafeTcl [MailEnabled], which consists of a pair of Tcl interpreters, the restricted interpreter and the unrestricted interpreter. The restricted interpreter has all possible "dangerous" commands (file manipulation, program execution) removed, and the unrestricted interpreter has all the interesting extensions (expect, extended Tcl) added, as well as primitives for allowing *unrestricted* interpreter procedures to be executed in the *restricted* interpreter. This produces an effect similar to the privileged/unprivileged modes in ADM, above, and allows the same sort of procedural access control, with the extensibility allowed by the TCL in sysctl. These procedures are organized into groupings called libraries, which can be loaded with the loadlibrary command. Exu, of course, handles authentication and wire security via Kerberos.

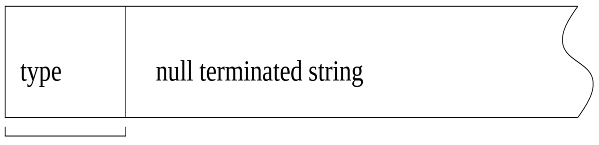
Protocol

The Exu protocol is conceptually very simple: The client creates a TCP connection to the server, exchanges Kerberos authentication information via the *krb_sendauth* and *krb_recvauth* library routines, and then the client and server send logical "packets" back and forth over it. Each packet consists of an "outer packet" and an "inner packet". The outer packet consists of a 32-bit network byte order integer which represents the length of the rest of the packet, followed by that number of bytes which represent the inner packet DES encrypted with the Kerberos session key.



Outer Packet

The inner packet consists of another 32-bit network byte order integer followed by a null-terminated string. The integer is the "message type" code, and the string is the message itself.



Inner Packet

At the moment, the message type number is only used for Tcl return codes; any string sent to the server is presumed to be a Tcl command string. In the future, the client to server message type field may be used for things like client-side logging of information, and the other direction may be used for server output.

Client Library

The Exu client library is conceptually very simple, with only three functions: *exu_open* and *exu_close()* which open and close a connection to the Exu server, and *exu*, which sends a command to the Exu server for execution. At the moment, the syntax of these functions is such that they can be directly linked into a Tcl interpreter, although a future version will export C bindings as well as Tcl bindings.

Example – kasexam

Here is an example of some code for a simple Exu client using the client Tcl bindings. Note that the switch between client execution and server execution is very clean looking, because they're both Tcl code.

Sample Code – kasexam.tcl

```
#!/path/to/exush

# Open Exu server
set server [exu_open blake7.duke.edu]

# exu_open returns name of
# server principal
puts "Remote: $server"

exu {
  loadlibrary kas
}
# do a remote loadlibrary command
# (loadlibrary is a renamed
# SafeTcl_loadlibrary)

puts [exu "examine [lindex $argv 0]"]
# retrieves and prints out ugly pile
# of information

exu_close
# close the server connection
```

Sample Run

Here is a sample run of the above code:

```
% kasexam hiro
Remote: exu blake7 ACPUB.DUKE.EDU
1 normal -1 785454095 admin {} \
776011185 90000 0
%
```

Server code

Here is an extract from the kas "library" of the functions that were executed on the server in the above example:

```
# use a regular expression to turn
# name.instance@REALM to
# {name instance REALM}
proc parse_princ {name} {
  regexp {([^.@]*)(\.([^.@]*)|)\
    (@(.*)|)} $name foo m1 m2 m3 \
    m4 m5 m6
  set l {}
  lappend l $m1
  lappend l $m3
  lappend l $m5
  return $l
}

# retrieve incidental information
# from Kerberos Authentication
# Service database; make sure that
# the caller is karl@ACPUB.DUKE.EDU
proc examine {user} {
  set p [parse_princ $user]
  if {[exu_client_principal] ==
       {karl {} ACPUB.DUKE.EDU}} {
    return [kas_examine [lindex $p 0] \
      [lindex $p 1]]
  }
}
declareharmless examine
```

Note the *exu_client_principal* Tcl command, which returns to the calling function the Kerberos principal that the client authenticated as. Also note the *declareharmless* command, which is used to make a function available to the restricted interpreter.

Choice of Tcl

Tcl has many advantages in this sort of application: it is well supported and there is a lot of code floating around for it because for anyone reasonably proficient in C or Bourne shell programming, Tcl code can be generated very rapidly. The existence of extensions such as Expect and SecureTcl also makes it very useful for this sort of application.

Unfortunately, many of the features of Tcl which make it easy to program mask potential flaws in its design, particularly its quoting. Any time you use the `eval` command, unless you are being very careful about where you got your input from, you run the risk of executing arbitrary code. An example (and this *is* somewhat contrived) follows:

```
proc runcommand {a} {
    # where a is actually an argument
    # to another function, doit
    eval doit $a
}
proc doit {args} {
    foreach i $args {
        puts $i
    }
}
```

If you send it

```
runcommand {a b c}
```

where {a b c} is the quoted string "a b c", you get

```
a
b
c
```

instead of

```
a b c
```

as you might expect. There are ways to program around this, but it's merely annoying, right? Wrong. It's dangerous, because if you run

```
runcommand {foo [puts gotcha]}
```

you don't get

```
foo
[puts
gotcha]
```

as you might now expect, but

```
gotcha
foo
```

Because the Tcl interpreter *in the context of the doit runcommand procedure* executed the `puts gotcha` and substituted the result (an empty string). This could have suboptimal results if you happen to be running as root on a sealed machine at the time. As mentioned before, there are ways to program around this, but it often requires some thought.

