## Access control models

One of the ways to defend against attacks is to prevent them by providing rigorous means of determining who has access to various pieces of information. All of the models assume that there are data managers, data owners, or system administrators who are defining the access control specifications. The access shall be restricted to those who have a need to access and/or modify the information in question. That is, they should apply the principle of least privilege.

Access Control Matrices (ACM)

It is a table that defines permissions. Each row of this table is associated with a subject, which is a user, group, or system that can perform actions. Each column of the table is associated with an object which is a file, directory, document, device, resource, or any other entity for which we want to define access rights. Each cell of the table is filled with the access rights for the associated combination of subject and object. Access rights can include actions such as reading, writing, copying, executing, deleting. An empty cell means ~~c,~~that no access rights are granted.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Subject\Objects | /etc/passwd | /usr/bin/ | /u/Roberto/ | /admin/ |
| Root | Read, write | Read, write, exec | Read, write, exec | Read, write, exec |
| Mike | Read | Read, exec |  |  |
| Roberto | Read | Read, exec | Read, write, exec |  |
| Backup | read | Read, exec | Read, exec | Read, exec |

Access Control Lists (ACL)

These are columns of ACM arranged as lists, showing for each object who (subject) has what access the object.

End of the list is indicated by NULL pointer; 4 bytes

ACL=>1stobject=>2nd object=>…=>nth object NULL

 |

 V

 Privileges of 1st user

 |

 V

Priviliges of the 2nd user

 NULL

1000 users, 1000 objects; each user accesses 10 objects; 1 000 000 will be used only 100 000

Each node 4 bytes+4=8 bytes Memory for ACL = 100 000\*8=800 000

Matrix = 1000 000\*4 = 4 000 000

Capability Lists (CL)

These are rows of ACM showing for each subject what objects and in what mode are allowed for him/her

CL=>1stuser=>2nd user=>…=>nth user NULL

 |

 V

 Object 1 Privileges

 |

 V

Object 2 privileges

 NULL

Role-Based Access Control (RBAC)

Administrators define roles, specify access control rights for tem, and then assign roles to subjects/roles.

Student role

Teacher role

Academic advisor role

A new user is enrolled he/she is assigned to a particular group

A gets Student role

B=>Teacher

Table (User, Role)

A Student

A Teacher

A Academic Advisor

A Chair

Priviliges are defined by Union of Roles’ privileges

A={a,b,c} B={c,d} Union(A,B)={a,b,c, d}

Mandatory Access Control (MAC), Bell – La Padula model (BLP)

BLP model is derived from the military multilevel security paradigm (Top secret 4, Secret 3, Confidential 2, Unclassified 1)

Each document has 1 out of 4 security levels, and each user has “clearance”, also 1 out of 4.

A document of a certain level can be accessed only by users with the same or higher clearance level, “no read-up” rule.

Read:

A level=2>= Doc level=1; yes? Grant access

A level=2>= Doc level=3; no? Deny access

A level=2>= Doc level=2; yes? Grant access

Write:

A level=2<= Doc level=1; no? deny access

A level=2<= Doc level=3; yes? grant access

A level=2<= Doc level=2; yes? Grant access

To counter Trojan horse attack

Hacker wants accessing Secret 3 documents

Bob is level 3 can read Level 3 documents

Hacker is level 1 unclassified

Hacker knows Bob like gaming prepares a game interesting to Bob, and Bob finds game, invokes it, and game runs on Level 3 , can read secret documents

Game fully functional + additional functionality of reading Secrets and saving (writing ) them in “back pocket” file of level 1

System requirements but also no other additional functionalities

Acceptance testing

A user can write only in the documents of his or higher level of security, “no write down” rule, ‘\*’ property.

## Passwords

Dictionary attack

For the English language, there are less than 50000 common words, 1000 common human first names, 1000 typical pet names, and 10000 common last names. There are only 36525 birthdays and anniversaries for almost all living humans on the planet. So, an attacker can compile a dictionary of all these common passwords of fewer than 100000 entries.

Secure passwords

Number of possible 8-character passwords on a typical American keyboard is 94^8=6\*10^15, trying of such number of passwords is infeasible

Fugaku Top500 10^17 FLOP/s K 10^3, M 10^6 G 10^9 T 10^12\*4\*10^5=4\*10\*17 Flop/s

Complex passwords is difficult to remember

It is recommended to have at least 8 symbols, including letters, digits, special signs, and different case symbols (upper and lower)

Social Engineering

The three B’s of espionage – burglary, bribery, and blackmail – apply equally well to computer security. Together with trickery we get one of the most powerful attacks against computer security – social engineering.

Pretexting

Eve calls a helpdesk and tells that she has forgotten her password, when she is actually calling about the account of someone else, say, Alice. The helpdesk agent may ask Eve a few personal questions about Alice, which, if Eve has done her homework, she can answer with ease.. Then the agent resets the password for Alice’s account and gives Eve the new password. This attack does not any hardware or software but very effective.

