

GTL: a semi-formally developed secure Linux

Contents of the talk

- The problem: Trojan horses
- Military security
- Multi-Level Security (MLS)
- GTL: making a MLS system more usable
- The GTL project

The problem: Trojan horses

Trojan horses

- A *Trojan horse* is a program that attack a system from inside it.
 - The idea is borrowed from Homer's novel
- Some times the Trojan horse shows to the user two functions:
 - One is useful and interesting for the user
 - And the other is hidden and dangerous; in other words, the user doesn't know that the program is attacking his/her system.

An example

- Let's say that you need a new e-mail program.
- So, you search the Internet and find a nice and free program that fulfill your needs.
- Then you download, install and use it in your computer.
- The problem is that every time you send an e-mail to a friend the program forwards it to the person who built the program without your consent.
 - Once you click on the Send button the hidden function of the program adds to the Cc field the e-mail address of the attacker, so you don't see anything suspicious.

Stealing information

- Trojan horses could be used to modify, delete or steal information.
- We'll concentrate in solutions to prevent the fraudulent disclose of information.
- This is known as the *confidentiality problem*.
 - The goal is: to prevent an unauthorized user to read confidential data.
 - Take into consideration that once a user has read some data, then he/she is able to write, send or say it on some other place.

DAC systems

- Traditional operating systems cannot prevent nor mitigate the damage caused by Trojan horses.
- This is because traditional operating systems implement a *Discretionary Access Control* (DAC) model.
 - In a DAC model ordinary users can set security attributes of files and directories.
 - They do so by running OS commands such as `chmod`, `chown`, etc.
 - However, if a person can run those commands, then any program can do the same, or worse any program can invoke the same system calls the commands do.

An example

- Say you own file payroll.xls.
- Then you can run

```
$ chmod +rw payroll.xls
```

Giving read and write access for payroll.xls to every user on your system.

- But chmod invokes an OS call named `sys_chmod`.
- This call checks whether the process invoking it can change permissions on file payroll.xls or not; if it can, then they are changed accordingly.
- A process can change permissions on a file, if the user who run the process can.
- Thus, if you run a Trojan horse, then it'll be able to successfully invoke `sys_chmod` even if you don't want it; even without you noticing it.

Conclusion

- The access control model of traditional operating systems cannot help in preventing Trojan horse actions.
- A new access control model must be implemented.

Military security

What's secret and what it isn't

- There's no way to fight against a Trojan horse if it doesn't exist a clear definition in the system of what is secret and what is not secret.
- Once there is such a definition, appropriate controls can be implemented in order to impose restrictions on information flow.
 - That is, where can be stored or sent, data that was taken from some other source.

Military security

- Military and intelligence corporations were the first in defining secrecy.
- For them, secrecy is fundamental; if a secret is disclosed, lives can be lost.
- The Department of Defense (DoD) of the United States of America imposes a confidentiality policy before the first computer was installed in one of its offices.
- *Multi-level security* (MLS) is a formalization and generalization of that policy written in computer system terms.

DoD information security policy

- In this policy every individual and every document has an *access or security class*.
- A security class is an ordered pair of the form (n,C) where:
 - n : is called *security level* and is one of Unclassified, Confidential, Secret or Top Secret
 - C : is called *category set* and its elements may be CIA, Nuclear, Iraq, F16, Patriot, etc., etc., etc.
 - Access classes are partially ordered (*dominates* relation)
- A person can read a document if and only if his/her access class dominates the access class of the document.
- No one but a reduced and trustworthy bureau of officials can change the security class of documents and persons.

Multi-level security (MLS)

Computer systems aren't the real world

- You may already note that the DoD security policy doesn't mention who and in what conditions can create, write, delete and modify documents.
- When this policy has to be implemented in a computer system, all those actions are important.
- This is so because in a computer system programs, and not people, manage information.
 - Programs could have been built by an enemy; any of them may be a Trojan horse.
 - But people is trustworthy (or there is a little to do to avoid abuse of authority).
- Thus, when the DoD security policy has to be implemented, you have to consider that software or even hardware has been built by your enemy; you're only allowed to trust your people, but not your software.

Trusted Computer Base

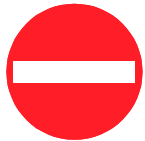
- If you think that all of your hardware and software is potentially harmful, then you won't be able to build a secure computer system.
- It's necessary to assume that there is a portion of your hardware and software that it is trustworthy; in other words, that this H&S haven't been built by your enemy.
- This H&S is known as *Trusted Computer Base* (TCB).
 - It has to be as simple and small as possible
 - If there is a failure in any of its components then your information is at risk

Bell and La Padula model (BLP)

- During the early seventies the DoD funded R+D projects to find a way to implement a computer system secure enough to be immune to Trojan horse attacks against confidentiality.
- In 1972 Elliot Bell and Leonard La Padula from The Mitre Corp. were one of the first in proposing a model for such a system.
 - Their model is the most influential, studied, analyzed and implemented of all advanced security models.
 - It took many years to note that BLP does not necessary means security.

The computer system behind BLP

- Although BLP is an abstract model, the authors assume some features of a computer system:
 - There is an standard TCB
 - Processes run on behalf of users that have been authenticated
 - Processes can read and write information
 - The OS kernel cannot control all the actions of processes
 - This is not necessary true, although it is almost always true.



Problems with BLP and its interpretation

- BLP is a rather good approach to security.
- BLP's standard interpretation suffers of implementation bias.
- One of the most insidious problems in interpreting BLP as it was traditionally done, is that the resulting implementations are unacceptable unusable for modern computing environments.
 - If your organization believes that military security is an acceptable security policy for it, your users will find a system implementing BLP unusable.

