International Journal Of Engineering Research And Development
e- ISSN: 2278-067X, p-ISSN: 2278-800X, www.ijerd.com
Volume 14, Issue 5 (May Ver. II 2018), PP.08-16

Concept of Computer Security

Dr. Amamer Khalil Masoud Ahmidat
Higher Institute of Medical Technology in Baniwaleed, Libya
Faculty of Sciences Baniwaleed University -Libya
Corresponding Author: Dr. Amamer Khalil Masoud Ahmidat

ABSTRACT: After many years of work on computer security, computers are not secure till now. The main reason is that security is expensive to set up and a nuisance to run, so people judge from experience how little of it they can get away with. Since there's been little damage, people decide that they don't need much security. In addition, setting it up is so complicated that it's hardly ever done right. While we await a catastrophe, simpler setup is the most important step toward better security. In a distributed system with no central management like the Internet, security requires a clear story about who is trusted for each step in establishing it, and why. The basic tool for telling this story is the "speaks for" relation between principals that describes how authority is delegated, that is, who trusts whom. The idea is simple, and it explains what's going on in any system I know. The many different ways of encoding this relation often make it hard to see the underlying order.

Date of Submission: 05-05-2018 Date of acceptance: 21-05-2018

I INTRODUCTION

Engineers have been working on computer system security. During this time there have been many intellectual successes. Notable among them are the subject/object access matrix model, access control lists, multilevel security using information flow and the star-property, public key cryptography, and cryptographic protocols. In spite of these successes, it seems fair to say that in an absolute sense, the security of the hundreds of millions of deployed computer systems is terrible: a determined and competent attacker could destroy most of the information on almost any of these systems, or steal it from any system that is connected to a network. Even worse, the attacker could do this to millions of systems at once. The Internet has made computer security much more difficult than it used to be. In the good old days, a computer system had a few dozen users at most, all members of the same organization. It ran programs written in-house or by a few vendors. Information was moved from one computer to another by carrying tapes or disks. Today half a billion people all over the world are on the Internet, including you. This poses a large new set of problems.

- Attack from anywhere: Any one on the Internet can take a poke at your system.
- Sharing with anyone: On the other hand, you may want to communicate or share information with any other Internet user.
- Automated infection: Your system, if compromised, can spread the harm to many others in a few seconds.
- Hostile code: Code from many different sources runs on your system, usually without your knowledge if it comes from a Web page. The code might be hostile, but you can't just isolate it, because you want it to work for you.
- Hostile physical environment: A mobile device like a laptop may be lost or stolen and subject to physical attack.
- Hostile hosts: If you own information (music or movies, for example), it gets downloaded to your customers' systems, which may try to steal it.
-

All these problems cause two kinds of bad results. One is vandalism, motivated by personal entertainment or status seeking: people write worms and viruses that infect many machines, either by exploiting buffer overrun bugs that allow arbitrary code to run, or by tricking users into running hostile code from e-mail attachments or web pages. These can disrupt servers that businesses depend on, or if they infect many end user machines they can generate enough network traffic to overload either individual web servers or large parts of the Internet itself. The other bad result is that it's much easier to mount an attack on a specific target (usually an organization), either to steal information or to corrupt data. On the other hand, the actual harm done by these attacks is limited, though growing. Once or twice a year an email virus such as "I love you" infects a million or two machines, and newspapers print extravagant estimates of the damage it does. Unfortunately, there is no accurate data about the cost of failures in computer security: most of them are never made public for fear of embarrassment, but when a public incident does occur, the security experts and vendors of antivirus software that talk to the media have every incentive to greatly exaggerate its costs. Money talks, though. Many vendors of security products have learned to their regret that people may complain about inadequate security, but they won't spend much money, sacrifice many features, or put up with much inconvenience in order to improve it. This strongly suggests that bad security is not really costing them much. Firewalls and anti-virus programs are the only really successful security products, and they are carefully designed to require no end-user setup and to interfere very little with daily life. The experience of the last few years confirms this analysis. Virus attacks have increased, and people are now more likely to buy a firewall and antivirus software, and to install patches that fix security flaws. Vendors like Microsoft are making their systems more secure, at some cost in backward compatibility and user convenience. But the changes have not been dramatic. Many people have suggested that the PC monoculture makes security problems worse and that more diversity would improve security, but this is too simple. It's true that vandals can get more impressive results when most systems have the same flaws. On the other hand, if an organization installs several different systems that all have access to the same critical data, as they probably will, then a targeted attack only needs to find a flaw in one of them in order to succeed.