Another large drawback of Tcl is the fact that it does not support (at the moment) standard dynamic loading, which means that when you incorporate all your interesting extensions into your server interpreter, it's not so lightweight anymore.

If there were a nice clean language, with somewhat stronger typing and somewhat less abstruse quoting conventions and a standard mechanism for dynamic loading (and a reasonable security model), then it would probably be a better choice for this sort of thing.

Applications

homepage

One of the flagship uses of Exu at Duke is a service called *homepage*. Homepage allows users to remotely add, change, and remove their own entries in the campus web server's listing of web home pages, without having login access to the web server or access to its filesystem. The Exu server software has proven to be very reliable under Solaris and Ultrix. The homepage client side is also implemented in Tcl and offers to set up the user with a blank template home page if they don't have one yet.

Future Thoughts

In addition to further implementation of AFS Tcl extensions to the Exu server library, Exu is well-suited to delegated management of portions of large databases, such as are used in site-wide account administration and large-scale mailing list management. There also needs to be an unauthenticated mode (probably without encryption), and a library for handling ACLs. Another interesting application might be for non-privileged users maintaining software in replicated AFS volumes. In some environments, where privacy is a concern and web page access statistics might be considered privileged information, users could use Exu to retrieve usage statistics on their pages.

User database

Having a master user database from which all password tables, NIS maps, NIS+ maps, Hesiod tables, etc. are updated and generated from is an old idea. We envision a database with the person's account name, uid number, real name, student/employee id number (SSN), home directory, mailbox location and special access privileges. This can be implemented trivially with any of the several Tcl database extensions. Alterations could be pushed out nightly, or be instantly updated depending on the type and priority of the update, and on the service being updated. Exu could thus handle all phases of user account creation and maintenance.

If user accounts are managed entirely by Exu, it becomes simple to leverage that into a pretty Tk-based user management tool. Such a tool simplifies administrative tasks for non-technical staff, and is very good for impressing the natives.

This approach can also provide working *chfn*(1) and *chsh*(1) commands to allow users to alter their own full name and login shell entries in the master database which otherwise is problematic on distributed or otherwise heterogeneous systems (such as Ultrix Hesiod).

Mailing List Management

Exu could also be used to help manage a mailing list server. List owners could use a simple line-oriented menu interface or a Tk-based GUI to perform administrative functions such as altering descriptions, changing the list password, and insertion and deletion of list members. Mailing list subscribers with Kerberos principals in the local realm would similarly be able to use an Exu client to browse available lists and subscribe and unsubscribe. Since configurational and administrative tasks are notoriously CPU-intensive in some widely-used mailing list management server software, Exu could easily be a performance boost to a mailing list service if used in this way.

Software Availability

The current Exu distribution is available by anonymous ftp from ftp.duke.edu in */pub/exu*.

Conclusions

Exu should be a highly useful tool to any sufficiently overworked set of administrators. The design is optimized for flexibility, not performance; it strives for maximizing the capability/complexity ratio. The approach is new; long term performance in a production environment is yet to be measured.

Author Information

Karl Ramm is a systems programmer for MIT Information Systems, where he helps keep Athena from falling over. He can be reached via U.S. Mail at Room E40-342B, Massachusetts Institute of Technology; 1 Amherst St.; Cambridge MA 02139, while he can be reached electronically at kcr@mit.edu. He has written five different mail reading programs and has never written a windowing system.

Michael Grubb is a systems programmer for Duke University's Office of Information Technology. He is also a licensed attorney. He can be reached by post at Box 90132, Durham, NC 27708-0132 USA or by email at mg@ac.duke.edu.

References

[IPSecurity] Morris, R.T., *A Weakness in the 4.2BSD Unix TCP/IP Software*, Computing Science Technical Report No. 117, AT&T Bell Laboritories, Murray Hill, New Jersey, 1985.

[Kerberos] Steiner, Neuman, Schiller, "Kerberos: An Authentication Service for Open Network Systems", *USENIX Technical Conference*, Dallas, Texas, Winter 1988.

[Moira] Rosenstein, Geer, Levine, "The Athena Service Management System", *USENIX Technical Conference*, Dallas, Texas, Winter 1988.

[FlexAdmin] VanRyzin, "Flexible Administration for AFS", *AFS Users Group*, Fall, 1991.

[SysCtl] DeSimone, Lombardi, "Sysctl: A Distributed System Control Package", *LISA VII*, Monterey, California, 1993.

[Embeddable] Osterhout, "Tcl: An Embeddable Command Language", *USENIX Tehnical Conference*, Washington, D.C., Winter 1990.

[XTcl] Lehenbauer, Diekahns, "Extended Tcl", ftp://ftp.neosoft.com/languages/tcl/distrib/tclX7.4a-b2.tar.gz

[Automate] Libes, Don, "Using expect to Automate System Administration Tasks", *LISA IV*, Colorado Springs, Colorado, 1990.

[AFS] Howard, John, "An Overview of the Andrew File System", *USENIX Technical Conference*, Dallas, Texas, Winter 1988.

[MailEnabled] Borenstein, Rose, "MIME Extensions for Mail-Enabled Applications: application/Safe-Tcl and multipart/enabled-mail", unpublished draft distributed with SafeTcl, ftp://ftp.ics.uci.edu/pub/mrose/safe-tcl/safe-tcl-1.2.tar.Z