# Best Practices for Secure Development

**v4.01, Oct 2001**

**Razvan Peteanu**

razvan.peteanu@home.com

main location of this document:

http://members.home.net/razvan.peteanu

## Revision History

| Version | Release Date | Notes |
|---|---|---|
| 4.01 | October 10, 2001 | fixed some typos |
| 4.0 | October 9, 2001 | 3rd public release, major revision |
| 3.0 | September 23, 2000 | 2nd public release |
| 2.0 | July 18, 2000 | 1st public release |
| 1.0 | February 2000 | 1st non-public release |

## Acknowledgments

Following the publication of version 2.0, the author has received useful comments and contributions from the following people (in alphabetical order):

Jesús López de Aguileta
Vittal Aithal
Toby Barrick
Gunther Birznieks
Matt Curtin
Ilya Gulko
Tiago Pascoal
Dimitrios Petropoulos
Steve Posick
Dave Rogers
Kurt Seifried
David Spinks
David A. Wheeler
David Woods
Greg A. Woods

This document has and can become better thanks to such feedback. No document can be comprehensive, though. The reader is highly encouraged to look at other works such as those listed in the References section.

## Legal Notice



### About the author

Razvan Peteanu lives in Toronto  and can be contacted by e-mail at razvan.peteanu@home.com

# Table of Contents

# 1 Introduction

## 1.1 Motivation

The following document is intended as a guideline for developing secure applications. It is not about how to configure firewalls, intrusion detection, DMZ or how to resist DDoS attacks. In short, it is not about infrastructure and network security. Compared to a year ago, the availability of consolidated material intended for developers has definitely improved but effort is still required to make the developer community more security-aware.

One part of the reason for this lack of security awareness is that traditionally, developers have worked on systems for environments where hacking was not considered a real threat: internal systems, call centres, software for home use, intranets. The complexity (and sometimes the unfriendliness) of the applications were adding to the barrier of entry. There may have been occasional exceptions with disgruntled insiders, sometimes with embarrassing outcomes, but they could be dealt with at HR level and the example prevented others from attempting it again.

Then the world wide web exploded and everyone had online web servers serving static pages. Web sites defacement went rampant (and still going strong). Still, the problem most often resided in a non-secure OS/server configuration. If a breach occurred, the system was taken offline, rebuilt better and the site put up again. Everything at a system administration level.

However, as the Internet has become more and more commercial (after all, this is where the .com comes from), web sites becomes more and more an application. B2C and B2B e-commerce became the buzzwords. There has also been talk about e-government. Cost efficiency has also pushed the market towards an online access only, while traditional channels such as mail or faxes are scraped. This makes sense for a lot of reasons, however, it also brings security to a very personal level. Leaked credit cards are a nuisance but a call to the credit card company can cancel a lost card and repudiate the transactions. Leaked health or credit information has long-lasting effects on the victims and this brings an enormous responsibility on the shoulders of the e-service promoters. It has also put a pressure on the development community to switch to Internet technologies. Because of lack of security training in traditional programming books and courses, these developers have not been prepared to build systems that withstand a focused attack. And it is not their fault. A single chapter about security in a programming book is not enough, just as one cannot properly learn survival techniques in a single chapter of a mountaineering guide. Such limited coverage also fails to convey the *mindset* and the skills of the attacker.

We hope this document will fill some of the gap. Do not expect though to be "the only security document you'll ever need". It is and will continue to be a work in progress and your feedback is highly appreciated. Also make sure you read the other works on this topic (see the *Other Resources* section). It does not matter where good practices are learnt from as long as they are learnt. You may also find an amount of overlap between this and the other documents. This is expected, after all “best practices” are not relevant unless they are shared.

This document is less intended to be about secure coding as about how building secure systems should be addressed at a slightly higher level. We will not stay away from code, though but in most cases, we will point the reader to dedicated resources.

## 1.2 Target Audience

- *developers and architects*: they are the people building the system and thus building security into it.

- *system administrators*: just as developers would benefit from learning more about the network & infrastructure, the people administering the latter would understand more about security risks if they become more familiar with what the dev team is working on.

- *project managers*: security does not come for free and the real world imposes schedules on software development. The effort required to build security into a system must be taken into account into the schedule. Since security is not necessarily immediately visible, project managers may be interested in finding where are those person-weeks going in.

- *business analysts*: these are the people who gather requirements from the customer. At the end of the day, security is for the customers' sake so having them involved from the beginning can only be beneficial.

# 2 Common Mistakes

The following are assertions that we often see (as users of a system) or hear (as developers).

- *"Your shopping experience is 100% secure through the use of powerful 128-bit encryption"*
- *"We need security? Then we'll use SSL"*
- *"We need strong authentication? PKI will solve all problems"*
- *"We use a secret/military-grade encryption"*
- *"We had a hacking contest and no one broke it"*
- *"We have an excellent firewall"*

and the all times favourite,

- *"We'll add it later, let's have the features first"*

Some of them are plain mistakes, others are over-simplifications. We hope that this text will (in)directly explain why.

# 3 Security in a Project's Lifecycle

## 3.1 Introduction

Inasmuch as a software project does not start with coding, building security into an application does not start by implementing security technologies. We will suggest an approach recommended by the existing risk management and software building practices.

We will loosely follow and extend the Rational Unified Process view of a project's lifecycle, but the readers will easily map the concepts to their own methodology. The first four phases listed below come from RUP whereas the last two come in after the software was built.

- *inception*, in which the vision and business case for the end-production and the scope of the project are defined
- *elaboration*, in which the planning, features and high-level architecture are defined
- *construction*, in which the product is built alongside with refinements of the architecture
- *transition*, in which the built product is made ready for the user community, including manufacturing, delivery, training and support
- *operation*, in which the software is actually used.
- *disposal*, in which, for a number of reasons, the system has ended its life and the hardware, software and data will be disposed (through archiving, destroying or otherwise)

We will not cover the last two phases because this document is focused on building not operating the system. There are other works that cover these two stages. A good starting point is NIST's *Engineering Principles for Information Technology Security* [Feringa2001].

## 3.2 Inception

In the inception phase, security is not yet a topic of discussion unless the goal is an explicit security product (such as encryption software). Even then, however, it is too early to discuss security risks or technologies. This phase is intended to roughly scope what value the product will bring.

It may be useful, however, to recognize the importance of security for the later stages and include a member in the team who will be responsible for the security aspects.

## 3.3 Elaboration

### 3.3.1 Requirements Gathering

This stage is not specific to security but a normal step in building any project. There are entire books dedicated to this topic. The security-related information should be captured at this stage?

- the *assets*.

- the *actors* and the *use cases*. (if the terminology is unclear, you'll benefit from doing some reading on UML). Who uses what and how is essential to understand the security implications. Remember that at this stage actors are roles: we are not concerned about specific users but rather classes of users (guests, regular users, administrators etc).

- legal and business requirements: does the environment impose certain restrictions or policies? Is there a formal certification required and if yes, to what level of compliance? Should the system support non-repudiation? An audit trail? If digital signatures are envisioned, what is their legal status where the system will operate? Will there be any encryption restrictions? (fortunately, this is not as acute as it used to be but still it is worth checking)? How does the existing security policy affect the system?

### 3.3.2 Asset Identification

The first step is asking WHAT are we going to protect? The question is deceptively simple and may produce quick answers but understanding what is at stake is critical for building a proper and relevant defense. If you start considering what is important for your business, you will notice the assets fall into two categories.

- *Tangible*: computers, communication systems

- *Intangible*: information (stored in or vehicled through the above), trust, the availability of the site itself (wouldn't it matter if the site is down for two days?)

Tangible and intangible assets are often related. Sometimes they are closely coupled. A laptop theft may also compromise its data (unless encrypted), a stolen system containing private information also affects customers' trust and so on. At other times, the assets are related but weakly coupled. For instance, stealing credit card information from a database does not necessarily mean stealing the systems.

Identifying the assets is rather easy and perhaps the biggest trap is to limit the team doing it to one area only (i.e., by asking technical only or business only teams). Involve all stakeholders.

### 3.3.3 Asset Valuation

Given that we live in an information economy, we shouldn't be surprised if we notice the intangible assets matter more. How much we value them is the outcome of asset valuation process. The reason will become clear later, but for the time it is a good exercise in itself because it will clear up and level the perceptions. Evaluating the tangible assets is rather easy

and perhaps not very attractive. Obtaining the value of a system may consist of sending an email to accounting, considering the physical and moral depreciation etc. Evaluating the intangible assets is a different game, though. How to evaluate the customers' trust or the information on CEO's laptop? You could attempt to think in terms of lost business (if 35% customers decide to switch to a competitor or the cost of law suits if customer data is lost). However, before we launch into economics theories, it's important to stress out that we do not want to lose sight of the forest because of the trees. Our end goal is to build the security architecture and not to perform a financial audit. Therefore, if at this time we do not know the value of the asset, a qualitative designation such as "low", "medium" or "high" or quantitative ranges (<1000$, 1-10K or anything that suits your needs) would do fine. If we need it refined further, we can always come back.

### 3.3.4 Threat Analysis

The threat analysis is the process of identifying [as many as possible] risks that can affect the assets. Note that we did not say "all risks" because such omniscience is rather out of our reach. A well-done threat analysis performed by experienced people would likely identify most *known* risks, providing a level of confidence in the system that will allow the business to proceed.

Once the threats are identified, there are three commonly recognized approaches:

- **Acceptance:** with or (less preferable) without a contingency plan. A contingency plan (the famous "plan B") provides an alternative if something went wrong, but does not address the risk directly. Example of acceptance: "we will use password-based authentication and accept the risk of some users being impersonated"

- **Mitigation** of the risk. The risk is not accepted as it is and measures are taken to reduce it to an acceptable level. Most security solutions fall under this category. Eg: encryption reduces the risk of disclosure.

- **Insurance:** this transfers the risk to another party (the insurance company). It is traditionally used to address disasters that cannot be avoided or reduced to an acceptable level. Eg: protection against the financial effects of an earthquake.

As mentioned above, this document is primarily concerned with mitigation strategies.

### 3.3.5 Trees and Tables

A good way to capture the threat model is by using **attack trees**. [Schneier1999a] and [Schneier1999b] explain this concept. If the attack tree becomes hard to follow, the final results can be captured in a table such as below. This approach works reasonably well for low-ceremony projects.

| *RiskID* | *Asset* | *Asset Value* | *Cost per incident* | *Incidents per year* | *Cost per year* | *Risk Strategy* |
|---|---|---|---|---|---|---|
| | Asset1 | | | | | |
| risk1 | | | | | | |
| risk2 | | | | | | |

(shaded cells are not used)

Where the columns mean as follows:

- **RiskID**: an identifier for easy reference to the risk. String identifiers are preferred over numbers because if they do not require renumbering (if a risk is inserted between the two) and reordering the risks does not affect the references.

- **Asset**: the name of the asset

- **Asset Value**: the value (can be qualitative like low, medium or high or quantitative if the figure is known)

- **Cost per incident**: the cost of one successful occurrence of the attack (again, it can be quantitative or qualitative)

- **Incidents per year:** how many such incidents are expected to happen within a year. Can be qualitative as well (like 'few', 'some', 'many')

- **Cost per year:** the previous two columns multiplied. If qualitative values were used, you can use conventions such as (high x medium = high, high x low = medium etc)

- **Risk strategy**: a simple marker for the strategy to take: Accept, Mitigate or Insure

Of course, take the above as a starting point. This approach has a shortcoming: the links between the various attacks and gaps in the analysis are easy to miss because of the table format.

An alternative approach to threat identification was proposed in [Howard2000a]. The STRIDE model attempts to identify the threats by analyzing several categories: **S**poofing identity, **T**ampering with data, **R**epudiation, **I**nformation disclosure, **D**enial of service and **E**levation of privilege.

### 3.3.6 System Architecture

As far as this document is concerned, the system architecture should provide a clear understanding of how security is provided by and inside the system, how trust is managed and how the system would withstand the major attack paths. It should contain a high-level threat model and references to the related documents describing the security processes. Major technologies used should also be represented (eg. the system uses authentication and delegation through Kerberos) but care should be taken that the technologies do not become

sole guarantors for security. "We use Kerberos therefore we are secure" is a mistake. Explaining how Kerberos fits into the big picture and how it provides some of the system's security services is more to the point.

Include here the protocols used within the subsystems in a distributed environment. Protocols are notoriously difficult to get right from a security standpoint (just look at how much time it took SSL or IPSec to mature) so the choice of one protocol or another is very relevant to the security architecture.

The system architecture should include the high level infrastructure architecture as well. Firewalls are perhaps the immediate example, but other infrastructure elements or decisions affect security decisions. For instance, using a load balancer may influence how SSL certificates or hardware accelerators will be used. In another example, routers doing NAT (Network Address Translation) may break some protocols that are not NAT-compatible (IPSec in tunnel mode and NetBIOS are two such cases).

### 3.3.7 Legal Requirements

I am not a lawyer and cannot really advise on this topic. The recent years have seen a increasing role for legal counselling, especially when cryptography, privacy and post-incident handling. Several resources the reader might be interested to start with the following resources:

- general Internet law: the *Internet Law and Policy Forum* at http://www.ilpf.org/
- laws on cryptography: the [Bert-Jaap Koops' *Crypto Law Survey* http://cwis.kub.nl/~frw/people/koops/lawsurvy.htm, updated twice a year
- privacy: Europe's *Data Protection Act*, http://www.dataprotection.gov.uk, Canada's *Personal Information Protection and Electronic Documents Act* and the US Department of Commerce *Safe Harbor* http://www.export.gov/safeharbor/

Another legal requirement can be formal security certification. We will briefly cover this topic in a dedicated section later.

## 3.4 Construction

Construction is the phase where most developers spend their time (and overtime…). This is when the detailed design, implementation and testing of the software take place.

### 3.4.1 Detailed Architecture & Design Patterns

Identifying patterns in a world of problems to solve is a significant step towards solutions (What a platitude I've just written, haven't I?). Any experienced developer will have at least one book with the word "pattern" in its title. Fortunately, after years of neglect (or just maturation) , application security also benefits from identified design patterns.

[Barcalow1998] is the first work (to our knowledge) that documented a number of design patterns for security (namely *Single Access Point, Check Point, Roles, Session, Full View With Errors, Limited View* and *Secure Access Layer*). A commentary on it (<shameless plug>

[Peteanu2001a]</shameless plug>) was written earlier this year and in the correspondence that followed, it turned out that there has been work on this topic. There is in fact a web site dedicated to security patters, http://www.security-patterns.de, where the reader can find a number of research papers. [Romanovski2001] is another (and recent) paper that takes the original Yoder & Barcalow paper and extends it into best practices.

The Open Group has announced a *Guide to Security Patterns* document for November 2001 ([Opengroup2001]) , but since this document was written before the publication date, please check their web site for details.

### 3.4.2 Implementation

"This section will teach you everything you need to know to make secure implementations." Wish this were true, isn't it?

This section will be very short because secure coding practices, one of the underlying reasons of this document, are covered throughout the entire paper.

### 3.4.3 Testing

There are two aspects regarding security testing. One of them is the testing performed by the QA team. The other is the formal evaluation required for certifications.

#### 3.4.3.1 Security, Testing and QA

In a project's life, most of the testing is performed by the Quality Assurance team. (We say most and not all because the unit testing is normally performed by developers). Software testing is a specialty in itself, but at its very core, testing will ensure the software does what is supposed to do. Or, more precisely, what it is required to do.

Our requirement is, of course, to have a secure system. However, this is much too vague to be useful as a requirement. The requirement can be detailed (eg. the system should be immune to this attack and that attack) but, if you look closely, we are only defining security by negation. All of the attack conditions can be tested, but this only guarantees those attack paths are not successful. Traditional testing cannot guarantee the security of a system against *any* conceivable attack, essentially because no one can have this omniscience. (see the works referred to in the section on "Thinking outside of the box" to learn how innovative some of the attacks can be).

Does this mean that security testing is a futile exercise? Absolutely not! Inasmuch as absolute proof of security cannot be obtained, omniscient attackers do not exist as well. The overwhelming percentage of attacks a system will see are *known* attacks and thus whether a system is vulnerable can be determined prior to launching the system.

The other practical difficulty is having relevant testing, that is, integrating those known attacks into the security testing plan. This requires testers to have skills and knowledge to attack systems, which is not exactly a common scenario.

Not that such people do not exist. On the contrary, there are companies or team specialized on "ethical hacking" or "penetration testing" as the domain is called. SecurityFocus maintains a very interesting mailing list on this very topic. It is just that the people who know

how to do pen-testing are not that much involved in testing a product. They are often called to attack an existing system after it has been deployed.

An up-to-date and structured repository of application vulnerabilities is essential for such testing to be relevant. This is not the regular product-focused repository (a la CVE), that is useful, but we will assume the system is up-to-date in terms of patches. Rather, what is needed is a repository of classes of attacks so given any system, the tester would know *what* to try. A promising initiative is the *Open Web Application Security Project* at http://www.owasp.org. Another useful resource is [Wheeler2001] and the *The Open Source Security Testing Methodology Manual* [OSSTMM2001] although the latter is geared more towards system penetration testing.

