Tools to Administer Domain and Type Enforcement

Serge Hallyn and Phil Kearns – College of William and Mary

ABSTRACT

Domain and Type Enforcement (DTE) is a mechanism that can be effective in providing extremely fine-grained mandatory access control in complex networked systems. Unfortunately, this level of control comes at a price: the configuration of a DTE policy is usually in the form of a dense ASCII file which does not lend itself to an understanding or administration of the policy itself. We describe a set of graphical tools which aid in that understanding and administration.

Introduction

Precise access control is generally acknowledged as a means of providing greater system security. The gross access control of the standard Unix scheme of twelve protection bits specifying access for the owner of a binary, the owner's group, and the rest of the world has lead to two points of vulnerability in modern Unix systems: the existence of "set user id" (root) binaries and the root account being exempt from all access restrictions. Responses to these vulnerabilities include techniques such as access control lists (Linux-ACL [8] is a recent development in access control lists) and capabilities (such as POSIX capabilities [7] as partially implemented in the Linux v2.2 kernel). Each technique attempts to allow finer-grained access control than the standard Unix mechanism, especially for processes running with root privilege. For example, using POSIX capabilities, the `talkd` service may only need access to restricted network ports, so that it may be started with only the `CAP_NET_BIND_SERVICE` capability. If `talkd` is later compromised, the attacker's privileges on the system are still very limited, despite being root on the system.

Type Enforcement was introduced by Boebert and Kain [4] in 1985 as a method of implementing integrity systems without relying on a trusted user. It labeled objects as well as subjects, and specified access from subjects to objects and subjects to subjects in two matrices. Subject labels were called domains, and object labels were called types. Subjects to object access could be read, write, and execute. Type Enforcement was implemented first in the Secure Ada project (LOCK) [10] , and later by TIS in Trusted XENIX [1] . Secure Computing still uses TE in its Sidewinder firewall product [6] .

Domain and Type Enforcement was first presented by O'Brien and Rogers [10] and is an extension of TE. It differs mainly in specifying policies in an intuitive policy language rather than using two matrices. Domain-to-domain transitions are allowed by the execution of special binaries designated as entry points to the target domain. TIS did the first Unix implementation of DTE [3] on a BSD system.

Access control techniques similar to TE and DTE are a continuing source of research. Most notably, NSA's Security-Enhanced Linux project [2] uses a mechanism much like TE and DTE to control the access rights for "process labels" (similar to domains).

The Problem

We have implemented DTE for Linux 2.3 and 2.4. Administering our implementation of DTE consists of editing a plaintext policy file. The system must be rebooted to effect the changes. The policy file consists of several sections. The first section enumerates the types and domains. Next are specified the default type for the file system root ("/") and its children, and the domain in which to run the first process. This is followed by a detailed definition of each domain. For each domain, we specify the entry points, permitted type access, permitted domain transitions, and permitted signals to processes in other domains. The last section lists the type assignment rules. See the sample policy file in Listing 1.

This policy file should be viewed as part of a much larger file which defines the DTE policy for a networked Unix system. It was written primarily to demonstrate how DTE could defend against root compromise via the recent `wu-ftpd` exploit [5]. The ftp daemon provided with Red Hat Linux 6.2 contains a well-known string format vulnerability that allows any remote (or local) user to obtain a root shell. This policy file was implemented in a version of Linux 2.3 which had been modified to support DTE [9] and was shown to eliminate root compromise through the `wu-ftpd` vulnerability.[1]

The policy defined in Listing 1 attempts to confine the access rights of the ftp daemon (/usr/sbin /in.ftpd) so that a successful overflow exploit against it does not compromise the rest of the system – certainly, we do not want to allow the attacker to run a shell with root privileges. Lines 19-21 of the policy define the ftp_d domain in which the daemon is required to execute. Note that processes in the ftp_d

[1] See the cited paper for details of the syntax.

domain may only execute binaries beneath /lib and /home/ftp/bin, in addition to /usr/sbin/in.ftpd itself. This follows from the rules

```
rxd->lib_t
rxd->ftp_xt
```

on line 20, together with the type assignments on lines 29, 30, and 34. If there is, in fact, an exploitable overflow in the ftp daemon, our policy will ensure that the daemon cannot execute a shell, which is assumed not to reside in a location accessible for execution to /usr/sbin/in.ftpd .