Of course, computer security is not just about computer systems. Like any security, it is only as strong as its weakest link, and the links include the people and the physical security of the system. Very often the easiest way to break into a system is to bribe an insider. This short paper, however, is limited to computer systems. It does not consider physical or human security. It also does not consider how to prevent buffer overruns. You might think from the literature that buffer overruns are the main problem in computer security, and of course it's important to eliminate them, especially in privileged code, but I hope to convince you that they are only a small part of the problem.

II METHODS AND MATERIALS

A. SECURITY

What do we want from secure computer systems? Here is a reasonable goal: *Computers are as secure as real world systems, and people believe it.* Most real world systems are not very secure by the absolute standard suggested above. It's easy to break into someone's house. In fact, in many places people don't even bother to lock their houses, although in Manhattan they may use two or three locks on the front door. It's fairly easy to steal something from a store. You need very little technology to forge a credit card, and it's quite safe to use a forged card at least a few times. Why do people live with such poor security in real world systems? The reason is that real world security is not about perfect defenses against determined attackers. Instead, it's about

- value,
- locks, and
- punishment.

The bad people balances the value of what they gain against the risk of punishment, which is the cost of punishment times the probability of getting punished. The main thing that makes real world systems sufficiently secure is that bad guys who do break in are caught and punished often enough to make a life of crime unattractive. The purpose of locks is not to provide absolute security, but to prevent casual intrusion by raising the threshold for a break-in.

Well, what's wrong with perfect defenses? The answer is simple: they cost too much. There is a good way to protect personal belongings against determined attackers: put them in a safe deposit box. After 100 years of experience, banks have learned how to use steel and concrete, time locks, alarms, and multiple keys to make these boxes quite secure. But they are both expensive and inconvenient. As a result, people use them only for things that are seldom needed and either expensive or hard to replace. Practical security balances the cost of protection and the risk of loss, which is the cost of recovering from a loss times its probability. Usually the probability is fairly small (because the risk of punishment is high enough), and therefore the risk of loss is also small. When the risk is less than the cost of recovering, it's better to accept it as a cost of doing business (or a cost of daily living) than to pay for better security. People and credit card companies make these decisions every day. With computers, on the other hand, security is only a matter of software, which is cheap to manufacture, never wears out, and can't be attacked with drills or explosives. This makes it easy to drift into thinking that computer security can be perfect, or nearly so. The fact that work on computer security has been dominated by the needs of national security has made this problem worse. In this context the stakes are much higher and there are no police or courts available to punish attackers, so it's more important not to make mistakes. Furthermore, computer security has been regarded as an offshoot of communication security, which is based on cryptography. Since cryptography can be nearly perfect, it's natural to think that computer security can be as well. What's wrong with this reasoning? It ignores two critical facts:

- □ Secure systems are complicated, hence imperfect.
- □ Security gets in the way of other things you want.

Software is complicated, and it's essentially impossible to make it perfect. Even worse, security has to be set up, by establishing user accounts and passwords, access control lists on resources, and trust relationships between organizations. In a world of legacy hardware and software, networked computers, mobile code, and constantly changing relationships between organizations, setup is complicated. And it's easy to think up scenarios in which you want precise control over who can do what. Features put in to address such scenarios make setup even more complicated. Security gets in the way of other things you want. For software developers, security interferes with features and with time to market. This leads to such things as a widely used protocol for secure TCP/IP connections that use the same key for every session as long as the user's password stays the same, or an endless stream of buffer-overrun errors in programs that are normally run with administrative privileges, each one making it possible for an attacker to take control of the system. For users and administrators, security interferes with getting work done conveniently or in some cases at all. This is more important, since there are lot more users than developers. Security setup also takes time, and it contributes nothing to useful output. Furthermore, if the setup is too permissive no one will notice unless there's an audit or an attack. This leads to such things as users whose password is their first name, or a large company in which more than half of the installed database servers have a blank administrator password, or public access to databases of credit card numbers, or email clients that run attachments containing arbitrary code with the user's privileges.