#### 3.4.3.2 Certification

In more regulated environments systems are required to be security-certified. The meaning of such certification is specific to each evaluation program. Just because a system is certified does not mean it is secure. Some of the certifications assess the security *capabilities* of a system, such as whether it has authentication or authorization services and not whether it cannot be hacked. At other times, the "fine print" matters (when doesn't it? :-). For instance, the old Windows NT 3.51 was C2-certified, but only as an individual workstation, without the networking components.

Security certification already has a history. The 80s saw the US developing the *Trusted Computer System Evaluation Criteria (TCSEC)*, also known as the Orange Book. Europe followed with the *Information Technology Security Evaluation Criteria (ITSEC)* in 1991. Canadians had their own flavour, the *Canadian Trusted Computer Product Evaluation Criteria*. With the increasing globalization of business and the necessity of having multi-national systems compliant to local regulations, these efforts were bound to be drawn to a unified approach. This is the *Common Criteria* which, in its authors' own words, "defines general concepts and principles of IT security evaluation and presents a general model of evaluation. It presents constructs for expressing IT security objectives, for selecting and defining IT security requirements, and for writing high-level specifications for products and systems.". CC's main site is at http://www.commoncriteria.org/

This type of certification has actually less to do with testing than with requirements and design. If your system needs such certification, you will surely know already more about the process than what is the short paragraph above so we will not spend more time on formal certification here.

## 3.5 Transition

Transition is the phase in which the software is given to the users and –when applicable– to the IT Operations department who will maintain the system. As far as information security goes, there are a number of steps to take at this stage:

- make sure the technical documentation describes the security-sensitive aspects associated with the system

- check whether the legal documents (such as a privacy policy) have been finalized

- make sure the documentation covers all security-relevant information. This is very important for the end-users or maintenance (as applicable) because this data will be used in their own risk assessment. Where and how are credentials stored, what encryption mechanisms are used, what kind of information is logged, whether temporary files are used, how security settings can be pre-set at installation/deployment time etc

- check that the process for handling security bugs has been defined and are all teams aware of the emergency procedures to be followed. For instance, if a critical security hole is discovered, is it known what happens after the fix itself, that is how the QA team will integrate it into their testing schedule, how customers are notified, how the update will be distributed etc.

- make sure the documentation indicates what data is to be backed up. This is not a security-specific requirement, but if certain data is encrypted and the encryption key is not backed up, the data will be useless. You may also indicate the sensitivity level of the data.

# 4 Principles

Technologies come and go, as anyone working in the IT industry is keenly aware of. Before we start looking at the trees, let us step back and look at the entire forest we want to cross safely. Below are some suggested guidelines.

## 4.1 Relevancy & Weakest Links

Security solutions are intended to server a very practical purpose: protecting a system or data from unauthorized use and disclosure. This means (a) ensuring the real issues are addressed and (b) not putting security where it does not solve a problem.

The first is often referred to as focusing on the weakest links because they, especially the *known* weakest links, will be the first an attacker will try. There are fears about the 40-bit SSL or the "gap in WAP" but few if any attackers will choose these difficult paths when many systems have easier to exploit holes.

As far as the second implication goes, it is useful to remember that unlike other features that may originate from marketing needs (so the hot technologies appear on the box), security mechanisms should be risk-driven. Mitigating risks is the main goal of information security. Too much security in the wrong area does not address them. If the above goal is not reached, interested parties will make the real issues surface sooner or later and the writing on the packaging will not mean much.

## 4.2 Clarity

A security solution should be crystal-clear in terms of technology and processes. Vagueness means the system behaviour is not fully understood, not even by the people who are supposed to build it. How could they then make it secure?

We also admit that in practice, the crystal clarity will not be found. Especially when people are involved, mistakes occur, information is forgotten and so on. However, imperfect implementations of good designs are far easier to fix than imperfect implementations of poor designs. Therefore, strive for clarity in requirements and designs and hope for the best.

## 4.3 Quality

Security is more a quality of the system than a matter of [security] features. Much like the human immune system, it's not about how many types of T-cells are there but how well it behaves overall to an infection. Security features do not necessarily increase the security of the system. They *would* if properly done and if relevant to the identified risks, but complexity brings its own risks so a system implemented with less features but greater care may end up being more secure than a complex system which had little quality assurance.

## 4.4 Involve All Stakeholders

Software does not operate in a vacuum and today's systems often span multiple machines spread across multiple networks. Such complex systems are managed by teams of people and often serve business purposes. Thus, apart from the development team, the systems (and therefore their security) are also relevant to the infrastructure team as well as to the business units. Involve all of them: the former need to understand how the software running on different machines interacts whereas the latter should be part of the risk management process.

Gaps in communication among the different stakeholders can result in security problems. We often read complaints about management not allocating enough funds for security. Well, at least one reason could be that the risks perceived by management are lower than those seen by the security team. Disfunctionalities also arise when the infrastructure team is not aware of what the running software does or, conversely, when the development team is not aware of the infrastructure constraints.

## 4.5 Technology and Processes

Light is both a particle and a wave and you can't separate the two aspects. Similarly, security is both technology and processes. (Or if you ask Mr Schneier, "security is a process"). We often focus on the technology, for a number of reasons. Perhaps one of them is that technology sells much better than processes so security vendors push software, even software that "intelligently" and "automatically" takes care of the process part. Research in this area is very welcome, but like car driving, there is still need for human brains and human interactions to solve security problems.

There are quite a few resources on security processes, especially on incident handling and response. Since the purpose of this document is on building secure software, the processes we refer to are development-specific. Several examples:

- if passwords are used, is it possible to have them reset by an administrator and/or help desk personnel (with fewer rights)?

- if certificates are used, is it clearly defined how the subjects apply for, receive and revoke the certificates? What about the other PKI-related processes?

- is there a process for handling reports of security vulnerabilities? Triaging the reports, dispatching to the right team, escalating the issue, keeping in touch with the people who reported the problem, issuing a patch, working with the QA team to do the regression testing before the patch is released and, of course, handling the media.

## 4.6 Fail-safe Operation

Systems fail. Hardware fails and software fails. But like in sport, it matters how you fall to the ground. In security, systems should have a fail-safe behaviour: if the security subsystem becomes non-operational, it should get to a state where the security is not [critically] affected. Functionality will certainly be, but there is a world of difference between a door that cannot be opened (inconvenient but secure failure) and one that cannot be closed. The

typical example is a firewall: if the firewall process crashes, the machine should be in a state where network traffic between interfaces is blocked.

This is not only a "in the unlikely case of a system failure" scenario. If a particular system is known to be vulnerable to such cases, you can rest assured attackers *will* attempt to crash the system, thus opening the gates.

## 4.7 Defense in Depth

This is the formalized version of the "Never put all your eggs in one basket" old saying. Systems and software fail and security systems or software are no exception. The reasons are numerous: design or implementation flaws, operational or human mistakes, new attacks, bad luck, you name it. The practical result is that if there is one and only security mechanism in place, one day that mechanism will fail, totally exposing the system. It is much wiser to design multiple layers of security mechanisms so even if one of them fails, the damage is reduced.

Interestingly, today's multi-layered, distributed systems make it difficult not to implement defense in depth. The technology in each layer often has its own security mechanism. If you take a typical web site, there is a web server talking to an application server which in turn talks to a database server. We already have 3 layers at which we can enforce authentication, authorization, confidentiality and integrity. The application server should not allow any client to connect to it nor should the database server. Yes, there will be authentication to the web clients, but passwords should also be stored encrypted in the database in the case someone breaks into it. A typical defense-in-depth deployment will also have a firewall between the public network and the web server and others in between the other layers. Well-designed policies would also reduce the amount of damage in the system. For instance, even if the web server is compromised, if the firewall separating it from the application server only allows SOAP traffic, the attacker will find it difficult to penetrate into the system further as opposed to the case when traffic was not restricted and telnet traffic was allowed.

The above should not be construed as "a multi-layered architecture" is automatically using the defense-in-depth principle. The mechanisms that may be offered by the various layers still need to be used (and used properly).

## 4.8 Principle of Appropriate Privileges

The usual phrase most readers will be familiar with is "principle of least privilege" and indeed, out of the two this is the more important. However, since we are considering the development phase as well (not only the final product), the second recommendation becomes relevant.

### 4.8.1 Principle Of Least Privilege

You have probably read this many times: *only grant the minimum set of privileges required to perform an operation for the minimum time*. This refers to:

- privileges of accounts used during development

- privileges of accounts used at runtime.

The most frequent violation of this principle occurs when the application or the developer uses administrative-privileged accounts for regular tasks. Even samples encourage the programmer to use "system", "sa", "administrator", "root", "sysmgr" or their equivalent. Why is the administrative login not good, even in a secure environment without any sensitive data? Several reasons:

- it prevents application isolation

- it prevents accurate testing: since administrative accounts are not subject to the usual restrictions, the authorization functionality is not tested so bugs can lurk in. If the production environment will have a proper security policy, then likely some functionality will break.

- it prevents proper accountability: an administrator is still accountable for his/her actions. If the account is shared by applications and/or non-admin users, tracing incidents becomes daunting.

- it is plain dangerous: let's face it, we can and we do make mistakes. A mistyped parameter to a delete command can wipe out other directories or databases. If an account with limited privileged to the necessary resources is used, then the chances of inflicting damage are minimized.

### 4.8.2 Principle Of Necessary Privilege

If authentication and authorization are needed, then by all means, use them during development. A mistake opposite to using too much power is using too little. In practice this often consists of using anonymous/guest access even when the final application will not allow such lower privileged accounts. It can occur with web applications (when the web server is not configured to enforce authentication) or distributed applications where passing null credentials is often allowed and the caller impersonated with a default account.

Why is there a problem when using anonymous access? For a number of reasons:

- passing credentials may require a different syntax than the anonymous access. For instance, to access a remote resource on a Windows network, besides a username and a password, a domain name may be required, a field that is not required when connecting anonymously.

- it may hide various authentication/authorization/impersonation issues

- using anonymous access to resources also means that the code responsible for authentication/authorization is not actually used (or not as intended). If it's not used, it cannot be tested. If it cannot be tested, bugs cannot be discovered until later.

## 4.9 Interacting With Users

Human users are part of most systems involving security. Furthermore, they are an *active* part: they login and logout, they make requests, they make decisions to trust or not, to proceed given a risk or not. Equally important: human users *make mistakes*. Even given correct and full information, they may still take the wrong decision.

### 4.9.1 On Relying On Users

As the real world repeatedly proves, users make poor choices when it comes to security. Just think of the Melissa and ILOVEYOU viruses. There are many reasons, ranging from the impossibility of having all users security-knowledgeable to the human nature of seeking quick results despite negative long time effects.

### 4.9.2 Help the Users

- Do not annoy users by uselessly asking for security information. Repeated login prompts instead of temporarily caching credentials do not increase the security as it might be expected. On the contrary, users become insensitive to password prompts and will automatically type in any prompt resembling the known dialog box, thus increasing the risk of vulnerability to malicious code.

- Do not allow poor passwords. It is much more efficient to enforce a simple password strength check when a user defines a password than presenting a long list of recommendations on how to select a good password. Unless such recommendations are enforced, users (who only in the best case had time or interest to read the text) will select an easy password. A simple regular expression pattern matching can go a long way in helping the user choose a better password. Also, be careful not to impose arbitrary restrictions on the secret value, such as digits-only or max 8 characters when there is no justification for such restrictions. Be aware of semantics: prompted to enter "a PIN" instead of a password many users will provide a 4 to 6-digit numerical value because so they are used with ATM PINs. The term "passphrase" conveys the possibility of a longer secret.

- Choose the default settings to be the more secure ones. Many applications compromise security in favour of an "allow everything to happen so the user will be happy the software is powerful" approach. Given many users will not change the default settings, having more secure default values reduces the number of potential victims. For instance, Outlook and Outlook Express come with the default security zone set as "Internet Zone" which allows active scripting to run in HTML emails. Thus scripted virus have chances to be executed.

- provide plain language prompts. "Do you want to allow this ActiveX object to run?" is not very helpful to anyone who is not aware of what ActiveX or even an object are. If the user does not understand, the prompt did not serve its purpose.

- explain the risks. If the user still has to be prompted to make a security decision, at the very least explain the level of risks (eg. "choosing Yes can potentially allow malicious actions on your computer").

### 4.9.3 User Interfaces

Some principles:

- hide sensitive information. Everyone is familiar with the asterisks instead of passwords characters, but even here there are nuances. Make sure:

  (a) when *displaying* asterisks instead of password characters, make sure the real password is not stored under the hoods. A number of Windows applications have been trivially attacked because they actually put the password in text boxes and only changing the style of the text box to "password", thus causing asterisks to be displayed. However, changing the text box style to normal with an external tool revealed the password.

  (b) if the number of asterisks is matched with the number of characters, a quick glance provides useful information to an attacker, especially if it is low. Better display the same number of * every time the placeholder for password is shown. After all, there is no real need to display *'s at all, but users often expect that as a confirmation a password has been set.

- have "honest" interfaces. If there is a logout function, then it should really mean the user has been logged out and the session terminated. A number of web applications have been found vulnerable to such a problem: while the user clicked on the Log Out link and got a confirmation page, the session was not actually terminated on the server so subsequent users of the same terminal could simply click on the browser's Back button to gain access to the user's account.

## 4.10 Trust Boundaries

Any useful information system interacts with the environment. And any complex system will have layers and subsystems. A security architecture should capture which of these entities trust whom and what for. One subsystem can trust another for authentication but not for authorization (a typical scenario for a J2EE architecture in which the request is handled through a web server which then propagates the caller identity to the EJB server).

If identifying the trust boundaries appears difficult, this can be a sign that more work is needed on understanding the technology or the requirements. The more complex the system, the more attention is needed.

### 4.10.1 Trusted Computing Base (TCB)

[TCSEC1983] defines the TCB as "*the totality of protection mechanisms within a computer system, including hardware, firmware, and software, the combination of which is responsible for enforcing a security policy*. Note: The ability of a trusted computing base to enforce correctly a unified security policy depends on the correctness of the mechanisms within the trusted computing base, the protection of those mechanisms to ensure their correctness, and the correct input of parameters related to the security policy".

The TCB is a useful concept because it identifies, within a system, the subsystem which "owns" the security. The rest of components will call this TCB and rely on it to make correct security decisions. The implication is two-fold: (a) that there is such a defined

subsystem that makes the security decisions and (b) that no security decisions are made outside of the TCB. A TCB that is called for 90% of the decisions while the remaining 10% are made by various modules is not a true TCB.

Now for the reality check. It is wonderful to have this TCB black box and everyone calling it. The challenges are (a) how can we ensure the other modules really call the TCB and (b) how scalable is the above model?

As the reader suspects, if the calls to the TCB are left at the discretion of the other systems, the TCB will become something of an advisory, but definitely not a policy enforcer. Which is not good. The correct way to enforce a TCB is to design the system in such a way that the TCB becomes a "Check Point" (see the section on Design Patterns), the only way for one subsystem to interact with another one. Such a patterns occurs in object-oriented execution environments like EJB, COM or CORBA. The objects do not talk to each other directly, but through the container or COM runtime engine, respectively. Thus all method calls unavoidably go through a layer where the security checks can be enforced. The same model can be used with regular applications: identify what objects or modules are candidate for Check Points and plug the calls to the security component there.

We also asked whether the TCB model scales. Well, not without bending what the TCB is a bit. In today's n-tier, distributed systems, policy enforcement is no longer performed by a single entity. There will be multiple and partially overlapping security domains. A typical web application will have the web server, we have an EJB- or COM-based middle tier and then a database server. Each of them enforces its own authentication & authorization policy, relevant to its tier. The TCB becomes distributed and harder to pinpoint. The security module in a web server might be a separate entity for its developers, but for an integrator, it is the *functionality* and not the actual packaging boundary that becomes relevant. The TCB has become more virtual.

As the DoD definition indicates, if the TCB becomes corrupt, the entire security can be considered broken. An out-of-system intervention is required to assess whether this is the case and the extent of the problem. The typical example is when an intruder acquires root access on a Unix system, thus compromising the system utilities, the logs and so on. The only safe way to assess the intrusion is to compare the data with known good backup copies and doing this check on a known clean system (thus ensuring everything is happening under a correct TCB).

### 4.10.2 Trusted Path

We have previously discussed what the TCB is and said other layers in the system would interact with the TCB. For instance, the TCB would authenticate a user given the username and the password. Users would have a very legitimate point of view: how can they be sure they are talking to the TCB when providing their credentials?

This is where the concept of a "Trusted Path" comes in. TP is a direct communication path between a user and the TCB which the user can trust. Probably the most familiar example would be the often misunderstood Ctrl-Alt-Del sequence in Windows NT/2000. This sequence (called "secure attention sequence") initiates the login process and Windows has been designed so that this key combination cannot be trapped by a trojan application intending to impersonate the logon process. Thus, no matter what application the user is running, pressing those 3 keys is guaranteed to establish a path to the Windows TCB.

When building a system, consider if there is a need for a formal trusted path. It is a nice thing to have, but not easy to design and the decision to implement one should come after a threat analysis process.

## 4.11 Third-Party Software

Few systems today are built from scratch, most often new code is integrated with existing components or subsystems. When assessing the security for an application, do not forget to include these entities in the threat analysis. Anything from a small component (that could have a buffer overflow) to a different server running on the same machine (which could still have a buffer overflow but many other holes as well because it is more complex) could ruin the security of the system. Such vulnerabilities can be more damaging that the "expected" ones and because they can compromise the entire machine instead of an application only, they can be far reaching in consequences. As an example, gaining root/administrator access on a web server allows changing configuration or system files, installing a sniffer and so on.