The ability to use DTE to confine the access rights of an important process like the ftp daemon, to which we want to allow relatively easy network access, clearly limits the damage that can be done if that daemon is compromised. This power comes at an obvious price: the DTE policy file is dense, relatively unstructured, text. In some sense, this is a natural consequence of fine-grained access control. We simply have a lot to specify when we specify the DTE policy for a complex system with many domains and types. However, the impact of an error in this file may be disastrous. We address this problem with a tool that takes the DTE policy file as input and produces a plugin for a Perl/Tk graphical analysis tool.

Our Solution: DTEedit/DTEview

DTEedit is a DTE policy file editor that understands, and enforces, proper syntax of the policy file. DTEedit produces a well-formed policy file and (in a separate file) a representation of the policy expressed as Perl code. DTEview is a Perl/Tk program that assists in the administration of DTE. Its representation of the DTE policy is provided by the Perl plugin output by DTEedit. DTEview internally treats the policy as a directed graph in which nodes represent types and domains. An edge from a domain to a type is labeled to indicate the appropriate access rights; an edge from a domain to a domain indicates a possible domain transition through an entry point or, although not relevant in this example, signals allowed from one domain to another. DTEview takes various closures of the

```
01 # ftpd protection policy
02 types root_t login_t user_t spool_t binary_t lib_t passwd_t shadow_t dev_t \
03       config_t ftpd_t ftpd_xt w_t
04 domains root_d login_d user_d ftpd_d
05 default_d root_d
06 default_et root_t
07 default_ut root_t
08 default_rt root_t
09 spec_domain root_d (/bin/bash /sbin/init /bin/su) (rwxcd->root_t rwxcd->spool_t \
10     rwcdx->user_t rwdc->ftpd_t rxd->lib_t rxd->binary_t rwxcd->passwd_t \
11     rxwcd->shadow_t rwxcd->dev_t rwxcd->config_t rwxcd->w_t) (auto->login_d \
12     auto->ftpd_d) (0->0)
13 spec_domain login_d (/bin/login /bin/login.dte) (rxd->root_t rwxcd->spool_t \
14     rxd->lib_t rxd->binary_t rwxcd->passwd_t rxwcd->shadow_t rwxcd->dev_t \
15     rxwd->config_t rwxcd->w_t) (exec->root_d exec->user_d) (14->0 17->0)
16 spec_domain user_d (/bin/bash /bin/tcsh) (rwxcd->user_t rwxd->root_t \
17     rwxcd->spool_t rxd->lib_t rxd->binary_t rwxcd->passwd_t rxwcd->shadow_t \
18     rwxcd->dev_t rxd->config_t rwxcd->w_t) (exec->root_d) (14->0 17->0)
19 spec_domain ftpd_d (/usr/sbin/in.ftpd) (rwcd->ftpd_t rd->user_t rd->root_t \
20     rxd->lib_t r->passwd_t r->shadow_t rwcd->dev_t rd->config_t rxd->ftpd_xt \
21     rwcd->w_t d->spool_t) () (14->root_d 17->root_d)
22 assign -u /home user_t
23 assign -u /tmp spool_t
24 assign -u /var spool_t
25 assign -u /dev dev_t
26 assign -u /scratch user_t
27 assign -r /usr/src/linux user_t
28 assign -u /usr/sbin binary_t
29 assign -e /usr/sbin/in.ftpd ftpd_xt
30 assign -r /home/ftp/bin ftpd_xt
31 assign -e /var/run/ftp.pids-all ftpd_t
32 assign -r /home/ftp ftpd_t
33 assign -e /var/log/xferlog ftpd_t
34 assign -r /lib lib_t
35 assign -e /etc/passwd passwd_t
36 assign -e /etc/shadow shadow_t
37 assign -e /var/log/wtmp w_t
38 assign -e /var/run/utmp w_t
39 assign -u /etc config_t
```

Listing 1: Sample policy file.

graph representation of the policy in response to user queries. In this paper we restrict the description of DTEview to those features needed to detect a problem with the sample policy file listed above, although clearly the tools will analyze any DTE policy.