Better security models

- In 1982 Goguen and Meseguer published today's dominant security model.
- It is known as noninterference.
- It gives a more abstract notion of security.
- An informal definition is: what a low-level user may see from the system has to be the same no matter whether there are high-level user working on the system or not.
- However this model has some weaknesses.

**GTL: Making a MLS system
more usable**

Moving access control

- BLP-based systems implement security controls at open time.
- However, opening a file doesn't necessary means a security violation.
- Security violations may appear only at read or write time.
- Reads and writes come after opens, then by moving access control from open to read and write we let processes to compute more freely.

Processes are not individuals

- The real problem is to forbid individuals to read certain data, and not to forbid processes to read files.
- Persons may read data:
 - On a screen
 - By printing it on a printer
 - Sending it to another computer
 - Saving it on a removable media
 - Converting it into sound and outputting the sound on a speaker
 - In summary, on output devices that allow a direct or an indirect reading.
 - For example, data stored in a hard disk which is inside a computer cabinet which in turn is locked, cannot be read directly nor indirectly by any person.

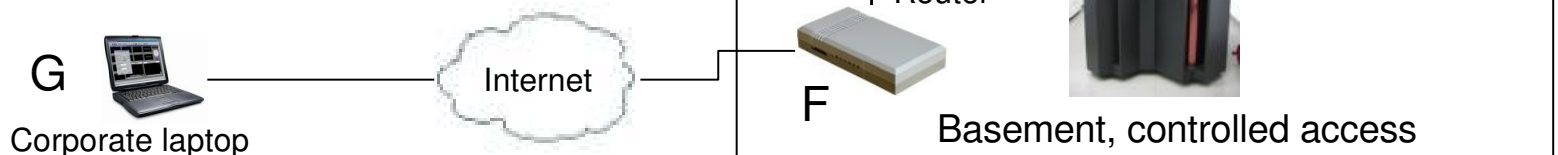
Watch the borders

- Then, GTL' security model will strength controls on the system boundary, and will be more lousy inland.
- We do it as follows:
 - Physical output devices that allow people to read information are classified with an access class.
 - Their logical counterparts are classified accordingly.
 - Those two access classes are static and can change only if the physical device is moved to another location.
 - Each process has a dynamic access class.
 - It starts at $(0, \{ \})$ and it's increased by the OS kernel as long as the process reads information from higher files; it's never decreased.
 - Whenever a process wants to write data on one of those output devices, the OS kernel checks whether the access class of the process dominates the access class of the device or not; in the first case the OS denies the action, while in the second it's authorized.

An example

- All components run a MLS system (router included).
- We set the access class for displays and network cards.
- Possible access classes are Public and Private.

Component	Display	Net. Card
A	Public	Private
B	Private	Private
C	Public	Private
D		Private
E		Private
F		Public
G	Public	Public



The example explained

- Why did we set B's display access class to private?
 - Because nobody but authorized personnel can enter into the protected room, then a person who is able to watch the screen can read what's on that screen.
- Why each router's network card has a different access class?
 - Cards labeled E are Private because private data can be received and sent through any of them because that data remains in trusted computers.
 - Card labeled F is Public because only public information may leave the corporation.

A more detailed example

- Say an employee is writing an e-mail on B.
- He attaches a private file to the e-mail.
 - The e-mail program opens the file and reads it into its memory space.
 - Then, the OS kernel increases the access class of that process to Private.
- The user press Send.
 - The process opens a socket and ask the kernel to send the e-mail text and the attached file through it.
 - The kernel compares the process' and the card' access classes, and authorizes the transmission.
 - Because, sending on a socket is the same as writing on a file.

Limitations of this approach

- If a process reads information from both a private source and a public source, then the OS will let the process to write only into a private source.
- This is so because the OS has no way to know what is the information the process is trying to write.
- Thus, the OS takes a conservative (secure) choice.
- But this goes against usability because that particular operation may be legal.
 - Perhaps the process wants to write public information from one file into another public file.

Controlling process activity: Inline reference monitor

- The first time a program is executed, the OS kernel inserts instructions in critical places of the program image.
- These instructions control the flow of information within the process.
 - They control assignments and conditional structures

The GTL project

Philosophy

- GTL was the source of all the ideas presented in this talk, and, at the same time, the target of them.
 - GTL goals: to semi-formally build an usable MLS system.
- At this time we have a running prototype which represents aprox. 70% of a complete system.
- Also, GTL is a case study on which we experiment with formal development techniques.

GTL Formal Model

- The MLS model implemented in GTL is written in the Z formal notation.
 - Like almost all Z models, GTLSM is an abstract state machine.
 - The set of states is the set of states of a UNIX-like operating system.
 - Transitions are system calls, some user level commands (login, and others), and some programming instructions ($:=$, if-then).

GTL Design

- GTL has to be a Linux version as compatible as possible at the application level.
- Then, the first and most important design restriction was to respect Linux's interface as much as possible.
 - In fact, we didn't modify the signature of any system call; we just added a few more.

GTL Implementation

- The formal model and the design were delivered to programmers.
- They made the final adjustments and implemented them.
- There were no formal refinement.

Testing GTL

- We have used functional testing based on formal specifications to test the implementation.
 - The formal model is used as the source and oracle of test cases.
- We have done it manually twice.
 - We won't do it again!
- We believe is a very good technique but we need tool support.

Documenting GTL

- We have documented:
 - Specification (Z/EVES)
 - Design (LaTeX)
 - Implementation (Doxygen)
 - Testing (HTML)
- We need a way to keep all this information up to date and traceable.
 - We want to go from a requirement all the way down to a test case and back again.

TODO: areas for possible interaction between LASSY and GIDISS

- Inline reference monitor
- Testing
- Documenting
- GTL Distribution
 - To build and maintain a Linux distro.