As a reality check, assessing the security quality of a third party product (especially if source code is not available) is extremely difficult. Often the project's timeline does not afford including a full security review of third party products. If their vendors did not do it (and they had full access to design and source code), how can a user even hope to do a comprehensive job?

The solutions at hand are not really direct solutions, but the best it can be done with limited information and time:

- assess the overall feeling of how "professional" the product looks. Less bugs, reliable behaviour, better documentation and other tell-tale signs of a mature software development cycle. This does not ensure security, but increases the chances of having a quality product.

- assess the company and how it historically reacted to security issues. Do they have a support site with downloads and patches? Having no patch or service pack is necessarily not a sign of quality, it may simply mean the vendor has not gone to the effort of providing one. Ask the vendor what they do if there is a security report, whether they have a process and read between the lines when you get the answer: does the reply seem to indicate a coherent process behind it or does it appear to be the product of a marketing spin, obviously taken off-guard by the question?

- assess whatever security features are there: how are the password dialog boxes, run a sniffer to see if what goes on the wire has obvious flaws, try some well-known security holes such as buffer overflows, cross-site scripting, SQL injection etc. Look at the configuration files or in the registry, look at the files with a hex editor or dump the ASCII/Unicode strings in search for obvious errors such as hard-coded credentials, perhaps some hidden debug commands etc.

- search the Internet for vulnerabilities for the product, for other products by the same vendor and for other products in the same class. If you find any, try them. Also, ask the vendor if they guarantee there are no buffer overflows (or a similar risk) left.

As you see, it is more of a strategy of exclusion: nothing gives you the assurance of security, but you can find reasons not to trust the product. Still, even with the above, you may need to go with a certain product or a certain class of products. What then?

In this case, assume the worst will happen and the product has a security hole. What would you like to be able to do the day you hear the news? Besides checking for updates, you'll want to be able to change the product with something similar. Here's where a good design pays off: if the product's functionality has been abstracted through another layer, it will be easier to make the transition. This is more applicable to components than to entire subsystems. Look at the design patterns section for more information. The API/SPI pattern is a good example which allows changing a component's implementation without affecting the rest of the application.

Another desirable feature or rather knowledge in this case is the ability to isolate the vulnerable piece in the third-party software and/or to monitor access to it so at the very least you will be notified when an exploit is attempted. This is where good documentation and support from the vendor comes in. If the communication is documented, you may be able to tune IDS filters for the particular attack that was reported. If on the other hand the communication is proprietary and undocumented, the task is daunting.

## 4.12 Manipulation of Sensitive Data

Most often such data consists of authentication credentials or credit card data, but by no means should only these data be dealt with care.

### 4.12.1 Protect or Avoid

This is a strategy we will try to follow in dealing with sensitive data: either protect it (properly) or avoid manipulating it directly. The latter case is not that impossible as it may sound.

### 4.12.2 Generation

Passwords can be chosen by the user or by the system. The former are usually weak but have the advantage of being easily remembered by the user. The latter are just the opposite: strong (at least they should be!) but hard to remember thus leading to passwords written down or many support calls to reset them (which, apart from increasing the cost of the system, can also add security risks if someone impersonates another user and succeeds in changing the password).

A compromise could be reached by still letting the user choose a password, but not *any* password. The password strength should still be validated by the system. Such a solution has already been implemented for several major operating systems such as Unix or Windows NT/2000.

### 4.12.3 Storage

"Protect" strategy: guard the data itself (by encrypting or hashing it) or the medium (by using secure storage mechanisms such as smartcards).

That is the theory. In practice, there are pros and cons for each approach: the former is cheaper and software-only, so updates to the scheme are feasible within acceptable limits. The downside is that the risk is still there: symmetric encryption requires a key which has to be protected in turn whereas hashing does not protect from brute-force attacks, so weak passwords are still weak, even hashed. Salting the hashing process increases the time needed to brute-force many hashes with different salts, but for one password only the effort is the same.

Hardware token like smartcards are more secure, but they impose deployment and operational problems. Customizing, revoking, replacing lost cards etc are real-world tasks that need to be addressed successfully in order to make a system accepted by users.

"Avoid" strategy usually means delegating the responsibility to another system. For instance, credit card payments are often processed through a server-side component with the bank, the e-commerce site only receiving the status of the transaction and non-sensitive confirmation data (such as the transaction confirmation number, perhaps a few digits from the card etc). Unless the online shop wants to alleviate the customer from entering credit card data every time, the shop can go on about its business without storing credit card data.

### 4.12.4 Transit

A classic problem in authentication mechanisms is how to protect the credentials during the login process.

"Protect" strategy: perform the communication over a secured channel (such as SSL or IPSec) or through protected packaging (such as when sensitive information is encrypted, but not the entire traffic). This basically delegates the problem to another layer.

"Avoid" strategy: instead of passing clear text credentials, use challenge-response or other mechanisms that prove possession of an information without sending it. Note that a challenge-response mechanism does not automatically ensure security: a poorly designed protocol will only lead to a false sense of security. Case in point: Microsoft NTLM. Cracking NTLM is the raison d'etre of L0phtCrack.

## 4.13 Attack first. On paper!

During the design of a security architecture, try to break it. Almost guaranteed there will be issues to address after the first rounds of attacks. If nothing was found, try again or ask others (see the section on reviews). Get informed on attacks against similar schemes and pay attention to the approach more than the details. If possible, leave a gap between the time spent on design and the attack. Some attacks require some lateral thinking and a rested brain is definitely better positioned.

## 4.14 Think "Outside of the Box"

Security attacks are not played "by the rules". Neither by the rules of fairness nor by the rules of conformity. If some data is protected by 128-bit encryption, this does not mean the attacker will choose to focus on breaking the encryption algorithm. Good attackers do not follow stereotypes. Sometimes they will find very imaginative ways of getting around the

security mechanisms. This often is the result of the defense designer being caught – unintentionally - in a mental trap about the system. In another shameless plug, we will refer the reader to a 2-part text (see [Peteanu2001b] and [Peteanu2001c]) where side-channel attacks are discussed. The "arsenal" is not your typical collection of sniffers and hacking tools, but rather precise timers, measurement equipment and … a bio-lab.

## 4.15 Be Humble!

Perhaps not the expected advice in a technical paper, but we found it useful in the real world. As with military strategy, underestimating the adversary is a costly mistake. Whether a system is secure or not can only be determined by using it in real situations. Security architects should never leave backdoors in the system because "they are well hidden and nobody will find them", history has proved such approaches are flawed. Nor should we assume that the sheer complexity of a system will make it incomprehensible (or even boring) to an attacker. And finally, perhaps the most common mistake, we should not assume that if we designed a system, it is flawless. Design reviews should be mandatory. It is human nature to build a complex work iteratively, by constantly improving the design and correcting previous errors, instead of doing it perfect the first try. We could wish we were that way but we are not.

## 4.16 Declarative vs Programmatic

If security restrictions are to be enforced during the execution of a program, there are two approaches: either the program checks the conditions itself and then takes the decision or the execution environment does it outside of the program's control. Typical examples: an application written in C which manages its own resources uses programmatic security. A Java application running with a security manager loaded, a COM+ component or an EJB all run within an execution environment which enforces the access control according to policies set outside the code in question. There are pros and cons for each, but generally security is better when the declarative approach is used.

There are a number of reasons for this recommendation:

- consistency: if the execution environment enforces a policy, it should do it uniformly. A code-base approach will have differences, subtle or not.

- flexibility: if security decisions are made in code, changing the behaviour or the policy will likely require a code change. A declarative approach only requires changing the policy

- separation of roles: often the security behaviour is to be defined by different people than those who developed the code. They may not even know each other, such as in the case when a third-party component is integrated. A declarative approach will allow the integrator to define the security behaviour of the component without knowing implementation details.

- security expertise: writing code for security mechanisms is not trivial. Leaving security decisions to developers who do not have experience and expertise in this area will surely lead to security holes. It is better to have the security authority written by a dedicated

developer, have it reviewed and tested thoroughly and have the rest of the system rely on this component.

When is then the programmatic approach better? In a nutshell, when there is no alternative. In some cases, the declarative approach will not be granular or mature enough but there will be methods that will allow the desired level of granularity. Or, perhaps even more frequent, the access control logic depends on business rules or other dynamic conditions that simply cannot be expressed declaratively.

## 4.17 Reviews Are Your Best Allies

In the previous section we mentioned reviews, but what kind of reviews would benefit a security design? The practice of software reviews is not new, although judging by today's quality of software they are not as widespread as they should be. [Wiegers1997] is a good text on how to properly conduct a software review.

### 4.17.1 Design Reviews

What to look for at a design review? A few suggestions:

- usage of proven (versus brand-new) technologies. For instance, if encryption is needed, given that there are proven algorithms that sustained extensive analysis, there's little reason to come up with a brand new encryption or algorithm.

- clarity & completeness. The system's behaviour should be defined so that there are no vague areas where users or implementers may take different paths leading to different states.

- look at "how" and "why" not only "what": just because a particular technology is considered secure, it does not mean the way it is used is proper or that an attacker cannot find alternative paths.

### 4.17.2 Code Reviews

A good security design is great to have, but unfortunately it is not what will be used. The design is only the blueprint of the real thing and security flaws can easily be created by improper implementations of good designs. Some issues to watch for:

- is the design used as intended? If pseudo-random numbers are to be generated, are the values really cryptographically random? Is it used properly?

- known mistakes (buffer overflows, hidden fields used inappropriately and the entire list of coding errors explained in various secure coding papers).

- does the secure code rely on insecure libraries or subsystems?

- does the code give a "clean" and professional impression (well structured, has error handling etc)? If not, pay even more attention. While good code is not necessarily secure code, bad code will likely lead to security lapses.

# 5 Security Services

## 5.1 Authentication

Authentication is the process through which one party proves the identity it asserts to have. We are all familiar with the most common form, username/password.

Choosing an appropriate authentication mechanism is not trivial. A real-life project would have conflicting requirements:

- *strength*: an authentication scheme that is very weak will serve little of its purpose.
- *usability*: after all, if authentication is intended for human users, they should be able to use it (and do it without writing the password on a piece of paper)
- *manageability*: a good percentage of support calls are for password resetting. Help desk centres are well aware of this and this is why the automated password reset (with the secret question and answer) is so common. However, other authentication methods may involve more work. Token authentication is much more expensive in terms of issuing new credentials (i.e. programming a different token).
- *scalability*: a large system should scale well. This includes the ability of having a distributed security authority.
- *capabilities*: in distributed environments, authentication schemes are often required to support delegation and impersonation. Only some mechanisms support such features.

### 5.1.1 Context Propagation, Delegation and Impersonation

These three terms are often found when authentication schemes are discussed.

**Context propagation** is the ability of a multi-layered system to propagate the identity of the initial caller to the other layers. Note that this does not imply *proving* such identity beyond the initial check. Rather, propagation works based on trust: once the caller has been authenticated by the first layer, the others trust that layer to propagate the correct identity. J2EE is a typical example, in which the web tier authenticates the HTTP client and propagates its identity to the EJB container. Context propagation also occurs when calls go across processes or threads.

**Delegation** is the ability of a multi-layered system to propagate a caller's credentials through its layers. Unlike simple context propagation, delegation also transfers the credentials, so the identity of the caller is more strongly determined by each layer (which can perform authentication, if needed).

**Impersonation** is a subsystem's ability to [temporarily] assume another identity and perform work under that identity.

Impersonation relies on either simple context propagation or on delegation to work: the latter two convey the identity to the subsystem whereas the former performs the actual

work. A system that has delegation but not impersonation can still work but is rather limited: the work is still done under fixed credentials (because the code cannot assume another identity) but the identity of the caller is propagated. This can serve for auditing (the system just records who made the call) but again, the possibilities are not many.

### 5.1.2 Single Sign-On

SSO (as it is often referred to for short) is the ability of a user of multiple systems to only log on once and then be authenticated to all other systems as well. This has become a hot issue in enterprise environments where literally dozens of different systems have been developed in time, each of them with its own authentication domain. Apart from the management nightmare, a secure integration of these systems (as required by the evolving business) is hindered by the separation of the security mechanisms. SSO assumes a central authentication authority and support from the participating systems.

Kerberos and PKI have been used for implementing SSO. A more recent approach, actively promoted by Microsoft, is MS Passport. Several years ago, an Open Group effort was geared towards specifying an API for SSO, details at http://www.opengroup.org/onlinepubs/008329799/

## 5.2 Authorization

### 5.2.1 DAC – Discretionary Access Control

The DAC mechanism will be most familiar to readers as it is implemented by the major operating systems. In this model, each resource has an *owner* who decides who else has access to that resource and what operations are permitted. The list of permissions is usually called an *Access Control List* (ACL), nothing more than a simple list of *Access Control Entries* (ACE). As mentioned, a typical ACE associated with a given resource would look like (principal, operation). Depending on the system, principals can be users, groups or roles. Operations can be positively or negatively qualified: "user X is allowed to read file F" or "group Guests is prohibited from writing to directory upload".

The owner is often the creator of the object but this is not carved in stone. An owner can choose to relinquish ownership to another principal (but cannot take it back, as he is no longer the owner).

When a principal requests access to a resource, the authority in place (the security subsystem in an OS, the Security Manager in Java etc) will check the principal's identity against the ACL and decide whether to allow the access or not.

It is important to note that the system authority will need additional information apart from the principal and the ACL. For anything but small systems, maintaining an ACL for every single resource would be an unacceptable overhead. Fortunately, complex systems will have a hierarchy of resources (eg directories in an file systems, the Java class hierarchy) so **inheritance rules** can be defined, propagating an ACL down a hierarchy. Thus an object's ACL will potentially be composed from inherited ACEs and explicit ACEs. What happens if these ACEs are in conflict? Or what if explicit ACEs contradict themselves? This is where one more element comes into the picture: the system's **security policy** which establishes behaviour rules such as "explicit ACEs override inherited ACEs", "user-level ACEs override group-level ACEs" or "negative permissions override positive permissions".

The DAC model has been pretty popular because it is easy to understand (thus easy to apply correctly), often mimics an existing hierarchy of rights and delegates administration of security rights to the object's owners. The disadvantages of DAC become apparent in large systems: administrating the ACLs is a significant effort, prone to mistakes. Speaking of errors, *detecting* an incorrectly-assigned permission is not trivial in complex systems. Whereas a too restrictive ACL would be quickly discovered (the user would complain of denied access), an ACL too permissive is much harder to spot. Few users would ever complain of accessing too much and in certain environments knowing more is definitely advantageous. Permissions are usually given as a result of a business rule and checking an ACL's correctness requires verifying the entries against the original request. Most organizations will consider this too un-productive to enforce it, especially if the resources at stake are not sensitive enough to warrant a constant check. There is a legal touch as well: as dictated by the risk management evaluation, the leak of information can be reduced by enforcing penalties for breaches. Even if a user would get unauthorized access to certain information, exploiting this knowledge for profit (such as in the insider trading restrictions) or to create damage to the organization (by disseminating it) would lead to a legal suit.

What does the above mean for the developer? Two cases will be more common:

- building an application within a given DAC environment, such as writing a backup tool or a Java application that makes use of the security policy. In this case the inheritance rules are defined by the execution environment (the OS or the JVM). The security policy can be partially changeable and this highly depends on the specific environment. For instance, under Windows the administrator can specify how to solve the conflict that occurs when a user is allowed access to a file but denied to an upper level directory. This is the Directory Traversal privilege (a POSIX attribute, actually). But other bevahiours, such as the precedence of negative over positive permissions, are invariant.

  Thus, what developers are left to do is to fully understand all the system behaviour and to design the scheme of assigning principals and resources so the system can enforce the desired behaviour. If a nested groups are desired, then clearly the native pre-Active Directory Windows mechanism is not enough because groups cannot become part of another group. This limitation is not applicable to systems where authentication can be done against an LDAP directory (such as Windows 2000 with Active Directory).

- building a custom DAC mechanism for application-specific resources. A database-based payroll application where both the application resources (whatever financial records) and principals exist in a database only must enforce the access control by itself. In this case, the system must define all the elements in a DAC: scheme of principal hierarchy, inheritance rules, policy and the enforcing authority itself. In the previous example we could rely on the underlying execution environment for this task, but not any more. This is clearly a more complex case and repeated design reviews should be performed to identify any vagueness, conflicts or loopholes.

### 5.2.2 MAC – Mandatory Access Control

The MAC model will be somewhat familiar if you think of it in the context of intelligence agencies and classified information. In the MAC's view of access control, resources are assigned *sensitivity levels* (such as public, restricted, secret, top secret etc) and principals have *access levels* (or *clearance levels* in an alternative terminology). The levels are ordered so a hierarchy can be defined.

Access control is defined through a policy. The policy describes both the access control rules themselves and how a resource's sensitivity level is affected by operations. A typical example would be as follows:

- a principal can read all resources with a sensitivity level equal to or lower than the principal's clearance level

- a principal can write to resources with a sensitivity level higher than the principal's clearance level but cannot read those resources

- a principal cannot increase its clearance level upwards

Note that there is no ACL per resource.

This model has been formalized in a classic paper, [Bell1975].