We begin by showing a DTEview file type analysis window in Figure 1. We start a directory traversal at /home/ftp. This results in a display of the contents of the /home/ftp directory, which in this case are the bin/ and incoming/ subdirectories. Each element of the display is labeled with its DTE type. By left clicking on the arrows pointing out of the subdirectory elements, one may traverse the hierarchy as deeply as needed, seeing associated types for all file system objects. The most relevant parts of this display for our purposes are that:

- the directory subtree rooted at /home/ftp/bin is of type ftpd_xt, and hence, the ftp daemon can execute any binary beneath /home/ftp/bin, and
- the /home/ftp directory is of type ftpd_t.

The next DTEview window is the process tree analyzer shown in Figure 2. When invoked, the tool starts with the domain of /sbin/init and shows the one-step domain transitions possible by the entry point mechanism. In our example, the top two rows of rectangles are drawn. Each rectangle represents a process capable of running in the system, labeled with the domain in which it runs. Left clicking on a rectangle in the second row will show, in similar form, domains into which a process represented by the rectangle may transition through an entry point. No such transitions are shown in Figure 2. Right clicking on a rectangle shows the types to which the process/domain has access rights. The types are represented by the circles on the third row of Figure 2; this row was obtained by right clicking on the rectangle associated with the ftp daemon, and filtering the display to show only types to which the process has directory write (c) access.

Note that the ftp daemon, running in the ftpd_d domain, has ‘‘rwcd’’ access rights to any files or directories of type ftpd_t. Clicking on the ftpd_t icon now brings up the window shown in Figure 3. We see in the last two lines that ftpd_t is assigned to /home/ftp and all its children. This is significant because the directory write access to /home/ftp means that ftpd_d can rename and replace /home/ftp/bin, under which it has execute access. If the daemon is susceptible to an overflow-based exploit, for example, it is possible to replace the /home/ftp/bin directory with one populated with Trojan Horse binaries. Since the /home/ftp/bin directory is intended to contain service binaries, such as ls, for the ftp daemon, it is easy to imagine a replacement ls which replies with the contents of /etc/passwd and /etc/shadow to enable password cracking off-line. Although this is not as immediately serious as a root shell, it still represents a substantial security threat for a real system.

The above problem is an error pattern: if a domain can change binaries or directories which contain binaries to which the domain has execute access rights, then processes in that domain cannot have their