B. REAL SECURITY

The end result should not be surprising. We don't have "real" security that guarantees to stop bad things from happening, and the main reason is that people don't buy it. They don't buy it because the danger is small, and because security is a pain.

- Since the danger is small, people prefer to buy features.
- A secure system has fewer features because it has to be implemented correctly. This means that it takes more time to build, so naturally it lacks the latest features.
- Security is a pain because it stops you from doing things, and you have to do work to authenticate yourself and to set it up.
- A secondary reason we don't have "real" security is that systems are complicated, and therefore both the code and the setup have bugs that an attacker can exploit. This is the reason that gets all the attention, but it is not the heart of the problem.
- Will things get better? Certainly when security flaws cause serious damage, buyers change their priorities and systems become more secure, but unless there's a catastrophe, these changes are slow. Short of that, the best we can do is to drastically simplify the parts of systems that have to do with security:
- Users need to have at most three categories for authorization: me, my group or company, and the world.

- Administrators need to write policies that control security settings in a uniform way, since they can't deal effectively with lots of individual cases.
- Everyone needs a uniform way to do end-to-end authentication and authorization across the entire Internet.

Since people would rather have features than security, most of these things are unlikely to happen very quickly. On the other hand, don't forget that in the real world security depends more on police than on locks, so detecting attacks, recovering from them, and punishing the bad people are more important than prevention.

C. OVERVIEW OF COMPUTER SECURITY

Like any computer system, a secure system can be studied under three headings:
(Common name Meaning Security jargon Specification)
Policy Implementation:

In security it's customary to give new names to familiar concepts; they appear in the last column.
Assurance, or correctness, is especially important for security because the system must withstand malicious attacks, not just ordinary use. Deployed systems with many happy users often have thousands of bugs. This happens because the system enters very few of its possible states during ordinary use. Attackers, of course, try to drive the system into states that they can exploit, and since there are so many bugs, this is usually quite easy. This section briefly describes the standard ways of thinking about policy and mechanism. It then discusses assurance in more detail, since this is where security failures occur.

Policy: Specifying security

Organizations and people that use computers can describe their needs for information security under four major headings:

- *Secrecy*: controlling who gets to read information.
- *Integrity*: controlling how information changes or resources are used.
- *Availability*: providing prompt access to information and resources.
- *Accountability*: knowing who has had access to information or resources.

They are usually trying to protect some resource against danger from an attacker. The resource is usually either information or money. The most important dangers are:
Vandalism or sabotage that

- damages information
- disrupts service integrity availability
- Theft
- of money integrity
- of information secrecy
- Loss of privacy secrecy

Each user of computers must decide what security means to them. A description of the user's needs for security is called a security policy. Most policies include elements from all four categories, but the emphasis varies widely. Policies for computer systems are usually derived from policies for real world security. The military is most concerned with secrecy, ordinary businesses with integrity and accountability, telephone companies with availability.

Obviously integrity is also important for national security: an intruder should not be able to change the sailing orders for a carrier, and certainly not to cause the firing of a missile or the arming of a nuclear weapon. And secrecy is important in commercial applications: financial and personnel information must not be disclosed to outsiders. Nonetheless, the difference in emphasis remains.

A security policy has both a positive and negative aspect. It might say, "Company confidential information should be accessible only to properly authorized employees". This means two things: properly authorized employees *should* have access to the information, and other people *should not* have access.

When people talk about security, the emphasis is usually on the negative aspect: keeping out the bad guy. In practice, however, the positive aspect gets more attention, since too little access keeps people from getting their work done, which draws attention immediately, but too much access goes undetected until there's a security audit or an obvious attack, which hardly ever happens. This distinction between talk and practice is pervasive in security.

This paper deals mostly with integrity, treating secrecy as a dual problem. It has little to say about availability, which is a matter of keeping systems from crashing and allocating resources both fairly and cheaply. Most attacks on availability work by overloading systems that do too much work in deciding whether to accept a request.

D. IMPLEMENTING SECURITY

The informal access policy in the previous paragraph must be elaborated considerably before it can be enforced by a computer system. Both the set of confidential information and the set of properly authorized employees must be described precisely. We can view these descriptions as more detailed policy, or as implementation of the informal policy. In fact, the implementation of security has two parts: the code and the setup or configuration. The code is the programs that security depends on. The setup is all the data that controls the operations of these programs: folder structure, access control lists, group memberships, user passwords or encryption keys, etc.