The MAC model has advantages in larger systems. Let's take the example of an intelligence agency again: managing an ACL for each document (as in a DAC model) would be a daunting task that would quickly reduce the agency to a paper-shuffling organization. The MAC model makes the access control simple: does the user have the required clearance level for a particular document?

Note: the above description actually is a simplification of an intelligence agency's security model. To restrict further information dissemination, such an organization would enforce additional policies, such as the *availability-on-a-need-to-know* basis, which complicates the behaviour: how and to whom should the need-to-know conveyed so the matter at issue remains sensitive. Also, the real world adds its own complexity: once a Top Secret document leaks to the press (an external violation of the system's rules), how is the sensitivity level of related topics affected?

There is another example of MAC which is closer to the Internet user: content rating of web pages. Access to content is based on the level of the user (adult or minor) and the label the content carries.

If MAC authorization model is contemplated for an application, the first step would be to formalize the model. The next task is to define the architecture supporting it. Common operating systems are not built around a DAC model and matching the MAC paradigm is not exactly trivial. There has been work on developing MAC-based systems from ground up, but their application is not main stream. If you need such a system (and have an email address ending in .mil or .gov :-), you already know where to look

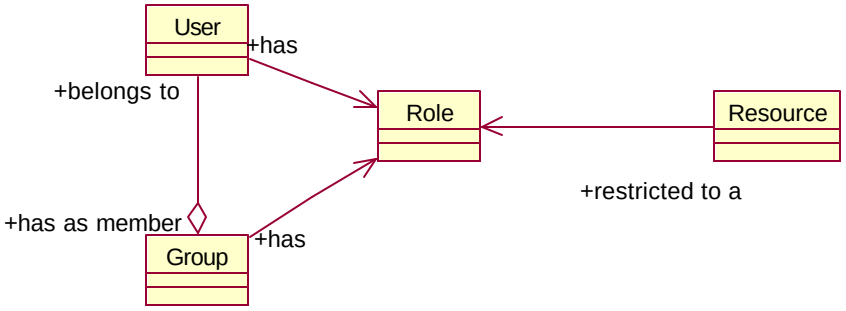
### 5.2.3 RBAC – Role-Based Access Control

The DAC model defines permissions by associating resources to users and groups (of users). It is easy to understand and we are all familiar with the concept of groups at the very least from the operating systems we use. However, there is something that is missing in this picture, something that appears in the real model yet the DAC framework does not model. Groups express *organizational affiliation*, be it hierarchical or geographical. They express less clearly notions of *competency, authority or responsibility*. Or, to be more precise, groups can in fact do the latter, but conflicts arise when groups are used to express both domains indicated above. Modeling the relationships in the real world requires both. The typical example is a

military base which has an Office On Duty every day. Officers are rotated to become O.O.D. While being an O.O.D., an officer has additional rights and responsibilities, as this *role* confers. However, that officer still belongs to the same organizational unit in the military base (platoon P, battalion B, regiment R and so on). Whenever something applies to his platoon, the officer is still considered in. However, access to the system that puts the entire base on high alert is not given to a particular individual in an particular organization unit, but to whoever has the Officer On Duty role.

In security models that do not distinguish groups from roles, groups can take the semantics of a role. When groups are intended to mean "who can do this" rather than "how should these users be collectively referred to", the real meaning is of a role.

The following UML diagram synthesizes the concept: resources are restricted to certain roles. Users can either have a role directly or by belonging to groups which have those roles.




Role-Based Access Control (RBAC for short) formalizes this approach and the main repository for information on RBAC is at http://csrc.nist.gov/rbac/. If you are interested in the formal approach, [Cugini1999] would be a good start. If interested more about the relationship between roles and ACLs, see [Barkley1997].

There are four RBAC models, as follows:

- **RBAC$_0$**: the basic model with users associated with roles and roles associated with permissions (depicted in the UML diagram above)

- **RBAC$_1$**: RBAC$_0$ with role hierarchies.

- **RBAC$_2$**: RBAC$_0$ + constraints on user/role, role/role, and/or role/permission associations.

- **RBAC$_3$**: RBAC$_1$+RBAC$_2$

RBAC$_1$ allows good flexibility in modeling the real life scenarios and significantly decreases the administration work. A role hierarchy allows implicit inheritance of capabilities (eg, a TeamLead role will include all the capabilities of a TeamMember without having to explicitly assigning the same rights of a TeamMember to the TeamLead role). If you plan to use RBAC, then consider whether going directly for RBAC$_1$ support is not better than starting just with RBAC$_0$ only.

What does the RBAC require to be implemented? Fairly obviously, the system must support the roles (and, in case of $RBAC_1$ and $RBAC_3$, role hierarchies). This support must come in the authentication phase, management (roles will have to be managed) and user interface. The permissions are now specified using roles. Especially if an existing project is to be converted to an RBAC approach, carefully consider the effort needed to do the conversion and (if the system is in use) to train the administrators to follow the new style.

[Beznosov2000] covers the usage of RBAC in a health-care system.

Several technologies that use the concept of a role are J2EE (the security descriptor for the EJB uses roles) and COM+. The Java Authentication and Authorization Service framework (JAAS) also makes it easy to implement roles: Subjects are essentially collection of Principals that are populated at authentication time. This allows a particular Subject that initially only had a username and a password to be populated with other principals representing the roles.

## 5.3 Confidentiality

### 5.3.1 Cryptography

This will not a section on different algorithms, not even on the difference between symmetric and asymmetric (PKI) cryptography. There are excellent resources for this, such as see [Menezes1996] (also available free online!) and how-can-we-not-mention-it, [Schneier1995]. [Singh2000] also is a fascinating exploration of the history of cryptography.

Perhaps the single most important practice is to *leave crypto design and implementation* (when possible) *to the experts*. This does not mean avoiding cryptography but avoiding designing it. It takes a significant amount of expertise to do it right and apart from the people who do it for their daily living, the rest of us simply do not have that expertise.

There are many crypto toolkits on the market, either from major PKI vendors such as RSA, Entrust, Baltimore or as open-source projects (such as Cryptix for Java, available at http://www.cryptix.org or Crypto++ for C++ available at http://www.eskimo.com/~weidai/cryptlib.html). It is also much easier to use such products than writing one's own implementation.

#### *5.3.1.1* The Value of Chaos: Good Randomness

Random values appear often in security. They can be:

- *encryption keys*. In order to avoid reuse of the same key, encryption keys are often established on a per-session basis. Of course, encryption keys should not be guessable thus the need for randomness.

- *session identificators*: stateless protocols like HTTP maintain state through a session id. After the login phase, possession of a valid session id is what identifies the user. These session id values should be random in order to prevent an attacker guess valid session ids and thus hijack other users' sessions.

- *challenge-response* authentication *protocols*, in which the authenticator sends a random value to the client.

Writing a random number generator is not trivial. The first hurdle is that anything generated algorithmically is not random in nature, it can only appear random to the human eye. This is why such algorithmic solutions are called *pseudo-random number generators (PRNG)*. Most modern languages have a built-in PRNG but this is usually limited to generating the same sequence of values. The typical implementation uses a linear congruential generator which, while producing a good distribution of the values (good enough if you simulate dice throws), is totally predictable. An attacker who manages to figure out where the generator is in the sequence would score a significant victory against the system.

Usual criteria for non-cryptographic PRNGs require the PRNG accept a seed and have a flat distribution of probability and no correlation between any two values. Crypto-strong PRNGs add additional constraints:

- the seed must be randomly selected and have at least as many bits as the generated values

- forward unpredictability: it must be impossible to compute the next value even if the algorithm and all previous values are known (note that the seed is still unknown)

- backward unpredictability: knowing generated values should not allow determining the seed

The reader interested more in the theory behind PRNGs should consult [Menezes1996] – Chapter 5, [Hall1999] which rightfully notes that PRNGs are often single points of failure for security implementations and [NIST2000] which will be mentioned shortly.

How to properly use randomness in secure application:

- Use random values where necessary. Unfortunately, many systems today use non-random values, not even the predictable linear congruential generator. Every once in a while reports come out about systems that use an extremely poor algorithm, even assigning session identificators sequentially.

- use a cryptographically-strong PRNG instead of the generators supplied with most language runtime engines.

- for specialized applications where real randomness is required, natural phenomena are often used to generate random data. For instance, http://www.random.org provides data sequences generated from where atmospheric noise. Other methods use air turbulence in hard drives, radioactive decay etc.

- verify undocumented PRNGs and consider replacing them whenever possible. Many servers that use session identificators have some sort of PRNGs but rarely are they documented in terms of design and cryptographic strength.

We have just said that we should verify unknown PRNGs. Unfortunately, a theoretical proof of a generator's randomness is impossible ([Menezes1996]) so we are left with an engineering approach: we have a generator, we try to break it in various ways and if it withstands the attacks, we declare it good for all practical purposes. At least until someone finds a better way.

We will not delve here in the details of testing for randomness. [NIST2000] is an excellent resource on this topic: a 160-page paper on testing PRNGs, a battery of 16 statistical tests (source included) and other papers on this topic. Another page with information on PRNG is David Wagner's at http://www.cs.berkeley.edu/~daw/rnd/. And for an interesting paper with a visual touch, see [Zalewski2001].

#### 5.3.1.2 Snake Oil

The Real World is not necessarily fair and trustworthy and this applies to security software as well. Once in a while, you will find products with larger-than-life claims. "Revolutionary breakthroughs", the cure to all your security concerns, the secret software that gives you 100% security forever without even requiring you to do or understand anything etc. You've got the idea. Why they are bad and how to spot such cases is the subject of a dedicated FAQ: [Curtin1998] and of numerous commentaries in Bruce Schneier's *Cryptogram* (http://www.counterpane.com/crypto-gram.html)

### 5.3.2 Steganography

Unlike cryptography which hides the meaning of data, steganography hides (or attempts to) the presence of sensitive data. It is not a physical hiding of the transport mechanism (the CD-ROM hidden in the double wall of a case), but obscuring that an existing data medium contains *another* message. Steganos in Greek means "covered writing". Many techniques have been developed, initially exploiting human limitations in perception: hiding data in the contents or layout of images, sounds or text. Note that by itself, steganography is not very secure: once the hiding method is discovered, the hidden message is revealed. Also, steganographic communication is potentially vulnerable to traffic analysis (it is suspect if two parties suddenly exchange many pictures and MP3s) and there are mathematical attacks designed to identify whether a communication contains hidden messages (see [Honeyman2001], [Provos2001]). Thus encryption becomes necessary and, except for specialized applications where the covert communication is needed (intelligence agencies or watermarking of digital art works), most readers will find encryption suitable for their needs.

## 5.4 Integrity

Integrity plays an important role in security, as in the real world. When we sign a contract, we want to be sure the document cannot be forged later. Since the human signature is not message-dependent and the paper copies can potentially be modified, the paper world has adopted the multiple-copy approach in which each signer has an original document, signed by him and all other signers. Changing one document, while possible, is quickly identifiable.

This approach does not suite well the computer world. Because the cost of the attack is much smaller (it is far easier to change a file than to forge a paper copy), such attacks would be more frequent. Whereas the multiple-copy approach would still work for rare cases, implementing it for every unit of data would become prohibitive in terms of time, traffic and storage needed to verify the signature. The integrity verifying data must be attachable to a single copy of the data.

Data integrity is one of the typical applications of PKI, through its digital signature service. With a PKI in place, digitally-signed data is a feasible undertaking. This has been a success

story: such signatures are used today for email and document signing, code signing (like Microsoft Authenticode or Java jar signing) or inside communication protocols.

What if there is no PKI in place? The alternative is to use a Message Authentication Code (MAC), computed based on the message and on a secret shared between the signer and the verifier. A MAC algorithm based on a hash function is called an HMAC. How a hash function can securely be used for integrity checks is discussed in [Bellare1997]. HMAC implementations, most of the time using MD5 or SHA-1 as hashing algorithms, are available in many cryptographic toolkits or APIs.

## 5.5 Accountability

Depending on the level required, accountability can be implemented from *auditing* (in which the system maintains a log of relevant events) to *support for non-repudiation*, much more demanding in terms of complexity.

Which to choose is a first a business matter. Actually, to be more correct, a properly designed system will almost surely have auditing in place, so the decision that is left is whether non-repudiation is needed.

A key factor in having relevant auditing is to have reliable sources for data. There is little point in collecting information if the information can be forged. If used for billing, this can even backfire.

One more decision to make is whether logging is to be done locally or centrally. Central logging is more convenient for administration and event correlation, but more complex to implement, including security-wise. The protocol used to communicate the logging information must ensure authentication of the client and of the logging facility, confidentiality (if applicable) and integrity of the data. What happens if the logging server is unreachable is another concern. SecurityFocus maintains the log-analysis mailing list focused on this topic, you will probably find the discussions useful.

### 5.5.1 Log Usefulness

An auditing log is not useful by itself but through the information it contains. Some suggestions:

- consider what will be meaningful and relevant to those who will use it. A mere "Access denied" error doesn't tell much. An "Access denied for user … while attempting to perform action … the resource …." is much better.

- if you are not sure what information to put in the log, consider asking your system administrator for opinion.

- more detail is generally better

- too much detail is annoying and can hinder performance. Provide the ability to change the log level, for instance based on severity or source.

- provide the ability to store in and export the log to a format that is easily parsable without ambiguity. For instance, if the data in the log contains spaces, do not separate the fields with spaces. Use another character that will not be part of the regular data, such at tab.

- logging to a database is useful, but do not forget the plain old text file (and never forget about the Perl fans :-). In some instances (database server unavailable or a crisis) being able to quickly read the information with a text editor and to extract with a grep command is very useful.

### 5.5.2 Other uses of logs

It may be surprising, but many developers do not use the existing logging facilities for their own debugging. A security violation, if not reported as such in the user interface, will often be masked by another error. Precious time will be spent trying to hunt down the other error when a quick peek in the system's logs can reveal the security violation. This can happen more often in distributed systems. Also, please note that in some cases security auditing must be turned on in order for anything to be logged. This is the case of Windows NT and 2000 systems where unfortunately security auditing is off by default.

### 5.5.3 Support for Non-repudiation

First, a clarification on the "support for" clause. As correctly pointed out in [Adams1999], solving repudiation claims still requires human reasoning to weigh the evidence. Yes, PKI-based digital signatures provide *the support* for strong binding of a person to a transaction but still, the signature is still not absolutely linked to the person. Someone could have obtained unauthorized access to the machine, copied the private key, installed a keyboard logger to capture the pass key and thus become able to impersonate the legitimate user. If the user claims such an attack occurred, it is up to the forensics team and eventually to a legal authority to believe one side or another.

[Caelli2000] distinguishes no less than eight types of non-repudiation:

- Approval (of a message's content)
- Sending (of a message)
- Origin (combination of the two above)
- Submission (of the message to the delivery authority)
- Transport (of the message by the delivery authority to the recipient)
- Receipt (of a message by the recipient)
- Knowledge (of the content of message by its recipient)
- Delivery (combination of the two above)

As you see, the topic has nuances, some of them going quite far from the simplistic approach that "digital signatures provide non-repudiation". For instance, whereas non-repudiation of sending can be obtained by the classic signature with a private key, non-repudiation of approval requires more thought. If the signature operation can be automated and uses repeatable input (such as the passkey required to access the private key), then it is conceivable that malicious software (which repeats the keystrokes and mouse clicks) can repeat the signature on unapproved content. When this is a concern, the security

mechanism must ensure that the signature can only be created if the content has been reviewed by the human user.

If non-repudiation support is required, then find out first what type(s). Here conferring with a legal counsel is the proper way to start with.

## 5.6 Distributed Security Services

Enterprise systems are most often distributed, platform-, infrastructure- and technology-wise. Maintaining security across such a varied landscape is not trivial and this challenge led to several solutions:

### 5.6.1 Kerberos

Kerberos is an authentication mechanism based on symmetric-key encryption. Documented in [Kohn1993], it is increasingly popular, one of the reason being scalability support for delegation. It is conceivable that Microsoft introduced Kerberos in Windows 2000 because of the lack of delegation support in the NTLM authentication mechanism used by previous versions.

There are plenty of resources on how Kerberos works so we will not describe the protocol here. What is more relevant to most developers is how to make use of Kerberos.

Perhaps the most important aspect is that using Kerberos (much like using a PKI) is first an infrastructure issue. Kerberos requires a software and service infrastructure that will not be built simply to serve one particular application when the rest of the systems use different mechanisms. Rather, the decision to implement Kerberos goes across multiple systems.

If a Kerberos infrastructure already exists, applications could be re-designed to make use of it. See [Coffman2001] for a possible approach.

### 5.6.2 SAML

In the list of major security services, Kerberos misses one: authorization. This is not a shortcoming, it is simply the case that the protocol has not been designed for authorization. Today's distributed environment have brought the necessity for having authorization information distributed across multiple systems and multiple platforms. As a representation language of choice, XML was easily selected, but what security information to communicate and how to structure it?

After a period of independent work on competing proposals (S2ML from Netegrity and others and AuthML from Securant), the work is now done under the auspices of OASIS (the Organization for the Advancement of Structured Information Standards) which overlooks other XML-based standards. The emerging standards named SAML *(Security Assertion Markup Language)* combines the previous work on S2ML, AuthML and X-TASS. You can read the documents at http://www.oasis-open.org/committees/security/ but this is still work in progress so there will be a while until it is mainstream development.