Baiting

It involves using some kind of “gift” as a bait to get someone to install malicious software. For example, an attacker could leave a few USB drives in the parking lot of a company with an otherwise secure computer system, even marking some with the names of popular software programs or games. The hope is that some unsuspecting employee will pick up a USB drive on his lunch break, bring it into the company, insert it into an otherwise secure computer, and unwittingly install the malicious software.

Quid Pro Quo

Latin “something for something”. For example, an attacker, Bob, might call victim, Alice, on the phone saying that he is a helpdesk agent who was referred to Alice by a coworker. Bob then asks Alice if she has been having any trouble with her computer or with her company’s computer in general. Or he could ask Alice if she needs any help in coming up with a strong password now that it is time to change her old one. In any case, Bob offers Alice some legitimate help. He may even diagnose and solve a problem she has been having with her computer. This is the “something” that Bob has now offered Alice, seemingly without asking for anything in return. At that point, Bob then asks Alice for her password, possibly offering to perform future fixes or offering to do an evaluation of how strong her password is. Because of the social pressure that is within each of us to want to return favor, Alice may feel completely at ease at this point in sharing her password with Bob in return for his “free” help. If she does so, she will have just become a victim of the quod pro quo attack.

Vulnerabilities from programming errors

The classic buffer overflow attack injects code into a running application by exploiting the common programming error of not checking whether an input string read by the application is larger than the variable into which it is stored (the buffer). Thus, a large input provided by the attacker can overwrite the data and code in the application performing malicious actions specified by the attacker.

char name[10]; x[i], i=0,..,9 \*(x+i) pointer to the 1st element of the array

Run-time range checking off x=5; i=x^2+1; i=26; x[i]; x[26]=1;

## Malicious software

* 1. Insider attacks
		1. Backdoors

Backdoors inserted for debugging purposes: some special key combinations allowing a developer getting access to the system variables. Backdoors may be intentionally left by developers for the next malicious accesses. Easter eggs are some not declared functionalities which might be deliberately made by developers.

* + 1. Logic bombs

Is a program that performs a malicious action as a result of a certain logical condition (e.g., developer is not paid money for his work).

* + 1. Defenses from insider attacks
* Avoid single point of failure. Let no one person be the only one to create backups or manage critical systems
* Use code-walk-through. Source code shall be available and analyzed versus specification
* Use archiving and reporting tools. When a program code is archived, it can be analyzed by managers (after automatic digesting), and it becomes harder for a team member to avoid the existence of malware source code to go undiscovered after an attack
* Limit authority and permissions. Use a least privilege principle
* Physically secure critical systems
* Monitor employee behavior. Be especially on the lookout for system administrators and programmers that have become disgruntled
* Control software installations. Limit new software installations to programs that have been wetted and come from reliable sources
	1. Computer viruses
		1. Virus definition

It is a code that can replicate itself by modifying other files or programs to insert code that is capable of further replication. Viruses phases of execution:

* Dormant phase: it is existing but lays low avoiding detection
* Propagation phase: replication is made
* Triggering phase: some logical condition causes virus performing payload, or transition from phase to phase
* Action phase: payload execution
	+ 1. Virus types
* Program virus (file virus)
* Macrovirus
* Boot sector virus
	+ 1. Defenses against viruses

Virus signatures

Virus detection and quarantine

* + 1. Encrypted viruses

They may be discovered by checking for decryption code

* + 1. Polymorphic/metamorphic viruses

They modify themselves using different encryption keys (polymorphic) or reordering code instructions (metamorphic)

A+B+C=C+B+A=B+C+A 3!=3\*2\*1=6 26!=10^26

A\*B=B\*A; A/B<>B/A

* 1. Malware attacks
		1. Trojan horse attack

Looking innocent programs attractive for a user but having undeclared functionality

* + 1. Computer warms

Is a program that spreads copies of itself without the need to inject itself into other programs, and usually without human interaction. They typically spread by exploiting vulnerabilities (e.g., buffer overflow) in applications run by Internet-connected computer systems that have a security hole. Once a system is infected, a warm must make steps to ensure that it persists on the victim machine and survives rebooting. On Windows machines, this is commonly achieved by modifying the Windows Registry, a database used by the operating system that includes entries that tell the operating system to run certain programs and services or load device drivers on system startup. One of the most common registry entries for this purpose is called

HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run

Associating with this entry the path to the executable file of the warm will result in Windows executing the worm on startup. Thus, malware detection software always checks this entry (and other registry entries specifying programs to run at startup) for suspicious executable names Fig. 1.



Fig. 1. Screenshot of regedit output of HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run

Detecting of worms is similar to detecting viruses. In addition, network-level scanning and filtering that analyze the content of network packets before they are delivered to a machine, allows to detect and lock worms in real time.