The job of a security implementation is to defend against vulnerabilities. These take three main forms:

- Bad (buggy or hostile) *programs.*
- Bad (careless or hostile) agents, either programs or *people*, giving bad instructions to good but gullible programs.
- Bad agents tapping or spoofing *communications*.

Broadly speaking, there are five defensive strategies:

1) *Coarse: Isolate*—keep everybody out. It provides the best security, but it keeps you from using information or services from others, and from providing them to others. This is impractical for all but a few applications.

2) *Medium: Exclude*—keep the bad guys out. It's all right for programs inside this defense to be gullible. Code signing and firewalls do this.

3) *Fine: Restrict*—Let the bad guys in, but keep them from doing damage. Sandboxing does this, whether the traditional kind provided by an operating system process, or the modern kind in a Java virtual machine. Sandboxing typically involves access control on resources to define the holes in the sandbox. Programs accessible from the sandbox must be paranoid; it's hard to get this right.

4) *Recover*—Undo the damage. Backup systems and restore points are examples. This doesn't help with secrecy, but it helps a lot with integrity and availability.

5) *Punish*—Catch the bad guys and prosecute them. Auditing and police do this.

Users need a very simple story, with about three levels of security: me, my group or company, and the world, with progressively less authority. Browsers classify the network in this way today. The corresponding private, shared, and public data should be in three parts of the file system: my documents, shared documents, and public documents. This combines the security of data with where it is stored, just as the physical world does with its public bulletin boards, private houses, locked file cabinets, and safe deposit boxes. It's familiar, there's less to set up, and it's obvious what the security of each item is. Everything else should be handled by security policies that vendors or administrators provide. In particular, policies should classify all programs as trusted or untrusted based on how they are signed, unless the user overrides them explicitly. Untrusted programs can be rejected or sandboxed; if they are sandboxed, they need to run in a completely separate world, with separate global state such as user and temporary folders, history, web caches, etc. There should be no communication with the trusted world except when the user explicitly copies something by hand. This is a bit inconvenient, but anything else is bound to be unsafe.

E. ADMINISTRATORS still need a fairly simple story, but they need even more the ability to handle many users and systems in a uniform way, since they can't deal effectively with lots of individual cases. The way to do this is to let them define so called security *policies*7, rules for security settings that are applied automatically to groups of machines. These should say things like:

- Each user has read/write access to their home folder on a server, and no one else has this access.
- A user is normally a member of one workgroup, which has access to group home folders on all its members' machines and on the server.
- System folders must contain sets of files that form a vendor-approved release.
- All executable programs must be signed by a trusted authority.

These policies should usually be small variations on templates provided and tested by vendors, since it's too hard for most administrators to invent them from scratch. It should be easy to turn off backward compatibility with old applications and network nodes, since administrators can't deal with the security issues it causes.

Some customers will insist on special cases. This means that useful exception reporting is essential. It should be easy to report all the variations from standard practice in a system, especially variations in the software on a machine, and all changes from a previous set of exceptions. The reports should be concise, since long ones are sure to be ignored.

To make the policies manageable, administrators need to define *groups* of users and of resources, and then state the policies concisely in terms of these groups. Ideally, groups of resources follow the file system structure, but there need to be other ways to define them to take account of the baroque conventions in existing networks, OS's and applications.

To handle repeating patterns of groups, system architects can define *roles*, which are to groups as classes are to objects in Java. So each division in a company might have roles for 'employees', 'manager', 'finance, and 'marketing', and folders such as 'budget' and 'advertising plans'. The 'manager' and 'finance' roles have write access to 'budget' and so forth.

The Appliance division will have a specific group for each of these: 'Appliance-members', 'Appliance-budget', etc., and thus 'Appliance-finance' will have write access to 'Appliance budget'.

Policies are usually implemented by compiling them into existing security settings. This means that existing resource managers don't have to change, and it also allows for both powerful high-level policies and efficient enforcement, just as compilers allow both powerful programming languages and efficient execution.