As very recent developments, Netegrity announced the availability of a Java-based SAML and if you are interested, you can find out more at their web site, http://www.netegrity.com

# 6 Technologies

## 6.1 Platforms

### 6.1.1 Mobile

Mobile platforms (handhelds, telecommunications devices) are increasingly becoming part of business systems. For the security architect, they present special challenges:

- theft risk: the devices are prone to theft, so the security mechanisms should be able to deal timely with revocation of access rights and managing the impact of information loss (assume the adversary has obtained access to the information on the mobile device). Also, because devices are easy to steal and even put back, never rely on the device to authenticate its presumed user. When the device is used for live connections to a system, consider prompting the user for data before establishing the connection instead of storing the credentials on the device.

- lack of security sophistication: mobile devices have primarily been design with convenience in mind. Protection of information is often naïve (see [Kingpin2001]) or backdoors exist.

- lack of extensive security analyses. Not much has been published on the mobile device security. Kingpin's analysis of PalmOS is notable ([Kingpin2001]). For Windows CE is covered by Microsoft in [Alforque2000], whereas RIM's Blackberry is covered in the official documentation: [RIM2001a], [RIM2001b].

- lack of computing power. Certain resource-intensive operations, such as digital signatures, are not appropriate for all mobile platforms because they would take an unacceptable time. Thus alternative, "light" mechanisms must be found.

- wireless transmission: some of these devices are wireless-enabled. Wireless communication, prone to varying signal strength, interruptions, roaming delays etc impose certain constraints on the protocols and mechanisms used. For instance, WTLS is a better suited mechanism than SSL, but such a replacement is far from being transparent to the system.

### 6.1.2 Desktop

The desktop environment should be treated as essentially out of control. Even in an enterprise setting we cannot rely the computer's configuration and behaviour is really the one intended. Either through user actions or software failure, even security mechanisms can be altered. Whatever security technologies are used throughout the system, the desktop computers are not relied-upon components.

### 6.1.3 Server

Servers live in a more "disciplined" world. Their configuration is more conservative and much less dynamic than a user's desktop. It is not unusual to have a server running older versions of a platform because they have matured and stabilized, as opposed to the latest releases. The emphasis is on running the business, not running the latest and greatest.

Some characteristics to be aware of:

- administrative control: servers are not typically administered by those who develop the applications running on them so be careful about relying on system settings. If your application requires certain system-wide settings (such as 128-bit encryption support or certain run time libraries), make sure such requirements are captured in the project's documentation so they will be known to system administrators.

- policies: always make sure the project's requirements include the relevant sections of the security policies enforced in the production environment. They can enforce restrictions on how the software should be built. If they are missed, the system may not be compliant with the policies.

- software versions: whereas developers prefer to work with up-to-date tools, sometimes the servers can be behind. This means some of the security features available in the latest releases may not be exist on the production system.

## 6.2 Performance

Performance has security relevance from two standpoints: (a) building the system so it can provide the required performance for security services and (b) maintaining the performance when this is the very target of an attack.

### 6.2.1 Performance As a Deliverable

An underperfoming and optional security mechanism will be avoided by users, an underperforming but mandatory security mechanism will likely lead to rejection of the entire system. Whereas security takes its toll in terms of resource usage, making security an excuse for poor performance is not acceptable.

Performance implies both speed and scalability (ability to grow with the demand). Most of the effort required to reach these goals is at a design stage and addresses the entire architecture. However, there are security-specific operations that have a significant toll on performance and, as expected, these are the cryptographic operations. The complexity of the algorithms used is unavoidably reflected into the load imposed on the processor (usually it is the CPU load and not the memory or I/O traffic that skyrocket when crypto operations are running).

PKI operations are a typical case and a design-level optimization has been chosen for digital signatures: instead of signing an entire message, the message is hashed and only the hash is signed. This would not have worked without the strong binding between a message and its digest so there is a proven and trusted dependency chain here that allows this "shortcut" to be secure. Such design-level optimizations have improved speed, but scale remains an issue for a busy e-commerce site.

Hardware-based accelerators come in where off-the-shelf computing architectures stop. Crypto-accelerators are separate devices with a processor optimized for crypto operations that offload these tasks from the server's CPU. In their most typical usage, they will work in conjunction with a web server to handle the SSL traffic.

Crypto accelerators can come as internal boards or external devices. The former provide real end-to-end encryption but require support in the OS and in the web server which must know to delegate the SSL functions to the new device. If internal accelerators are considered, make sure they are supported by the web/application server. The latter category will be deployed in front of the web server which thus will receive unencrypted traffic. This will work with any server, but encryption will be client-to-accelerator not client-to-web-server, leaving a risk the attack can focus on the last segment between the accelerator and the server. The other impact a developer should be aware is that the web server will no longer know whether the connection was established over a secure channel so application-level checks of this condition will fail.

The section on SSL contains additional material on this topic.

### 6.2.2 Performance As an Attack Target

As we mentioned earlier in the document when we discussed assets, the availability of a service is an asset in itself. One way to attack the availability is by overloading the entities involved in delivering the service request to the provider and the response back to the client. Such entities can be servers (especially web servers as they are a system's interface to the world) or network devices (such as routers) through which the traffic goes. Such attacks are classified as *denials of service (DoS)* and have been quite popular in 2000 and 2001.

DoS attacks cannot be dealt with from within the application or the system that is the target (because the application is already overloaded and the additional CPU resources required to handle the attack are not available). The solution is to prevent the malicious traffic reaching the system and this is a matter of infrastructure design, monitoring and incident response. You must be able to detect them as early as possible (instead of finding out from customer calls), know how to react (notify the upstream ISP, for instance) and have an infrastructure capable of implementing the response (such as multiple ISPs, additional routers or proxies that can be used to filter the attack traffic before it reaches the server etc).

The details go outside of the scope of this document, please refer to recommended practices for network security design.

## 6.3 Languages and Security

Support for multiple languages through extended character sets (such as Unicode and UTF-8) has introduced security vulnerabilities. It is perhaps surprising, yet true. Readers who have worked with intrusion detection systems may already be familiar with one issue, that the same logical string can have multiple representations. In the IDS world, this increases the difficulty of pattern scanning. In the applications world, multiple representations of the same data means it is harder to validate the data. We have already covered how important it is to validate what a user provides to the system. [Wheeler2001] has a very good coverage of the security implications. References to security implications in the Unicode specification can also be found at http://www.unicode.org/unicode/reports/tr27/

# 7 Languages

## 7.1 C/C++

The biggest security problem with C and C++ is also the most frequent application-level attack: C's inability to detect and prevent improper memory allocation, with the direct result of allowing **buffer overflows**. A great deal of material has been written on this topic but it is still as valid as 20 years ago. The main reason is that prevention of buffer overflows, not being done by the language itself, is left to the programmer to implement. Which means that in the real world this is rarely done. Of course, until the day someone finds out and the vendor scrambles to fix it while the users hope the fix will come before an attacker uses it.

We will not describe the mechanism of buffer overflows here, many a paper have been written on this subject. The reader can consult [DilDog1998] and [Aleph1997].

A more recently found problem is the **format string attack** in which a combination of sloppy programming and lack of input data validation leads to the same fatal result. Format string attacks have been publicized since 1999. Read more in the introductory [Newsham2000] or the excellent coverage in [TESO2001]. [Thuemmel2001] also covers this topic.

What approaches do we have today for handling buffer and format string vulnerabilities?

- the best way is to use good coding practices: check for boundary limits, use the proper way to call a function. This requires a great deal of discipline from the programmer and in practice even the most experienced developers can overlook such checks once in a while. Source code analysis tools such as ITS4, Flawfinder or RATS (see the Tools section) help in scanning the code for known patterns but they must be used. If a project ignores the code review phase, then the tools do not reach their goal. Integrating such checks into the syntax check of the code would certainly increase the frequency of the checks.

- another approach is to automatically insert code for run-time checks, either of whether the memory allocation is fine or of whether the stack has changed after a function call. The former means augmenting C/C++ with the run-time bounds checking that other languages have, the latter means using a mechanism to detect abnormal changes in certain memory structures.

- the third approach is to make the execution environment itself more resistant to the error conditions underlying the attack. This requires replacement of system libraries.

[Simon2001] and [Wheeler2001] discuss in more detail the last two approaches and a number of tools. [Mixter2001] also contains some guidelines on what to check for in a C program. [Foster2001] offers another approach for format strings.

## 7.2 Java

One of the reasons Java is so popular is the intrinsic security mechanisms. Assuming the virtual machine is implemented correctly, malicious language constructs are not possible in Java. For instance, buffer overflows, use of initialized variables, invalid opcodes and other vulnerabilities that plague other languages/platforms are stopped by the JVM (again, assuming a JVM that works *correctly*. Like with any piece of software, there have been bugs in the JVM implementations allowing exploits that got around the mechanisms).

We will not discuss here the Java security model, there are plenty of excellent resources on this topic: [Gong1999], [Oaks2001] and [Jaworski2000]. [Gong1999] is an interesting read for the security architect for one more reason than the obvious: the author, no other than the architect behind the Java security model, retraces how various design decisions have been made. How would you design such a model if you had the chance of design a programming language from scratch?

JDK 1.4 brings in the long-waited user-based security. Already available for 1.3 as the *Java Authentication and Authorization Service (JAAS)* extension, the classes under `javax.security.auth` complement the traditional Java approach of relying on codesource and identity of the code signer to make access control decisions for code execution. The JAAS approach also provides a beneficial decoupling between the security mechanism and the code calling it.

JDK 1.4 has brought in a number of changes to the security features, see http://java.sun.com/j2se/1.4/docs/guide/security/ for the list of changes. Apart from the integration of JCE, JSSE and JAAS into the main SDK, the addition of a Java implemention of GSS-API supporting Kerberos and of the Certification Path API (for certification path construction and validation) are good news.

Having JCE, JSSE, JAAS and GSS-API may seem confusing so the following documents may prove useful: *Introduction to JAAS and Java GSS-API Tutorials* at http://java.sun.com/j2se/1.4/docs/guide/security/jgss/tutorials/index.html and *When to use Java GSS-API vs. JSSE* at http://java.sun.com/j2se/1.4/docs/guide/security/jgss/tutorials/JGSSvsJSSE.html

In terms of recommended practices, [Wheeler2001] and [Sun2000] contain good collections of secure coding guidelines for Java programs.

As a quick tip, if you are debugging security-related problems and want to go beyond just the exceptions, try running the application with the `-Djava.security.debug` flag. You will be able to see the results of `CheckPermission` calls, loading and granting of policies, a dump of all relevant domains and other info.

## 7.3 Perl & CGI

CGI is a technology, not a language, but in the web world, Perl is often used for CGI scripts (the administration uses are not covered by this document) so we have merged these two topics. Good resources are: [Stein2001], [Birznieks1998] and the `perlsec` doc pages found in your distribution or at http://www.perl.com/CPAN-local/doc/manual/html/pod/perlsec.html

# 8 Frameworks

## 8.1 GSS-API

In the early applications, the implementation of security services was an integral part of the application. Logins were prompted for and authenticated inside the application, along with the implementations of encryption mechanisms. This, however, was error-prone (just try to make your own implementation of DES for instance and you'll see it takes a while to get it right) and highly unflexible. Later, the security code was moved into separate libraries, which was better, but the main application still had to link to and call specific functions from specific libraries. Moreover, if the security-specific code had to run on a different machine than main application, managing the trusted communication between the two became a problem in itself.

Enter the Generic Security Service Application Program Interface. What GSS-API brought was a separation between the client application and the provider of security services. Only the latter is really aware of how those services are implemented, the client is merely selecting those services and those configurations that are suitable. Out of the plethora of security services, GSS-API provides authentication, confidentiality and integrity.

The typical usage of GSS-API is network software, such as network authentication protocols. The network login process is a sensitive operation and, in order to be trusted and ultimately meaningful, it must take place in a secure fashion.

### 8.1.1 GSS-API Implementations

The GSS-API model of separation the interface to security services from the provider and mechanisms of those services has been implemented on various platforms.

#### 8.1.1.1 Windows SSPI

In the Windows operating system, the public interface is named the *Security Support Provider Interface* whereas the mechanisms are implemented by *Security Support Providers* (by the way, see [Brown2000a] and [Brown2000d] for a detailed view).

#### 8.1.1.2 Implementations á la GSS-API

Architectures similar to GSS-API have been used for other security mechanisms. In Windows, this is the case of the CAPI-CSP pair. CAPI (Crypto API) is the interface to cryptographic functions whereas CSPs (Cryptographic Service Providers) provide the implementations. In the Java world, the Java Cryptography Architecture (JCA) defines the Java Cryptography Extension as the abstract classes that will be implemented by Cryptographic Service Providers.

#### 8.1.1.3 Kerberos v5 GSS-API

[Linn1996] defines how GSS-API should be implemented on top of a Kerberos security infrastructure.

#### 8.1.1.4 GSS-API in C

The C bindings for GSS-API are defined in [Wray2000].

#### 8.1.1.5 GSS-API in Java

New in JDK 1.4 is a Java implemention of GSS-API, as defined in [Kabat2000]. In the section on Java we have pointed the reader to several useful Sun documents. Together with the support for Kerberos, this adds a reliable way to implement single sign-on.

#### 8.1.1.6 SPKM GSS-API

[Adams1996] complements [Linn1996] by specifying how a PKI mechanism can support the GSS-API.

### 8.1.2 Extensions of GSS-API

A useful framework evolves and in a field as dynamic as the IT, no framework can stay around for long unless it is expanded. GSS-API has led to a number of related efforts, below are a few:

#### 8.1.2.1 IDUP GSS-API

*Independent Data Unit Protection GSS-API* (defined in [Adams1998]) is an extension that is focused on protection independent units of information. These generic "units" are rather higher than lower level: can be email messages or files for instance, rather than network-level packets. These units will be treated separatedly by IDUP and this is the correct meaning of "unit independence". Of course, the units can be semantically or otherwise correlated, but the API would simply not care.

#### 8.1.2.2 GSI Extensions

This is a work in progress to add export and import of credentials, unrestricted and restricted delegation handling, as prompted by the requirements of computational grids. More information (including on what those grids are) and the current draft at http://www.gridforum.org/security/ggf3_2001-10/drafts/draft-ggf-gss-extensions-04.pdf

## 8.2 PAM

*Pluggable Authentication Modules* (PAM) is a framework for system authentication. Originally released with Sun Solaris, PAM is to user authentication what GSS-API is to application-level authentication. The PAM framework has decoupled the interface to system authentication services from the actual implementation of the mechanism. Moreover, apart from the pluggable architecture (allowing authentication modules to be plugged in and out as needed), PAM offers a stackable architecture as well. Authentication modules can be "stacked" in the sense that the user must go through several login mechanisms in order to be allowed in the system. A flexible authentication policy can specify whether the authentication modules are *required* , *sufficient* , or *optional*. More details in the various PAM documentations available.

JAAS (Java Authentication and Authorization Service), introduced as an extension to JDK 1.3 and now part of the JDK 1.4, is a Java implementation of the PAM framework. As an authentication mechanism JAAS has become very relevant to the EJB 2.0 specifications.

[Seifried2000] covers PAM for Unix and contains a number of useful links on PAM.

## 8.3 Common Data Security Architecture

Intel's CDSA is an open, interoperable, multi-platform software infrastructure for security services, geared towards C and C++ developers who want to use a common API for authentication, encryption, policy and management. CDSA proposes a layered architecture (which the reader will be familiar with, having gone through GSS-API and PAM) having cryptographic operations at the core of many services. Extensibility is another core value of the specification. The main site for CDSA is http://developer.intel.com/ial/security/.

Initiated by Intel in 1995, CDSA v2.0 was adopted by The Open Group at the beginning of 1998. Intel also initiated a reference implementation, later moved to open-source. It is now available for both Windows and different flavours of Unix from http://sourceforge.net/projects/cdsa

# 9 Distributed Systems

## 9.1 The Presentation Layer

In the old days, the presentation layer consisted of displaying characters sent over a serial line to a terminal and getting the keystrokes back to the server. No software was running on the terminal itself. Today's systems contain much more intelligence, either in fat clients (full-blown applications, popular in the client-server era) or thin clients (essentially web browsers).

Fat clients blur the line between presentation and business logic and for our purposes they are full applications. Much of this document is applicable to them.

Thin clients are not that thin, either. Definitely not in terms of MB occupied as anyone who downloaded a recent copy of Internet Explorer or Netscape knows. But size is not the only factor here: today's "browser" contains a sophisticated DHTML rendering engine, at least one scripting engine, a Java virtual machine, other application with which it is integrated (typically an email and a news client) and support for adding components or plug-ins to handle additional content types. The browser is also a dispatching mechanisms for other protocols and a launching mechanism for their associated applications.

A practical example of how many security holes can arise from the complexity of today's browsers is Georgi Guninski's site at http://www.guninski.com. Georgi has discovered numerous security vulnerabilities in Internet Explorer, Microsoft Office and Netscape, as well as other products.

[Stein2001] also a reference work.

### 9.1.1 Web Authentication

The pros and cons of the different authentication mechanism for WWW access have been described in several works. [Seifried2001] is one of them. [Feamster2001] is an excellent paper to read. Web authentication is closely related to session persistence, please see the section on this topic later in this document.

