Mandatory Access Control
Domain and Type Enforcement (DTE)

Simon Foley

February 2, 2016

Domains

Every subject (process) has an associated *protection domain*.

Domains entered by executing any program associated with that domain (like Unix suid mechanism).

Domains are like sandboxes that are used to limit the access that a program has to resources.

Example: `DOMAINS = {internet,system,COTS}`

- domain `internet` is used to limit access to resources by programs that access the Internet: eg, program `firefox` runs in domain `internet`;
- domain `system` is used for any system program: eg, program `/bin/passwd` runs in domain `system`;
- domain `COTS` is used for Commercial Off-The-Shelf programs: eg `openOffice` runs in domain `COTS`

Types

Every object has a type.

Within a domain, certain *types* of objects may be accessed.

Example, `TYPES = {critical,user,untrusted}`.

- operating system files have type `critical`, eg, `/etc/passwd`.
- user files may have type `user`, eg, `/myfile.doc`, or
- user files may have type `untrusted`, eg, `/.netscape/cache`.

Program files (executed by a subject) will also have a type, eg, `/usr/bin/passwd` has type `critical` and `firefox` has type `untrusted`.

The (program) type is used to control the domain from which program may be invoked. For example, program `/usr/bin/passwd` has type `critical` and may be invoked by any subject in domain `COTS`; once invoked, the invoking subject enters domain `system` and when the program returns, the invoker returns to domain `COTS`.

This is configured in the domain definition table.

Domain Definition Table DDT

A *Domain Definition Table (DDT)* defines the allowable access rights within a domain.

		TYPES		
		critical	user	untrusted
DOMAINS	system	RWX	RW	RW
	internet			RW
	COTS	X	RWX	R

- A program executing in domain `internet` may only access `untrusted` objects and may not invoke any other program.
- A program in domain `system` may RW access any type of data, but may only invoke system programs.
- A program in domain `COTS` may access `user` data and also permitted to invoke `critical` programs (enter `system` domain).

Example: DTE policy for Tetris High Scores

An Selinux based implementation of the Tetris game maintains information on player scores in the file `/etc/scores`. The game is executable by all, the high-scores file is readable by all but writable only by the game.

Domains = Trusted, Games, Apps; Types = system, scores, user

	`system`	`scores`	`user`
`Trusted`	`RWX`	`R`	`R`
`Games`	`RX`	`RW`	
`Apps`	`RX`	`R`	`RWX`

File `/etc/passwd` has type system; `/etc/scores` has type scores; `myDocument.doc` has type user

Program `tetris` (type scores) enters domain Games; `/bin/passwd` (type system) enters domain Trusted; `openoffice` (type user) enters domain Apps; `ls` (type system) stays in domain of invoking user.

→ Suggest a DDT for the Ruritanian Translation service example.

Multilevel Security as Type Enforcement Policy

DTE like MLS, but DDT has finer grained control.

For example, consider $\texttt{unclass} \leq \texttt{secret} \leq \texttt{topSecret}$

- DOMAIN={unclass,secret,topSecret}
- TYPE={unclass,secret,topSecret}
- DDT:

	unclass	secret	topSecret
unclass	RWX	W	W
secret	R	RWX	W
topSecret	R	R	RWX

In this scenario we assume that invoking a program causes entry to a domain equal to that of the invoker.

Suppose we have a group of programs that are known not to contain a Trojan Horse. Introduce a further domain, for example, topSecretNoTroj which can RW *all* classes, violating the no-write down rule of MLS.

Simple Interpretation of Chinese Wall in TE

`TYPES` correspond to the different organizations and possible combinations. For example, `TYPES={aib,boi,elf, aibelf, boielf, ...}`.

`DOMAINS` correspond to the legal combinations. For example, `DOMAINS={aib, boi, elf, elfaib, elfboi}`.

Configure DDT so that there's no conflict of interest on the accesses of a process executing in any domain.

Type Enforcement in Practice

- Early research by Secure Computing on high-assurance OS prototypes in 80's/90s'.
- Security Enhanced Linux selinux (an open source project from NSA)
  Replacement kernel for linux that uses TE to provide MAC security.
  A 'rootless' unix: root process is confined to operate within the constraints of a protection domain. EG: root process cannot simultaneously access `/etc/passwd` and `/etc/inetd.conf`.
- Sidewinder: a high-assurance firewall appliance that is implemented using on a TE operating system.
  Firewall processes run in separate domains with only required resources.
  A failure of a process (eg buffer overflow) is confined to the domain and limits how far an attacker can get.
- TE-like mechanisms also found in TrustedBSD (OpenBSD supporting DTE, MLS, etc.), virtual machines/Hypervisors such as Xen.

Mambo Exploit Blocked by SELinux

Jul 01, 2007 By Richard Bulling...

in Security

A real-world case where SELinux proved its worth.

From Issue #159 July 2007

If you operate Internet-connected servers, chances are you eventually will have to deal with a successful attack. Last year, I discovered that despite the multilayered defenses in place on a test Web server (targetbox), an attacker had managed to use an exploit in a partially successful attempt to gain access. This server was running Red Hat Enterprise Linux 4 (RHEL 4) and the Mambo content management system. It had multiple defenses in place, including Security-Enhanced Linux (SELinux). SELinux prevented the attacker from executing the second stage of the attack, possibly preventing a root compromise.

This article presents a case study of the intrusion response, explaining how I discovered the intrusion, what steps I took to identify the exploit, how I recovered from the attack and what lessons I learned regarding system security. I've changed machine names and IP addresses for privacy reasons.