F . DEVELOPERS need a type-safe virtual machine like Java or Microsoft's Common Language Runtime; this will eliminate a lot of bugs. Unfortunately, most of the bugs that hurt security are in system software that talks to the network, and it will be a while before system code is written that way. They also need a development process that takes security seriously, valuing designs that make assurance easier, getting them reviewed by security professionals, and refusing to ship code with serious security flaws.

III END-TO-END ACCESS CONTROL

Secure distributed systems need a way to handle authentication and authorization uniformly throughout the Internet. In this section we first explain how security is done locally today, and then describe the principles that underlie a uniform end-to-end scheme.

A . LOCAL ACCESS CONTROL

Most existing systems, such as Unix and Windows, do authentication and authorization locally. They have a local database for user authentication (usually by passwords) and group membership, and a local database of authorization information, usually in the form of an access control list (ACL) on each resource.

They either rely on physical security or luck to secure the channel to the user, or use an encrypted channel protocol like PPTP. The system identifies a logged-in user by a "security identifier" or SID.

You might think that security on the web is more global or distributed, but in fact web servers work the same way. They usually use SSL to secure the user channel; this also authenticates the server's DNS name, but users hardly ever pay any attention. Authorization is primitive, since usually the only protected resources are the entire service and the private data that it keeps for each user. Each server farm has a separate local user database.

There is a slight extension of this strictly local scheme, in which each system belongs to a "domain" and the authentication database is stored centrally on a domain controller. To log in a user the local system, instead of doing the work itself, does an RPC to the controller (secured by a shared encryption key that is set up when the system joins the domain), passing in the user's password response. The controller does exactly what the login system did by itself in the earlier scheme, and returns the user's identity. It also returns a token that the login system can use later to re authenticate the user to the controller.

B. DISTRIBUTED ACCESS CONTROL

A distributed system may involve systems (and people) that belong to different organizations and are managed differently. To do access control cleanly in such a system (as opposed to the strictly local systems of the last section) we need a way to treat uniformly all the items of information that contribute to the decision to grant or deny access.

C. CHAINS OF TRUST

What is the common element in all the steps of the example and all the different kinds of information? From the example we can see that there is a *chain of trust* running from the request at one end to the Spectra resource at the other. A link of this chain has the form "Principal *P* speaks for principal *Q* about statements in set
T."

D. AUTHENTICATING SYSTEMS

As we saw in section 3.4, we can treat a program image, represented by its secure hash, as a principal; the hash plays the same role as an encryption key. But a program can't make statements; to do so, it must be *loaded* into a *host H*. Booting an operating system is a special case of loading. A loaded program depends on the host it runs on; if you don't trust the host, you certainly shouldn't trust the running program.

IV VARIATIONS

There are many variations in the details of setting up a chain of trust:

- How secure channels are implemented.
- How bytes are stored and moved around.
- Who collects the evidence?
- Whether evidence is summarized.
- How big objects are and how expressive *T* is.
- What compound principals exist other than names.

We pass over the complicated details of how to use encryption to implement secure channels. They don't affect the overall system design much, and problems in this area are usually caused by over-optimization or sloppiness.

A. COLLECTING EVIDENCE: The verifier (the guard that's granting access to an object) needs to see the evidence from each link in the chain of trust. There are two basic approaches to collecting this information:
Push: The client gathers the evidence and hands it to the object.
Pull: The object queries the client and other databases to collect the evidence it needs.
Most systems use push for authentication, the evidence for the identity of the client, and pull for authorization, the evidence for who can do what to the object. The security tokens in
Windows are an example of push, and ACLs an example of pull.
If the client is feeble, or if some authentication information such as group memberships is stored near the object, more pull may be good. Cross-domain authentication in Windows is a partial example: the target domain controller, rather than the login controller, discovers membership in groups that are local to the target domain.

B. SUMMARIZING EVIDENCE: It's possible to replace several links of a chain like $P \Rightarrow Q \Rightarrow R$ with a single link $P \Rightarrow R$ signed by someone who speaks for R. In the limit a link signed by the object summarizes the whole chain; this is usually called a capability. An open file descriptor is a familiar example; it summarizes the access rights of a process to a file, which are checked when the process opens the file. The advantages are savings in space and time to verify, which are especially important for feeble objects such as computers embedded in small devices. The drawbacks are that it's harder to do setup and to revoke access.