### 9.1.2 GET Considered Insecure for Sensitive Data

When sensitive data is to be passed to the server, do not send it as a parameter in the query string like in:

```
http://www.your-own-site-here.com/process_card.asp?cardnumber=1234567890123456
```

This is not appropriate because, like the parameters that are passed in the GET request are logged by the web server as well as in whatever proxies might be on the way. The above request will get logged in clear text similar to:

```
2000-03-22 00:26:40 - W3SVC1 GET /process_card.asp cardnumber=1234567890123456 200 0 623 360 570
80 HTTP/1.1 Mozilla/4.0+(compatible;+MSIE+5.01;+Windows+NT) - -
```

As you see, the credit card number was stored in the log file. Also, the entire URL may be stored by the browser in its history, potentially exposing the sensitive information to someone else using the same machine later. Or a cross-site scripting attack which may reveal the URL through the REFERER field.

SSL wouldn't help in this case, because it only protects the request in transit. Once arrived at the destination, the request will be decrypted and logged in clear text. An apparent rebuttal may come from the standpoint that the server must be trusted. Yes, it must but this trust implies that private customer data be dealt with sensibly. There is no reason for the server to store such sensitive information in clear text. In some credit card authorization schemes, the application server is but an intermediate and once the payment is authorized, the web application does not store the credit card number or at least not all digits.

The POST method uses the HTTP body to pass information and this is better in our case because the HTTP body is not normally logged. Note however, that by itself POST doesn't offer enough protection. The data's confidentiality and integrity are still at risk because the information is still sent in clear text (or quasi clear text as in Base64 encoding) so the use of encryption is a must for sensitive information.

### 9.1.3 Don't Store Sensitive Stuff in the *SP Page

(*SP stands for ASP or JSP)

Most of the time, this "sensitive stuff" would be username/passwords for accessing various resources (membership directories, database connection strings). Such credentials can be entered there manually or automatically put by various wizards offered by the development environment.

A legitimate question is why would this be a concern since the *SP is processed on the server and only its results sent to the client. For a number of reasons: from the security standpoint, the past has seen a number of holes in web servers that allowed the source of an *SP page to be displayed instead of being executed. For example, two [old and] very well-known IIS bugs caused the ASP being displayed by appending a dot or the string ::$DATA after the URL ending in asp (`http://<site>/anypage.asp.` or `http://<site>/anypage.asp::$DATA`). Last year, the "`Translate: f`" bug let to the same outcome.

Similarly, two bugs have affected earlier version of BEA Weblogic and IBM WebSphere: by typing the URL to a JSP page with the capitalized extension (JSP instead of jsp), the source code would be revealed. The details can be found at Foundstone's R&D/Advisories page, look for the FS-061200-2-BEA and FS-061200-3-IBM documents, respectively. (http://www.foundstone.com/)

We have referred to the above vulnerabilities simply to show how diverse the ways to reveal the source are. Sometimes it is much more trivial: by exploiting forgotten samples intended to display a file. If the attacker guesses the physical path to the desired file, the file will be displayed.

Another reason for not hardcoding credentials in the page itself relates to development practices. Such information should be stored in a centralized place preferably in a resource to which access can be audited.

### 9.1.4 Beware of Extensions

An often seen practice is to distinguish included files by using an .inc extension on the server side. However, this opens security risks when that extension is not registered to be processed by the server. Such a file, if known to an attacker who asks for it specifically instead of the including page, will be served back in all its glory, possibly revealing sensitive information.

### 9.1.5 HTML Comments

Comments are not bad per se, but some are: those embedded in the HTML or client script and which may contain private information (such as a connection string that was once part of the server side script, then commented out. In time, through inadvertent editing, it can reach the client script and thus be transmitted to the browser). Such comments can reveal useful information to an attacker. Note that the problems with comments is not relevant to traditional compiled programs because the compiler will not include the comments in the binary output.

### 9.1.6 Semantic Injection

We have coined this term to refer to the cases when the data passed from an attacker to the system within a particular context carries an additional semantics that becomes relevant at a later moment. Two major examples are cross-site scripting and SQL injection. In both cases, text entered by the user has an additional meaning for other layers in the system which execution instructions specified in text. In cross-site scripting, the data input by the user contains HTML and scripting code (so the target layer is the browser). In SQL injection, the data contains SQL injection, the target layer being the database.

Semantic injection is possible when the application does not distinguish between data and language. It is a very effective attack because it bypasses other protection mechanisms such as firewalls or intrusion detection systems. It works solely at an application data level. Let's have a closer look.

#### 9.1.6.1 Cross-site Scripting

Maybe the label does not seem menacing, but the security vulnerability is significant. It can potentially occur with every site that allows the user to input some data and later display it to other users. Typical examples are bulletin board messages, web-based email systems or auction sites.

Why is this a problem?

Because it breaches trust. When a user visits a site, it has a level of trust in the content that comes from the server. Usually, this means the user expects the web site will not perform malicious actions and it will not attempt to mislead the user to reveal personal information.

With sites that accept user-provided data, later used to build dynamic pages, an entire can of worms is opened. No longer is the web content authored by the web creators only, it also comes from what other (potentially anonymous) users have put in. The risk comes from the existence of a number of ways in which the user-input fields can be manipulated to include more than simple text, such as scripts or links to other sites. Taking the script example, the code would be executed on the client machine because it would undistinguishable from the

genuine code written by the site developers. Everything comes in the HTML stream to the browser.

Injecting script can be done through a number of ways: form fields, URLs or cookies. The danger is compounded if the script can make use of other vulnerabilities on the target platform. Client script by itself is a safe subset that will not be able to do direct damage on a computer. If this computer, however, runs Windows and if the same script can call installed COM components, then indirectly the script can perform whatever actions can be carried out by the component. As it already happened, if such a component is incorrectly marked "safe for scripting" then it can be called from a script. If the component can manipulate files, then the script can. Thus, what used to be a myth years ago ("if you open this email, the files will be deleted") can become true. As a matter of fact, a number of viruses have been written exploiting this chain of vulnerabilities.

Solutions to the cross-site scripting problem work by breaking the injection of the script. This can be done by either stripping the script altogether or by converting into a form that will not be interpreted as script.

The first approach means identifying that an injection is attempted by essentially searching for the special characters used as script delimiters. The primary example is < and >. If the input is not supposed to legitimately contain such characters, then it can be discarded, with appropriate logging. However, this is not as trivial as a string matching when alternative character sets and encoding schemes are used (see the section on International issues).

What if legitimate data can contain those special characters, though? The solution is to use what already is used in HTML to *represent* the special characters without using them. After all, web pages can contain these characters and even lines of code without them codes being executed. The process called *HTML encoding* which replaces special characters with escape sequences, such as **`<`** for < and **`>`** for >. They will be displayed as the angle brackets by the browser, but the escape sequences can be used as delimiters. URLs can be encoded as well but with sequences of the form %xy where xy is the hex ASCII code of the character.

A more detailed analysis of cross-scripting is found in [CERT2000b] and [CERT2000c]. Specific instructions from several web servers vendors are at [Microsoft2000] (and the other articles referenced by it), [Apache2000] and [Allaire2001].

### 9.1.6.2 SQL Injection

SQL injection is the process in which the attacker provides SQL statements (carefully terminated) as input to an application. For the attack to work, the application must naively construct the final SQL string for the database server by appending the input data to a static SQL statement. You can find a well explained example at http://www.sqlsecurity.com/faq-inj.asp. Although the example is for Microsoft's SQL Server, the issue is applicable to all database servers. As a matter of fact, it is not a database server issue because all the server can do is to execute the statement it received. The problem lies in the lack of [semantic] input validation. If what the user inputs is first validated, such constructs would be identified. The easy case is when the expected data should not contain single quotes or any other delimiter supported by the database server. Then a character set normalization (see the section on Multi-language Support for the reason) followed by a regular expression matching would flag the input as invalid. If valid

data can contain such characters (single quotes are common in Irish names, for instance), input validation is more complicated.

Also note that a reliable validation must be performed on the server side (like with any other validation). Doing it on the client side (in embedded Javascript, for instance) has no security value. The HTTP query can be constructed manually and sent via other programs.

### 9.1.7 Validate Everything Which Cannot Be Trusted

In the section on cross-site scripting, we have mentioned that validating the client input is critical to preventing the attack. This is not a web-specific need, though. Whenever the system is to process some data that has been provided by an untrusted entity, the data must be validated first.

But what to validate and what not? The short answer has already been given: everything that cannot be trusted. This is where the need to clearly define the trust boundaries becomes relevant. They are not identified just for the sake of drawing nice diagrams, but this information is actively used in the design and implementation of the system.

In almost all client-server interactions, the client is out of control and should be considered untrusted. Whether it's a browser anywhere on the Internet or a custom application deployed in an entreprise, the practical truth is that the server cannot know whether it is actually talking with the genuine application and, even if it is, that the genuine application is run by a "nice" user.

How to validate? Search for known attacks and block them? Actually not. The safer way is analogue to what firewall policies begin with: everything blocked and only what is positively needed will be allowed. Thus, a conservative validation will only allow what is positively needed and will discard (with proper auditing and warnings) the incorrect data. Scanning for known patterns means the effectiveness of the check is nullified if another attack is found.

#### 9.1.7.1 State Persistence In Web Applications

Because HTTP is a stateless protocol, maintaining state during a browsing session is a common requirement. This is done by having the browser storing and communicating the server a session identifier previously received from the server. The session id is stored in the URL, in a cookie or, less convenient, in a form field. The session id must be a non-guessable value in order to prevent session hijacking and we will discuss the issue in the section on randomness. Surprisingly, as revealed on a number of occasions, even web sites belonging to reputable companies can do a poor job of secure session management.

However, to tie this subsection back to validation, what can often be seen in real world is the bad practice of storing other data in the browser and relying on it to be passed untouched back to the server. Perhaps the most impacting for a site's operator is the real example of product prices stored in hidden fields. In January 2000, ISS Inc identified a number of commercial products that exhibited such a flawed implementation (see [ISS2000]) and which allowed attackers to enter orders with lowered prices. Apart from being a useless step (why would the server – which knows the price – tell it to the client only to learn it back?), such a practice is an example of where the lack of validation can lead to. And, since we are in the web world, [Stein1998] covers another potential mistake: relying on the Referer field.

And yet one more reason and then we will move on. An ingenious attack was reported by Jeff Jancula on the vuln-dev mailing list in August. Suppose a web application stores the session id in the URL. The attacker sends an email to the victim, passing a properly constructed URL to the site where the victim has authorized access and some incentive for the victim to click on it. This URL contains a session id which, however, does not exist on the server side. When the victim clicks on the link, the web server does not let the request be served because, correctly, the session id is not associated with a valid user. The victim will be prompted to authenticate. Everything is fine so far. What the vulnerability discoverers found out, however, is that some web server, instead of issuing a new session id to the user, will accept the id provided and will associate that id with the victim's account. Thus, the attacker can effectively hijack the session because he now knows a valid session id for a user currently connected. There is a time factor here (the attacker must know if the victim actually performed these steps) but some acknowledgement can be obtained by using read receipts for emails or other forms of communication where the lag is smaller (such as instant messaging). This attack is yet another form in which unvalidated input can lead to a security compromise. If you search the vuln-dev archives, Jeff's email is the one entitled "Web session tracking security prob" posted on Aug 30.

### 9.1.8 Activex Controls

[CERT2000a] is an excellent resource on the security of ActiveX controls: risks, features and vulnerabilities seen from the perspective of developers, administrators and users. [Grimes2001] is also a useful reference and chapter 11, *Malicious ActiveX Controls*, is freely available at http://www.oreilly.com/catalog/malmobcode/chapter/ch11.html

### 9.1.9 Error Messages

Server error messages can be revealing and thus disclose information otherwise well protected under normal conditions. What can an error reveal however? A number of things, such as:

- *physical paths*. If an included file is not found, the server may reply with an error stating "include file: *c:\inetpub\wwwroot\common.asp* not found". The physical path is not a danger by itself, but can reveal information that can be used further in other attacks or can simply give away information about the infrastructure, such as in the case when UNC paths are used.

- *platform architecture*. For instance, an ODBC error message may reveal the database server used. Or the message can contain the exact version of the OS or the CGI/scripting engine, thus helping a malicious party to tune an attack. For a skillful attacker, even indirect information is helpful: if a particular piece of software is older, it may indicate the server is not properly maintained and thus other vulnerabilities are likely to be found out as well. Having detailed information about a platform can also serve in social engineering attacks, especially in large organizations.

The solution is to carefully review the error-related configuration of the server as well as how errors are handled throughout the application.

It would also be better to work with the QA team which systematically goes through the web site anyway and who can enter the revealing error as bugs.

## 9.2 Middleware: OO & Protocols

Middleware security is a complex topic. The practical truth is that condensing recommended practices within the size of this document would rather be over-simplistic. Most often, middleware technologies are deployed in an entreprise setting and in such scenarios canned answers do not work well. We will prefer to refer the reader to more detailed resources on these topics.

### 9.2.1 COM/COM+/.NET

COM security is a topic big enough for a book and in fact there is one. It's written by *the* man to ask about COM security, Keith Brown from Developmentor. Be sure to check his page http://www.developmentor.com/kbrown/ and http://www.developmentor.com/securitybriefs/ for details on his book, *Programming Windows Security* [Brown2000a], an invaluable resource on this topic. At his web site you can also find other cool info and utilities to explore the COM world.

For web applications, the following resources provide a good description on how IIS and MTS/COM+ work together to impersonate a client: [Brown2000b], [Brown2000c] [Gregorini2000a], [Gregorini2000b] and the backgrounder in [Dulepet]. This last resource has useful tips on the difference between DCOMCNFG and OleView for setting component security.

For the emerging .NET framework, the documentation is still subject to change as the framework is defined. You may be interested in consulting [Kercher2001] and [Brown2001] cover authentication and code access security.

The following table summarizes what COM/COM+ offer as security services.

| Service | Comments |
|---|---|
| Authentication | based on the underlying MS-RPC mechanisms, thus being the same as those available on the Windows system. Credential delegation depends on the underlying mechanism (possible with Kerberos in Windows 2000, not possible with NTLM). Anonymous (non-authenticated) clients supported. |
| Authorization | discretionary for COM, plus role-based for MTS and COM+ applications |
| Confidentiality | available through the MS-RPC infrastructure |
| Integrity | available through the MS-RPC infrastructure |
| Non-repudiation support | not available |
| Administration | Microsoft-provided tools such as OleView, DCOMCFG, MMC snap-ins |

### 9.2.2 EJB

The Enterprise Java Beans is a subset of the J2EE specification, implemented by various vendors. The overall J2EE security architecture is described in [Shannon2001] and the EJB security in [Sun2001].We will focus on the EJB subset as EJB is the core of the J2EE architecture, more precisely on the recently-released EJB 2.0.

Sun recommends isolating the developers (or, more precisely, the Bean Providers) from the security functionality which is to be provided by the execution environment. We have already discussed elsewhere that this is a good approach, which much higher chances of getting a correct and consistent security behaviour. However, the official specifications leave a number of security questions unanswered and we will briefly address the more important of them.

The EJB specs do not define a class hierarchy for classes pertaining to EJB security. There are two methods in the `javax.ejb.EJBContext` interface, `isCallerInRole` and `getCallerPrincipal` but there is no standard EJB security exception class. This has an impact on the class design and portability: security exception handling will be vendor dependent. In one case of personal experience, to report the case when an EJB client successfully authenticated to the container but was not allowed to call a particular EJB, one prominent server vendor defined an internal error directly inherited from `java.rmi.RemoteException.` This was very inconvenient for structured error handling. An internal error does not even guarantee the class will not be renamed in the future and the direct inheritance from such as generic exception as RemoteException makes it hard to separate security exceptions from other types.

Delegation is important in the EJB architecture. A very common scenario is of a client accessing a system through a web tier implemented with JSP which, in turn, calls the EJBs. Another scenario is when an EJB running in one container calls EJBs running in another container. Delegation in EJB is achieved in EJB as delegation of identity: the identity of the initial caller is propagated to the other layers. However, it is only the identity and not the authentication credentials that are propagated. All but the first layer cannot *verify* the identity of the caller, they can only trust the first layer to be honest about the identity. Such risk is acceptable in environments where all layers are under the same authority, but it may be less easy in a business to business environment.

Compared to EJB 1.1, EJB 2.0 brings a new feature: impersonation. Instead of passing on the caller's identity to the subsequent layers, an EJB can present a different identity, as configured in the deployment descritor with the the `run-as` element). As with delegation, this impersonation is based on trust: the EJB does not have the credentials needed to authenticate that identity, the next layers must trust this bean to be truthful.

Access control is role-based in the EJB specifications. Mapping those roles to real principals is done in a vendor-specific fashion. Thus, the end product (i.e. the code, the descriptors and other configuration) becomes vendor-dependent. The risk is not of the small work needed to change some settings, but of using a container's security features or mechanism that have no equivalent in other containers.

A side note regarding access control: as EJBs run inside a container, protecting various resources of the container itself is highly recommended. For instance, controlling who can use a database pool or perform JNDI lookups. Authorization on these objects is vendor-

specific, but it is important to take this into account. If the container is not secured, then the application running inside it is not.

EJB 2.0 specifies that interoperability between containers must be done by implementing level 0 of the Common Secure Interoperability v2 specification (see the section on CSIv2 later). Chapter 19, "Support for Distribution and Interoperability ", of [Hur1999] details the expected behaviour for EJB clients and servers.