* + 1. Rootkits

It Is an especially stealthy type of malware. Rootkits typically alter system utilities or the operating system itself to prevent detection. For example, a rootkit that infect the Windows Process Monitor utility, which lists currently running processes, could hide by removing themselves from the process list. Likewise, a rootkit might hide files on disk by infecting utilities that allow the user to browse files, such as Windows Explorer. Rootkits are often used to hide the malicious actions of types of malware, such as Trojan horses and viruses.

Concealment

Rootkits employ several techniques to achieve stealth. Software can either run in user-mode, which includes ordinary program execution, or kernel-mode, which is used for low-level, privileged operating system routines. Accordingly, rootkits may operate in either of these two modes.

Some user-mode rootkits work by altering system utilities or libraries on disk. While this approach may be the simplest, it is easily detected. Because checking the integrity of files can be performed offline by using a cryptographic hash function, as detailed below. Other user-mode rootkits insert code into another user-mode process’s address space in order to alter its behavior, using technologies such as DLL injection, <http://resources.infosecinstitute.com/using-createremotethread-for-dll-injection-on-windows/> . Anti-rootkit software running at the kernel level may detect such rootkits.

Kernel-mode rootkits are considered more difficult to detect, because they work at the lowest levels of the operating system. Kernel rootkits in Windows are typically loaded as device drivers, because the device driver system is modular – it allows users to load arbitrary code into the kernel.

Once rootkit code is loaded into the kernel, several techniques may be employed to achieve stealth. One of the most common methods is known as function hooking. Because the rootkit is running with kernel privileges, it can directly modify kernel memory to replace operating system functions with customized versions that steal information or hide the existence of the rootkit. For example, a rootkit might replace a kernel function that enumerates files in a directory with a nearly identical version that is designed to skip over particular files that are part of the rootkit. This way, every program that uses this function will be unable to detect the rootkit. Kernel function hooking is powerful in that rootkit developers only have to alter one function, as opposed to patching every system utility that lists directory contents.

Another kernel-mode rootkit technique involves modifying the internal data structures the kernel uses for bookkeeping purposes. For example, the Windows kernel keeps a list of information on the device drivers that are currently loaded into memory. A rootkit might modify this data structure directly to remove itself from the list and potentially avoid detection. A rootkit that performs this action may be difficult to remove.

Once a system is infected, a rootkit must take steps to ensure that it persists on the victim machine and survives rebooting, including the modification of appropriate entries in the Windows Registry. Since anti-rootkit software searches the registry for suspicious entries, to avoid detection, some rootkits modify the kernel functions that list registry entries. This is an example of the arms race that takes place between rootkit and ant-rootkit software which are constantly engaged in a complex game of hide-and-seek.

Detecting rootkits

User-mode rootkits can be detected by checking for modifications to files on disk. On Windows, important code libraries are digitally signed, so that any tempering would invalidate the digital signature and be detected. Another commonly employed technique is to periodically compute a cryptographic hash function for critical system components while the system is offline. This hash can be recomputed while the system is online, and if the hashes do not match, then a rootkit may be altering these files. In addition, kernel-mode anti-rootkit software can detect code injection in system processes.

Kernel-mode rootkits can be more difficult to detect. Most anti-rootkit applications detect kernel rootkits by searching for evidence of techniques such as function hooking. Such rootkit detectors may keep signatures of certain kernel functions that are likely to be targeted by rootkits, and inspect kernel memory to determine if any modifications have been made to this functions. However, because kernel rootkits operate at the highest level of system privileges, they may preemptively detect anti-rootkit software and prevent it from achieving its goals. Therefore, sometimes an in-depth offline analysis of an infected system, including inspection of the registry and boot records, is required to defeat rootkits.

Given the difficulty of guaranteeing the removal of rootkits, users are often advised to reformat their hard drive on suspicion of infection, rather than risking continued compromise by failing to remove all traces of rootkit activity.

Zero-day attacks

They use vulnerabilities that have not been known to the software developers. To detect such attacks, potentially dangerous actions of programs (deletion and modification of files, Internet activity) cause alerts to users, so that they decide whether these are legitimate or not. It is as a program runs in a virtual machine, or sand-box.

* + 1. Botnets

Networks of compromised computers are known as botnets, their owners as bot herders. Botnets can contain several million compromised machines.

One of the key properties of a botnet is a central command-and-control mechanism. Once bot software is installed on a compromised computer via a worm, Trojan horse, or some other malware package, the infected machine, known as zombie, contacts a central control server to request commands. To avoid detection, command servers use dynamic IPs and unexpected channels to send commands (e.g., Twitter, Instant Messaging).

Botnets may be used for credit card numbers stealing, sending of spam, DDoS attacks, etc.

* 1. Privacy invasive software
		1. Adware

Typically, an adware program is installed on a user’s computer because he visits an infected web-page, opens an infected e-mail attachment, installs a shareware or freeware program that has the adware embedded in it.

* + 1. Spyware

Collects passwords, monitors keystrokes, captures screen, tracks cookies