C. AUDITING

As is typical for computer security, we have focused on how end-to-end access control works and the wonderful things you can do with it. An equally important property, though, is that the chain of trust collects in one place, and in an explicit form, all the evidence and rules that go into making an access control decision. You can think of this as a *proof* for the decision. If the guard records the proof in a reasonably tamperresistant log, an auditor can review it later to establish accountability or to figure out whether some unintended access was granted, and why. Since detection and punishment is the primary instrument of practical security, this is extremely important.

V CONCLUSION

We have outlined the basic ideas of computer security: secrecy, integrity, and availability, implemented by access control based on the gold standard of authentication, authorization, and auditing. We discussed the reasons why it doesn't work very well in practice:

- Reliance on prevention rather than detection and punishment.
- Complexity in the code and especially in the setup of security, which overwhelms users and administrators.

We gave some ways to reduce this complexity.
Then we explained how to do access control end-to-end in a system with no central management, by building a chain of trust based on the "speaks for" relation between principals.
Each link in the chain is a delegation of the form "Alice@
Intel speaks for Atom.Microsoft about reads and writes of files named *.doc". The right principals (Intel,
Microsoft, and the file owner in this example) has to assert each link, using a secure channel. Every kind of authentication and authorization information fits into this framework: encrypted channels, user passwords, groups, setuid programs, and ACL entries. The chain of trust is a sound basis for logging and auditing access control decisions. Principals with hierarchical names are especially important.
A parent can delegate for all of its children. Rooting name spaces in keys avoids any need for a globally trusted root.
There are many ways to vary the basic scheme: how to store and transmit bytes, how to collect and summarize evidence for links, how to express sets of statements, and what is the structure of compound principals.

REFERENCES

[1]. Howell and Kotz, End-to-end authorization, *4th Usenix Symp. Operating Systems Design and Implementation*, San Diego, Oct. 2000, www.usenix.org/publications/library/ proceedings/ osdi00/howell.html

[2]. Lampson, Protection. *ACM Operating Systems Rev.* **8**, 1 (Jan. 1974), 18-24, research.microsoft.com/lampson/09 Protection/Abstract.html

[3]. Lampson et al, Authentication in distributed systems: Theory and practice. *ACM Trans. Computer Systems* **10**, 4 (Nov. 1992), pp 265-310, www.acm.org/pubs/citations/ journals/tocs/1992- 10-4/p265-lampson

[4]. Myers and Liskov, A decentralized model for information flow control, *Proc. 16th ACM Symp. Operating Systems Principles*,Saint-Malo, Oct. 1997, 129-142, www.acm.org/ pubs/citations/proceedings/ops/268998/p129-myers

[5]. Oasis, Web services security, oasis-open.org/committees/wss.

[6]. Rivest, Shamir, and Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Comm. ACM* **21**, (Feb., 1978), 120-126, theory.lcs.mit.edu/~rivest/ rsapaper.ps

[7]. National Research Council, *Computers at Risk: Safe Computing in the Information Ag*e. National Academy Press, Washington D.C., 1991, books.nap.edu/catalog/1581.html

[8]. National Research Council, *Realizing the Potential of C4I: Fundamental Challenges*. National Academy Press, Washington D.C., 1999, books.nap.edu/catalog/6457.html
[9]. Saltzer, Protection and the control of information sharing in Multics. *Comm. ACM* **17**, 7 (July 1974), 388-402
[10]. Saltzer et al., End-to-end arguments in system design. *ACM Trans. Computer Systems* **2**, 4 (Nov. 1984), 277-288, web. mit.edu/Saltzer/www/publications/endtoend/endtoend.pdf

Amamer Khalil Masoud Ahmidat PhD. in the computer and information Faculaty of electrical Engand information technical University of Kosice Slovaki (2003-2008) . BSc electronic .Eng Computer Dept (1987-1991).MSc in Computer engineering and information Faculaty of electrical Eng.and information technical University of Kosice Slovaki (1995-1998) . **Head of Higher Institute of Medical Technology in Baniwaleed-Libya,** Assistant Professor from (01-10-2013.)

Dr. Amamer Khalil Masoud Ahmidat. “Concept of Computer Security." International Journal Of Engineering Research And Development , vol. 14, no. 05, 2018, pp. 08–16