Regarding auditing, whereas the current specs do not cover security management, the JSR-77 (J2EE Management Specifications, http://java.sun.com/jcp/jsr/jsr_077_management.html) will address security administration as well.

[Beznosov2001] has a good analysis of the EJB 2.0 security. Vendor-specific problems are usually addressed in dedicated online forums.

The following table summarizes the EJB security as seen from the security services provided:

| Service | Comments |
|---|---|
| Authentication | vendor-specific, with JAAS mandatory in J2EE 1.3. Delegation of identity is achieved based on trust not propagation of authentication data. Anonymous (non-authenticated) clients optionally supported. |
| Authorization | declarative and role-based for EJBs, some programmatic support |
| Confidentiality | usually SSL-based, but may be vendor-specific |
| Integrity | usually SSL-based, but may be vendor-specific |
| Non-repudiation support | not available |
| Auditing | not specified, vendor-specific. |
| Administration | - vendor-specific for the server itself and for role-to-principal mapping<br>- direct or tool-based editing for the XML deployment descriptors<br>- vendor-specific for other resources and associated ACLs<br>- possibly addressed by the standard in future releases, see JSR77 |

### 9.2.3 CORBA

CORBA is a specification and its security coverage is quite ambitious and complex. It is by far the most complicated among the wide-spread distributed object technologies. The CORBASec service is formally defined in [OMG2001a], a hefty 434 pages. [Blakley1999] is a great conceptual introduction whereas [Beznosov2001] covers the topic from an enterprise perspective, including interoperability with EJB.

For the purpose of this document, the first comment is that developers will not work with the CORBASec specifications, but with implementations of it. As with the rest of the CORBA specs, vendors can implement more or less and thus the very first action is to check whether a particular vendor has the desired features implemented. That is, first ask WHAT.

CORBASec defines two functionality levels:

- *level 1* is intended for applications unaware of security and for those with limited requirements to enforce their own authorization and auditing.

- *level 2* is intended for security-aware applications, including administration of security policy

There are also *optional* features such as non-repudiation and replaceability of security services.

Even within a level, check what features of that level are supported by vendors. You may want to check the vendor's conformance statement for a more formal document (compared to sales brochures). Familiarity with the terminology in the CORBA specifications is very helpful. For instance, as per [OMG2001a], the claim of "conformance to CORBA Security" only requires support for level 1. It does not mean the *entire* CORBASec specification!

The next step is to ask HOW. As CORBA specifies the interfaces, not the implementations, it is relevant to understand what is behind the interfaces. Given that not all mechanisms are alike, understanding the level of security directly affects the risk management approach. As an example, suppose message confidentiality is implemented through symmetric encryption. Whether it is DES or AES it matters significantly for the level of protection (DES is no longer considered secure enough), especially for the future. Further, symmetric algorithims have different operating modes (such as ECB, CBC, CFB or OFB, see [Menezes1996]) with different strength levels. Some can be broken faster and this is important to know.

Communication is another topic for product reviews. Firewalling CORBA traffic (more precisely, configuring firewalls to allow CORBA traffic without introducing other risks) is covered in [OMG1998] and [Schreiner1999].

Other questions to ask your ORB vendor are found in [Blakley1999].

### 9.2.4 DCOM

*For more information on COM security, please first see the section on COM.*

Microsoft's DCOM is built on top of RPC. Security-wise, DCOM poses some problems because it is not a firewall-friendly protocol. You can learn details from a paper on this very topic, please see [Nelson1999]

At the end of the day, you will still have to open ports that administrators are not comfortable with. Allowing incoming traffic to the RPC port mapper (port 135) is not without risks: using an RPC end point mapper (either as a separate implemention or as part of a security scanner) an attacker can learn a number of things about a server.

*COM Internet Services* [Levy1999] has not really taken off (it used port 80 but not as an HTTP-based protocol), please see the section on SOAP for a much better solution.

### 9.2.5 RMI

RMI (Remote Method Invocation) by itself does not have much security and it does not look it will have soon. The "RMI Security for J2SE" Specification Request (JSR 76) was rejected in February 2001 so JDK 1.4 came out without RMI security. You can still read public documents at http://www.jcp.org/jsr/detail/76.jsp.

Note that the RMISecurityManager class does not provide security for the RMI traffic, but simply to define how the RMI-based dynamic code download will take place.

RMI can work over JRMP or IIOP. JRMP (Java Remote Method Protocol) is rather simple wire protocol developed by Sun for the first releases of JDK. It has several methods of connecting across the firewall: directly, tunnelled through HTTP and relayed through a CGI. The first would generally be less appropriate in a secure environment. You can read about the pros and cons of the other in [RMI1999].

RMI over JRMP is being replaced by RMI over IIOP. Using IIOP, the standard transport protocol in CORBA, opens the door to interoperability between Java and CORBA applications.

### 9.2.6 IIOP

IIOP (Internet Inter-Orb Protocol) is a central protocol in the CORBA world and has assumed the role of the primary communication protocol for EJB 2.0 as well. IIOP maps the CORBA GIOP (General Inter-Orb Protocol) messages to TCP traffic. Two approaches have been used to secure IIOP.

#### 9.2.6.1 SECIOP

SECIOP (Secure Inter-Orb Protocol) is formally described in [OMG2001a]. SECIOP allows its mechanisms to run over SPKM, Kerberos or CSI-ECMA. Check with the ORB vendor what mechanism is actually supported.

#### 9.2.6.2 SSLIOP

As IIOP runs on TCP/IP, the SSLIOP approach is to perform the IIOP communication through an SSL-secured channel thus inheriting the authentication, confidentiality and integrity to the SSL layer. SSLIOP has the advantage of using a familiar and widely implemented technology, but, by the design of SSL, it only secures point to point connections. In complex enterprise systems, the lack of end to end security may be a concerned.

### 9.2.7 CSIv2

The initial CORBASec specifications had some vague areas that led to interoperability problems between ORB vendors. The situation has improved with the approval of the *Common Security Interoperability version 2* protocol which also allows interoperability between CORBA and EJB, two essential technologies in the enterprise world. The formal specification is in [OMG2001b]

CSIv2 defines 3 conformance levels (their detailed description is in the spec), of which only the most basic one, level 0, is required for EJB interoperability.

[Beznosov2001] has a good coverage of CSIv2, including of the interoperability with EJB 2.0

### 9.2.8 SOAP

SOAP (v1.1 defined in [W3C2000] and the draft of v1.2 in [W3C2001b]) is a transport-independent, XML-based protocol for exchanging information, primarily designed to overcome the limitation of previous, proprietary & firewall-unfriendly protocols for RPC.

Using SOAP over HTTP is the typical scenario although, again, SOAP is transport-independent. Usage of HTTP makes the protocol more firewall-friendly as instead of opening an arbitrary range of ports, now only one open port is required. SOAP defines several specific headers for use in HTTP.

However, SOAP does not define any security mechanism. The official specification leave this task for future versions. It is up to the transport mechanism to offer the required security. Since HTTP is the most common transport, a frequent question is "if SOAP can work over HTTP, then the problem is solved by using HTTP over SSL. Or let's use IPSec to enforce network-level security". Well, this does not quite solve the problem..

Yes, using HTTPS as a transport protocol or IPSec as a network protocol offer a quick path to the security services offered by SSL/IPSec, namely authentication, confidentiality and integrity. It could be the appropriate solution for certain environments. There are disadvantages though:

- no application-level security. By the time the message is delivered to the SOAP parser, the task of the network and transport layers is done. The application is left with a bare-bones SOAP message without being able to control the security policy or behaviour of the underlying layers.

- no end-to-end security. In a complex environment where SOAP messages are carried through intermediaries, a transport-level security mechanism such as SSL requires point-to-point SSL sessions between intermediate nodes, which thus must decrypt the message and re-encrypt it to the next hop. Even worse, there is no guarantee the desired security level will be maintained as the messages go from one hop to another. It may happen that between some of them the SOAP data will be transported in clear text, without the end entities being able to control it. For sensitive applications, the lack of end-to-end security is simply unacceptable.

- there is no persistence of the security data. For some applications it is desired to store the security attributes of a SOAP exchange. However, this is not possible with SSL or IPSec because such information is outside of the SOAP parser. Nor is it possible to use mechanisms supporting non-repudiation.

Proposals have been made to enhance the SOAP specification to incorporate security features: [Hada2000] for encryption and authorization and the more recent [W3C2001a] and [Hada2001] for digital signatures.

## 9.3 The Database Layer

Database servers have been around for longer than web or application server. Consequently, their security model has had time to mature. Major security services (authentication, authorization, confidentiality, integrity and accountability) are present in any serious database server.

As with other layer, good practices recommend containing the access to the level that is needed. In an environment that provides sophisticated privilege controls and where the complexity of the system can lead to subtle vulnerabilities, controlling what database users can do is critical. Some suggestions:

- make use of the existing security mechanisms. This seems trivial, but often the database server is underutilized in terms of security capabilities.

- treat database security as an area in itself. Just because the application will not allow a user's access to certain tables does not mean all tables should be accessible to everyone. Databases servers can be accessed from other clients as well, not necessarily from the application.

- be granular with the resources. If all the data is not to be treated alike, then make the separation clear in terms of permissions.

- be selective with the operations. If certain tables or views are for read-only purposes, then enforce this restriction. Such restrictions can be the ultimate safety net if the previous layers have failed to catch a malicious attempt.

- consider what and how is backed-up, especially when sensitive data is at stake.

- input validation: we have already talked about the need to validate the data provided by a user. This is equally relevant to the database layer.

Good resources on database security are http://www.sqlsecurity.com (for Microsoft SQL Server) and [Newman2001] and[Finnigan2001] for Oracle.

## 9.4 Security Protocols

Protocols are critical to a system's security. Critical from both the value they bring (without them all pieces would be separated) and from the risks they cause. By definition, data vehicled through a communication protocol traverses other systems or networks, thus potentially becoming exposed to an attacker.

The design and evaluation of protocols used for security takes a significant slice from the overall effort spent on security. We will look at some of the security protocol an application developer may meet. Note that we will not discuss VPN protocols (like IPSec or PPTP). Most readers from the development community will not be actively involved in developing VPN software but in building systems that work on top of the network infrastructure.

Also, one of the most famous security protocols, Kerberos, is covered later in the section on distributed security services.

### 9.4.1 SSL & TLS

SSL (Secure Sockets Layer) is often misunderstood ("we use SSL therefore we are secure"). SSL is in fact a common term for several protocols (formally for SSLv2 and SSLv3 and, incorrectly but very used, for TLSv1). [Murray2000] contains an interesting survey on how SSL is actually used (and mis-used).

SSL almost invariably causes the word "certificates" to pop up in one's mind, however, TLS has been expanded to support Kerberos-based authentication ([Hur1999]), a good move towards enterprise integration (where Kerberos is used).

A development mistake is to leave SSL disabled close to the day of going live, in the idea that "everything runs the same except it is encrypted". Almost true, but this "almost" is not "totally". Here is what such an approach would lead to:

- performance is not tested with SSL. As we have already mentioned in the section on performance, crypto operations are CPU intensive and the same traffic with or without SSL makes a significant difference on the server's performance. A properly done project will have a distinct separation between sensitive data (to be protected by SSL) and non-sensitive data (to be left in clear). Typically the latter consists of images and other GUI elements that make no sense to be encrypted. Such separation may affect a website's map, thus affecting the development. A very good paper on SSL performance can be found is [Intel2000]. [Rescorla2000] covers the topic as well.

- SSL functionality is not tested, neither on the server nor on the client. Are all links generated correctly with https instead of http? Does the client have the necessary CA certificate(s) to validate the server certificate? Is the https traffic allowed to the existing firewall? Is the process of renewing the certificate(s) documented and explained to the maintenance team?

[Rescorla2000] is the definitive work on the SSL design, make sure you have it in your library. Apart from the high quality technical content, readers will also learn how politics and competition shaped the SSL protocol. [Thomas2000] is another book with high ratings.

If SSL is available in many commercial or open-source toolkits. When selecting one, pay attention to the cipher suites that are actually implemented, especially if you are looking for more recent features such as Kerberos.

### 9.4.2 SRP

The *Secure Remote Password* protocol, defined in [Wu2000], is a password-based user authentication and key-exchange mechanism. It is open-source and has a web site at http://srp.stanford.edu/. [Taylor2001] describes how SRP can be used for TLS authentication, instead of certificates.

### 9.4.3 SPEKE

SPEKE, standing for *Strong Password Exponential Key Exchange*, is another mechanism for password-based authentication and key exchange. Details are available at http://www.integritysciences.com/speke.html

### 9.4.4 SPNEGO

The *Secure and Protected GSS-API NEGOtiation* protocol, defined in [Baize1998], allows two parties to negotiate and then invoke a security mechanism that both support. If you need to implement such a handshake in your system, consider SPNEGO before designing a similar solution from scratch.

### 9.4.5 A Formal Approach: CSP

If designing protocols is what you are interested in, [Goldsmith2000] will be a useful reference to check. The British authors introduce a "mathematical framework for describing and analyzing the interactions between distributed agents, and one of the most powerful tools available for designing, verifying, and evaluating highly secure protocols" (we have quoted from the book description). Using their approach, protocols can be described in an algebra named *Communicating Sequential Processes* which can then be algorithmically evaluated for security flaws. The authors provide the Casper software used for this purpose.

# 10 Tools

The following tools are available for application-level security. [Wheeler2001] contains a brief review of some of the tools listed below. The tools are listed here in no particular order.

## 10.1 Source-code analyzers

These are tools that perform a (hopefully intelligent) pattern matching on source code in order to identify functions or constructs that are potentially dangerous.

### 10.1.1 ITS4

ITS4 is available as source code from http://www.cigital.com/its4/ and scans C and C++ code.

### 10.1.2 flawfinder

Flawfinder is developed by David Wheeler and available freely from http://www.dwheeler.com/flawfinder/

### 10.1.3 RATS

Rough Auditing Tool for Security, http://www.securesw.com/rats/ , for C, C++, Python, Perl and PHP

## 10.2 Application-level scanners

Both commercial products below are intended for web applications.

### 10.2.1 Sanctum AppScan

http://www.sanctuminc.com/

### 10.2.2 spiDYNAMICS WebInspect

http://www.spidynamics.com/webinspect.html

## 10.3 HTTP clients

### 10.3.1 HTTPush

"HTTPush aims at providing an easy way to audit HTTP and HTTPS application/server security. It supports on-the-fly request modification, automated decision making and

vulnerability detection through the use of plugins and full reporting capabilities". More at the project's web site, http://sourceforge.net/projects/httpush

### 10.3.2 Whisker and RFProxy

We have grouped these two because they are written by the same author, rain.forest.puppy. Whisker is an advanced (and well-known) CGI scanner while RFProxy has the same intent as HTTPPush. See http://www.wiretrip.net/rfp/

### 10.3.3 Elza & Scanning Web Proxy

Other two tools written by the same author, Philip Stoev. Details at http://www.stoev.org/

### 10.3.4 curl

Curl is a tool for transferring files with URL syntax, supporting FTP, FTPS, HTTP, HTTPS, GOPHER, TELNET, DICT, FILE and LDAP. Curl supports HTTPS certificates, HTTP POST, HTTP PUT, FTP uploading, Kerberos, HTTP form based upload, proxies, cookies, user+password authentication, file transfer resume, http proxy tunneling and a busload of other useful tricks.". See http://curl.haxx.se/

# 11 Other sources

The References section contain a number of works more or less related to the focus of this document, but for extended and dedicated resources on application security, please check the following.

Books:

- John Viega, Gary McGraw, **Building Secure Software: How to Avoid Security Problems the Right Way**, Addison Wesley 2001 (I haven't read it yet but this will surely be a quality resource given the authors)

- Michael Howard, David Leblanc, **Writing Secure Code**, Microsoft Press, 2001 (scheduled for November 2001 so not published yet. Same expectation as above).

- David A. Wheeler's Secure **Programming for Linux and Unix HOWTO**, http://www.dwheeler.com/secure-programs/ (referred to as [Wheeler2001] throughout this document)

- Julie Traxler, Jeff Forristal, Ryan Russell, **Hack Proofing Your Web Applications,** Syngress Media, 2001

Mailing lists:

- The **SECPROG** mailing list hosted by Security Focus, http://www.securityfocus.com

- The **WWW MOBILE** CODE (to be renamed **WEBAPPSEC**) mailing list hosted by Security Focus, http://www.securityfocus.com

WWW documents:

- Lincoln Stein & John Stewart, **The World Wide Web Security FAQ**, http://www.w3.org/Security/faq/

- **Secure Programming** v1.00, a document associated with the SECPROG list.