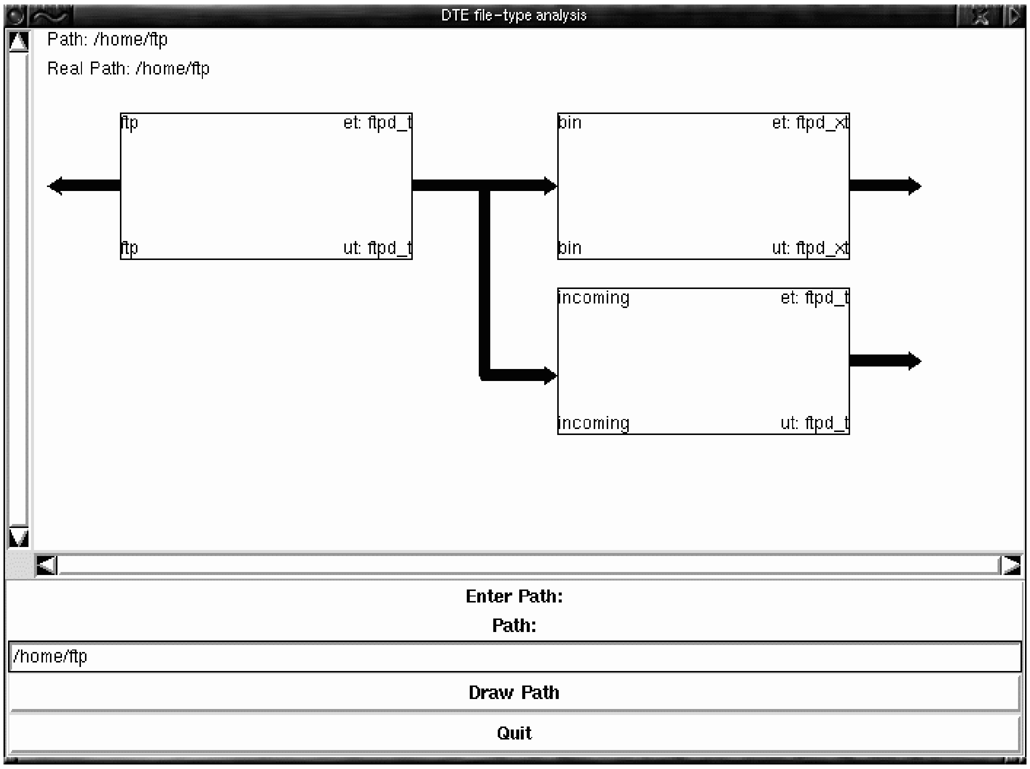

Figure 1: File type analyzer window for sample policy.

execution behavior effectively constrained. When DTEview is started, it runs a check for such a situation for all domains listed in a control file. If the error pattern is found, a popup window notifies the user. Figure 4 shows the popup window that appears when DTEview is applied to the sample policy.

Reachability

Treating the DTE policy as a graph allows us to make queries concerning access by domains to types and domains. Restricted to looking at the policy file, even a simple case such as asking whether a domain has create access to a directory can become tedious.

The task is complicated significantly when we consider that a domain may be able to access a file after performing a domain switch. For instance, a compromised web server may be allowed to write but not execute /bin/sh. If it can subsequently switch to a user domain, for instance, to run a CGI script, and execute what it wrote from the user domain, this could be just as bad. DTEview can answer the question ‘‘Can a process under domain daemon_d write /bin/sh within two domain transitions?’’

In order to aid a system administrator in analyzing query results, one can specify labels on domain transitions. For instance, the security group may have written a login daemon that uses Java Rings [11] for authentication, and which they have verified to be correct. They can then label domain transitions out of login_d, provided it was entered through /bin/login.javaring, with a string indicating the user’s identity has been verified by a trusted program. In a subsequent query as to whether a web server is able to enter the root_d domain, the security administrator can ignore domain transition paths which contain such a transition, as this transition acts as a barrier to any unauthorized users.

Using these features of DTEview, administrators can begin to verify specific assertions concerning the policies which they are writing.

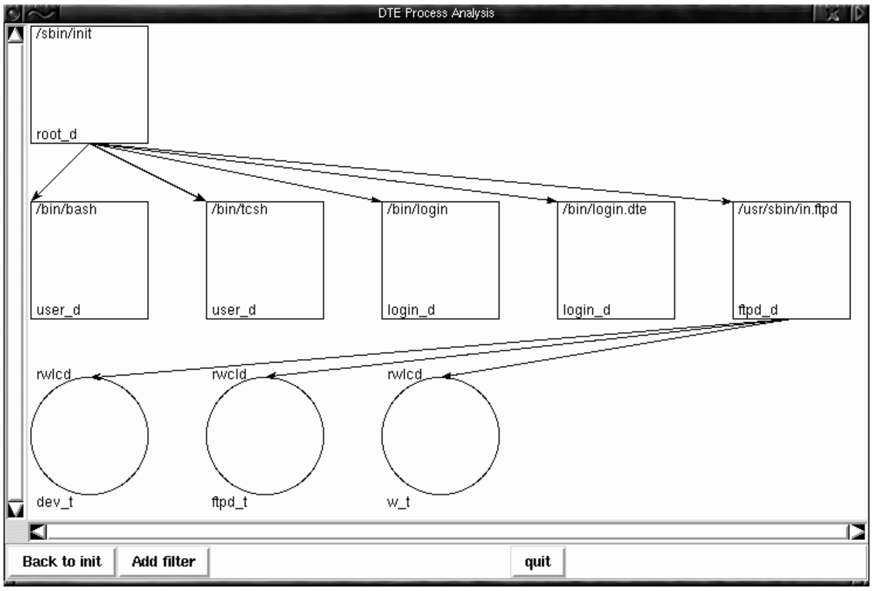

Figure 2: Process tree analyzer window for sample policy.