- **Open Web Application** Security **Project**, http://www.owasp.org

# 12 References

[Adams1996] Carlisle Adams, *The Simple Public-Key GSS-API Mechanism (SPKM)*, RFC2025, http://www.ietf.org/rfc/rfc2025.txt

[Adams1998] Carlisle Adams, *Independent Data Unit Protection Generic Security Service Application Program Interface*, RFC2479, http://www.ietf.org/rfc/rfc2479.txt

[Adams1999] Carlisle Adams, Steve Lloyd, *Understanding the Public-Key Infrastructure*, New Riders Publishing, 1999

[Advosys2001] *Writing Secure Web Applications*, http://advosys.ca/tips/web-security.html

[Aleph1997] Aleph One, *Smashing The Stack For Fun And Profit*, Phrack #49, http://www.securityfocus.com/data/library/P49-14.txt

[Alforque2000] Maricia Alforque, *Creating a Secure Windows CE Device*, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnce30/html/winsecurity.asp

[Allaire2001] *Allaire Security Bulletin (ASB00-05)*, http://www.allaire.com/handlers/index.cfm?id=14557&method=full

[Apache2000] *Cross Site Scripting Info*, http://httpd.apache.org/info/css-security/index.html

[Baize1998] E. Baize, D. Pinkas, *The Simple and Protected GSS-API Negotiation Mechanism*, RFC2478, http://www.ietf.org/rfc/rfc2478.txt

[Barcalow1998], Joseph Yoder. Jeffrey Barcalow, *Architectural Patterns for Enabling Application Security*, www.joeyoder.com/papers/patterns/Security/appsec.pdf/yoder98architectural.pdf

[Barkley1997] John Barkley, *Comparing Simple Role Based Access Control Models and Access Control Lists*, http://hissa.ncsl.nist.gov/rbac/iirf.ps

[Bell1975] D.E. Bell, L.J.LaPadula, *Secure Computer Systems: Unified Exposition and Multics Interpretation*, Technical Report ESD-TR-75-306, MITRE Corporation

[Bellare1997] H. Krawczyk, M. Bellare, R. Canetti, *HMAC: Keyed-Hashing for Message Authentication*, RFC2104, http://www.ietf.org/rfc/rfc2104.txt

[Beznosov2000] Konstantin Beznosov, *Engineering Access Control for Distributed Enterprise Applications*, Florida International University

[Beznosov2001] Bret Hartman, Donald J. Flinn, Konstantin Beznosov, *Entreprise Security with EJB and CORBA*, Wiley Computer Publishing, 2001

[Birznieks1998], Gunther Birznieks, *CGI/Perl Taint Mode FAQ*, http://www.gunther.web66.com/FAQS/taintmode.html

[Blakley1999], Bob Blakley, *CORBA Security*, Addison-Wesley, 1999

[Blixt1999] Karl-Fredrik Blixt, Åsa Hagström, *Adding Non-Repudiation to Web Transactions*, http://www.it.isy.liu.se/~asa/publications/NonRepudiation.html

[Brown2000a] Keith Brown, *Programming Windows Security*, Addison-Wesley

[Brown2000b] Keith Brown, *Web Security: Putting a Secure Front End on Your COM+ Distributed Applications*, http://msdn.microsoft.com/msdnmag/issues/0600/websecure/websecure.asp

[Brown2000c] Keith Brown, *Web Security: Part 2: Introducing the Web Application Manager, Client Authentication Options, and Process Isolation*, http://msdn.microsoft.com/msdnmag/issues/0700/websecure2/websecure2.asp

[Brown2000d] Keith Brown, *Explore the Security Support Provider Interface Using the SSPI Workbench Utility*, http://msdn.microsoft.com/msdnmag/issues/0800/security/security0800.asp

[Brown2001] Keith Brown, *Security in .NET: Enforce Code Access Rights with the Common Language Runtime*, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnmag01/html/CAS.asp

[Bütler2001] Ivan Bütler, *Oracle Checklist v2.0*, Compass Security AG

[Caelli2000] Adrian McCullagh, William Caelli, *Non-repudiation in the digital environment*, http://www.firstmonday.dk/issues/issue5_8/mccullagh/

[Canada2000]: *The Personal Information Protection and Electronic Documents Act*, http://www.privcom.gc.ca/english/02_06_01_e.htm

[CERT2000a] *Results of the Security in ActiveX Workshop*, CERT Coordination Center, http://www.cert.org/reports/activeX_report.pdf

[CERT2000b] CERT Advisory CA-2000-02 *Malicious HTML Tags Embedded in Client Web Requests*, http://www.cert.org/advisories/CA-2000-02.html

[CERT2000c] *Understanding Malicious Content Mitigation for Web Developers*, http://www.cert.org/tech_tips/malicious_code_mitigation.html

[Coffman2001] Olga Kornievskaia, Peter Honeyman, Bill Doster, Kevin Coffman, *Kerberized Credential Translation: A Solution to Web Access Control,* http://www.securityfocus.com/data/library/citi-tr-01-5.pdf

[Cugini1999] David F. Ferraiolo, Janet A. Cugini, D. Richard Kuhn, *Role-Based Access Control (RBAC): Features and Motivations*, http://hissa.ncsl.nist.gov/rbac/newpaper/rbac.ps

[Curtin1998] Matt Curtin, *Snake Oil Warning Signs: Encryption Software to Avoid*, http://www.interhack.net/people/cmcurtin/snake-oil-faq.html

[DilDog1998] DilDog, *The TAO of Windows Buffer Overflow*, http://www.cultdeadcow.com/cDc_files/cDc-351/

[Dulepet], Rajiv Dulepet, *COM Security in Practice*, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dncomg/html/msdn_practicom.asp

[Ellison2000] C. Ellison, B. Schneier, *Ten Risks of PKI: What You're Not Being Told About Public Key Infrastructure*, http://www.counterpane.com/pki-risks.html

[Evans2001] David Larochelle, David Evans, *Statically Detecting Likely Buffer Overflow Vulnerabilities*, http://lclint.cs.virginia.edu/usenix01.pdf

[Feamster2001], Kevin Fu, Emil Sit, Kendra Smith, Nick Feamster, *Dos and Don'ts of Client Authentication on the Web*, http://cookies.lcs.mit.edu/

[Feringa2001] Gary Stoneburner, Clark Hayden and Alexis Feringa, *NIST Special Publication 800-27 Engineering Principles for Information Technology Security*

[Finnigan2001] Pete Finnigan, *Exploiting And Protecting Oracle*, http://www.pentest-limited.com/oracle-security.pdf

[Foster2001] Umesh Shankar, Kunal Talwar, Jeffrey S. Foster, David Wagner, *Detecting Format-String Vulnerabilities with Type Qualifiers*, http://www.cs.berkeley.edu/~jfoster/papers/usenix01.pdf

[Goldsmith2000] Peter Ryan, Steve Schneider, Michael Goldsmith, Gavin Lowe, Bill Roscoe *The Modelling and Analysis of Security Protocols: the CSP Approach,* Addison Wesley, 2000 (also see the associated web site at http://www.cs.rhbnc.ac.uk/books/secprot/)

[Gong1999] Li Gong, *Inside Java 2 Platform Security,* Addison-Wesley, 1999

[Gregorini2000a] Marco Gregorini, *The Subtleties of Client Impersonation with IIS, ASP and MTS/COM+ - Part I*, http://www.asptoday.com/articles/20000224.htm

[Gregorini2000b] Marco Gregorini, *The Subtleties of Client Impersonation with IIS, ASP and MTS/COM+ - Part II*, http://www.asptoday.com/articles/20000302.htm

[Grimes2001] Roger A. Grimes, *Malicious Mobile Code* , O'Reilly, 2001

[Hada2000] Satoshi Hada, Hiroshi Maruyama, *SOAP Security Extensions*, http://www.trl.ibm.com/projects/xml/soap/wp/wp.html

[Hada2001] Satoshi Hada, *SOAP-DSIG and SSL*, http://www-106.ibm.com/developerworks/webservices/library/ws-soapsec/?open&l=860,t=grws,p=SoapSig

[Hall1999] John Kelsey, Bruce Schneier, David Wagner, Chris Hall, *Cryptanalytic Attacks on Pseudorandom Number Generators*, http://www.counterpane.com/pseudorandom_number.html

[Honeyman2001] Niels Provos, Peter Honeyman, *Detecting Steganographic Content on the Internet*, http://www.citi.umich.edu/techreports/reports/citi-tr-01-11.pdf

[Howard2000a] Michael Howard, *Secure Systems Begin with Knowing Your Threats - part I*, http://security.devx.com/upload/free/Features/zones/security/articles/2000/09sept00/mh0900[2]-1.asp

[Howard2000b] Michael Howard, *Secure Systems Begin with Knowing Your Threats - part II*, http://security.devx.com/upload/free/Features/zones/security/articles/2000/10oct00/mh1000[1]-1.asp

[Hur1999] A. Medvinsky, M. Hur, *Addition of Kerberos Cipher Suites to Transport Layer Security*, RFC 2712, http://www.ietf.org/rfc/rfc2712.txt

[Intel2000] *Designing a Secured Website: What You Need to Know About SSL Benchmarking*, http://www.intel.com/network/documents/pdf/sslbenchmarking.pdf

[ISS2000] *Form Tampering Vulnerabilities in Several Web-Based Shopping Cart Applications*, http://xforce.iss.net/alerts/advise42.php

[Jaworski2000] Jamie Jaworski, Paul Perrone, *Java Security Handbook*, Sams, 2000

[Kabat2000] J. Kabat, M. Upadhyay, *Generic Security Service API Version 2 : Java Bindings*, RFC2853, http://www.ietf.org/rfc/rfc2853.txt

[Kercher2001] Jeff Kercher, *Authentication in ASP.NET: .NET Security Guidance*, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/authaspdotnet.asp

[Kingpin2001] Kingpin and Mudge, *Security Analysis of the Palm Operating System and its Weaknesses Against Malicious Code Threats*, http://www.atstake.com/research/reports/security_analysis_palm_os.pdf

[Kohl1993] J. Kohl, C. Neuman, *The Kerberos Network Authentication Service (V5)*, RFC1510

[Levy1999], Marc Levy, *COM Internet Services*, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dndcom/html/cis.asp

[Linn1996] J. Linn, *The Kerberos Version 5 GSS-API Mechanism,* RFC1964, http://www.ietf.org/rfc/rfc1964.txt

[Linn2000] J. Linn, *Generic Security Service Application Program Interface*, *Version 2, Update 1,* RFC 2078, http://www.ietf.org/rfc/rfc2743.txt

[Luzwick2000] Perry G. Luzwick, *Value of Information,* http://www.shockwavewriters.com/Articles/PGL/kilo.htm

[Mcgraw2001] Gary Mcgraw, John Viega, *Practice Safe Software Coding*, Information Security Magazine, http://www.infosecuritymag.com/articles/september01/features_securing.shtml

[Menezes1996] A. Menezes, P. van Oorschot and S. Vanstone, *Handbook of Applied Cryptography*, http://www.cacr.math.uwaterloo.ca/hac/

[Microsoft2000] *HOWTO: Prevent Cross-Site Scripting Security Issues*, http://www.microsoft.com/technet/support/kb.asp?ID=252985

[Microsoft2001a] *Microsoft Security Best Practices* http://www.microsoft.com/technet/treeview/default.asp?url=/technet/itsolutions/security/bestprac/bestprac.asp

[Mixter2001] Mixter, *Guidelines for C source code auditing*, http://mixter.void.ru/vulns.html

[Mudge1997] Mudge, *How to Write Buffer Overflows*, 1997 http://l0pht.com/advisories/buffero.html

[Murray2000] Eric Murray, *SSL Server Security Survey*, http://www.meer.net/~ericm/papers/ssl_servers.html

[Nelson1999] Michael Nelson, *Using Distributed COM with Firewalls*, http://www.microsoft.com/com/wpaper/dcomfw.asp

[Newman2001], Marlene Theriault, Aaron Newman, *Oracle Security Handbook*, McGraw-Hill, 2001

[Newsham2000] Tim Newsham, *Format String Attacks*, http://www.guardent.com/docs/FormatString.PDF

[NIST2000] *Random Number Generation and Testing*, http://csrc.nist.gov/rng/

[Oaks2001] Scott Oaks, *Java Security*, 2nd Edition, O'Reilly, 2001

[OMG1998] *Joint Revised Submission CORBA/Firewall Security*, 1998, The Object Management Group

[OMG2001a] *Corba Security Service Specification v1.7*, March 2001, The Object Management Group

[OMG2001b] *Common Security Interoperability V2*, March 2001, The Object Management Group

[Opengroup2001] *Guide to Security Patterns* http://www.opengroup.org/security/gsp.htm

[OSSTMM2001] *The Open Source Security Testing Methodology Manual* http://uk.osstmm.org/osstmm.htm

[Peteanu2001a] *Design Patterns in Security*, http://members.home.net/razvan.peteanu/designpatterns20010611.html

[Peteanu2001b], [Peteanu2001c] *Zen and the Art of Breaking Security – I, II* http://members.home.net/razvan.peteanu/zenandsecurity20010301.html http://members.home.net/razvan.peteanu/zenandsecurity20010308.html

[Provos2001] Niels Provos, *Defending Against Statistical Steganalysis*, http://www.citi.umich.edu/techreports/reports/citi-tr-01-4.pdf

[Rescorla2000] Eric Rescorla, *SSL and TLS: Designing and Building Secure Systems*, Addison-Wesley, 2000

[RIM2001a] RIM, *Technical White Paper BlackBerry Security for Microsoft Exchange version 2.1*

[RIM2001b] RIM, *Technical White Paper BlackBerry Security for Lotus Domino version 2.0*

[RMI1999] *Frequently Asked Questions RMI ,Servlets and Object Serialization*, http://gene.wins.uva.nl/~nhussein/java.html

[Romanovski2001] Sasha Romanovski, Security Design Patterns, http://www.romanosky.net

[Schneier1995] Bruce Schneier *Applied Cryptography: Protocols, Algorithms, and Source Code in C, 2nd Edition*, John Wiley & Sons, 1995

[Schneier1999a] Bruce Schneier, *Attack Trees*, SANS Network Security 99

[Schneier1999b] Bruce Schneier, *Attack Trees* , http://www.counterpane.com/attacktrees-ddj-ft.html

[Schreiner1999] Rudolf Schreiner, *CORBA Firewalls*, http://www.objectsecurity.com/whitepapers/corba/fw/main.html

[Seifried2000] Kurt Seifried, *PAM -- Pluggable Authentication Modules*, Sys-Admin, http://www.samag.com/documents/s=1161/sam0009a/0009a.htm

[Seifried2001] Kurt Seifried, *WWW Authentication*, http://www.seifried.org/security/www-auth/

[Shannon2001] Bill Shanon, *Java 2 Platform Enterprise Edition Specification*, v1.3, Sun Microsystems, 2001

[Shostack2000] Adam Shostack, *Security Code Review Guidelines*, http://www.homeport.org/~adam/review.html

[Simon2001] Istvan Simon, *A Comparative Analysis of Methods of Defense against Buffer Overflow Attacks*, http://www.mcs.csuhayward.edu/~simon/security/boflo.html

[Singh2000] Simon Singh, *The Code Book : The Science of Secrecy from Ancient Egypt to Quantum Cryptography*, Anchor Books, 1999

[SPSMM2001] *The Secure Programming Standards Methodology Manual*, http://uk.osstmm.org/spsmm.htm

[Stein1998] Lincoln D. Stein, *Referer Refresher*, http://www.webtechniques.com/archives/1998/09/webm/

[Stein2001] Lincoln D. Stein, *The World Wide Web Security FAQ*, http://www.w3.org/Security/faq/www-security-faq.html

[Sun2000] *Security Code Guidelines*, http://java.sun.com/security/seccodeguide.html

[Sun2001] Sun Microsystems, *Enterprise JavaBeans TM Specification,Version 2.0*, 2001

[Taylor2001], D. Taylor, *Using SRP for TLS Authentication*, draft-ietf-tls-srp-01, http://www.ietf.org/internet-drafts/draft-ietf-tls-srp-01.txt

[TCSEC1983] *Trusted Computer System Evaluation Criteria*, US Department Of Defense, 1983

[TESO2001] Scut/Team Teso, *Exploiting Format String Vulnerabilities 1.2,* http://www.team-teso.net/articles/formatstring/

[Thomas2000] Stephen A. Thomas, *SSL & TLS Essentials: Securing the Web*, John Wiley & Sons, 2000

[Thuemmel2001] Andreas Thuemmel, *Analysis of Format String Bugs v1.0*

[W3C2000] *Simple Object Access Protocol (SOAP) 1.1*, http://www.w3.org/TR/SOAP/

[W3C2001a] *SOAP Security Extensions: Digital Signature*, http://www.w3.org/TR/SOAP-dsig/

[W3C2001b] *SOAP Version 1.2 Part 1: Messaging Framework*, http://www.w3.org/TR/2001/WD-soap12-part1-20011002/

[Wheeler2000] David A. Wheeler, *Java Security Tutorial*, http://www.dwheeler.com/javasec

[Wheeler2001] David A. Wheeler, *Secure Programming for Linux and Unix HOWTO*, http://www.dwheeler.com/secure-programs/

[Wiegers1997] Karl Wiegers, *Seven Deadly Sins of Software Reviews*, Software Development Magazine, March 1997 (also available at http://www.processimpact.com/articles/revu_sins.pdf)

[Wray2000] J. Wray, *Generic Security Service API Version 2 : C-bindings*, RFC2744, http://www.ietf.org/rfc/rfc2744.txt

[Wu2000] T. Wu, *The SRP Authentication and Key Exchange System*, RFC 2945, http://www.ietf.org/rfc/rfc2945.txt

[Zalewski2001] Michal Zalewski, *Strange Attractors and TCP/IP Sequence Number Analysis*, http://razor.bindview.com/publish/papers/tcpseq.html