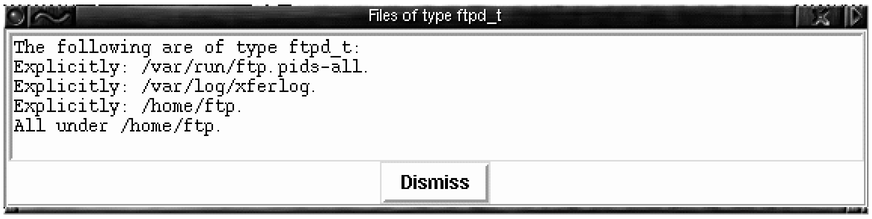

Figure 3: Window showing type assigns for ftpd_t.

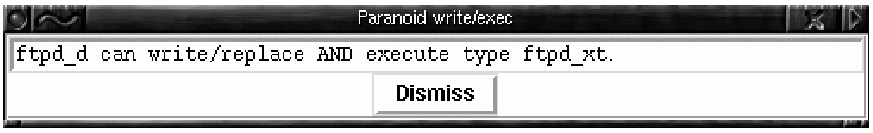

Figure 4: Popup window.

Concluding Remarks

Fine-grained access control, as provided by DTE, has obvious advantages in securing a modern networked system. It also has the obvious disadvantage of requiring a detailed and lengthy specification of the access control policy. An obscure text-based configuration file does not lend itself to understanding or debugging an access control policy. We have given an overview (by example) of a technique which models the policy in graph-theoretic form providing a means for automated analysis and graphical display of important aspects of the policy. Equally important is the fact that the model also provides a formal framework within which we can state and prove theorems about the logic behind the analysis and displays presented by DTEview.

Future Work

DTEedit and DTEview are a good start at easing security policy administration. Work is also under way to aid in constructing whole policies from several coherent pieces (modules), to offer still more intuitive ways of representing policies for analysis, and to provide more powerful means for proving policy properties. Naturally, this is in parallel to the continued implementation and maintenance of DTE itself, which is currently being ported to work with the Linux Security Module [12] project.

Availability

DTEedit and DTEview are available under http://www.cs.wm.edu/~hallyn/dte.

Thanks

The authors wish to thank the the USENIX organization, as well as the anonymous reviewers and Adam S. Moskowitz for helpful comments.

Author Information

Serge Hallyn <hallyn@cs.wm.edu> is a Ph.D. candidate at the College of William and Mary, whose research concerns systems security.

Phil Kearns <kearns@cs.wm.edu> is on the faculty of the Department of Computer Science at the College of William and Mary. His current research interests include distributed systems and operating systems.

References

[1] National Security Agency, "Evaluated Platforms: Trusted Xenix," http://www.radium.ncsc.mil/tpep/epl/entries/CSC-EPL-92-001-A.html .

[2] National Security Agency, "Security-enhanced Linux," http://www.nsa.gov/selinux , 2000.

[3] Lee Badger, Daniel F. Sterne, David L. Sherman, Kenneth M. Walker, and Sheila A. Haghighat, "A Domain and Type Enforcement Unix Prototype," *Proceedings of the Fifth USENIX UNIX Security Symposium*, June, 1995.

[4] Boebert, W. E. and R. Y. Kain, "A Practical Alternative to Hierarchical Integrity Policies," *Proceedings of the National Computer Security Conference*, Vol. 8, Num. 18, 1985.

[5] CERT, "Two Input Validation Problems in ftpd," http://www.cert.org/advisories/CA-2000-13.html, July, 2000.

[6] Secure Computing, "Type Enforcement Technology for Access Gateways and VPNs," http://www.secure-computing.com .

[7] POSIX Security Working Group, *POSIX System API Amendment 1003.1e: Protection, Audit, and Control Interfaces* (withdrawn), October, 1997.

[8] Andreas Grunbacher, http://acl.bestbits.at .

[9] Hallyn, Serge and Phil Kearns, "Domain and Type Enforcement for Linux," *Proceedings of the Atlanta Linux Showcase*, 4, October, 2000.

[10] O'Brien, R. and C. Rogers, "Developing Applications on Lock," *Proceedings of the National Computer Security Conference*, Vol 14, 147-156, October, 1991.

[11] Dallas Semiconductor, "Java-powered Ring," http://www.ibutton.com/store/index.html#jring .

[12] Various, "Linux Security Module Project," http://lsm.immunix.org, 2001.