Uncovering Assumptions in Information Security

Matt Bishop[1] and Helen Armstrong[2]

[1]Dept. of Computer Science, University of California at Davis, Davis, CA 95616-8562
United States of America, Email:
[2]School of Information Systems, Curtin University, Western Australia,
Helen.Armstrong@cbs.curtin.edu.au

Abstract

The design and implementation of security is based upon many assumptions. This paper discusses the need for students to learn to question assumptions, and in so doing identify unrealistic or incorrect assumptions and any associated policies. More realistic assumptions can then made and/or procedures implemented to protect against violation of the assumptions. A number of examples in the context of teaching computer security are discussed and some methods of teaching awareness of assumptions presented.

Keywords: information security education, assumptions, lateral thinking

Introduction

In a talk at the National Information Systems Security Conference [Denning 1999], Dorothy Denning commented that she had never seen anyone break into a computer system by taking advantage of a flaw in the model of the system. Invariably they entered because there was a flaw in the implementation or operation of the system. Even had the model been proved correct for the specific requirements of the system, thereby proving that the system be secure, an attacker would have been able to gain access illicitly. The specific problem here is that the proof of security relies on assumptions. Specifically, here the assumptions are that the implementation of the model of the secure system is implemented consistently with that model, and that the system is installed, operated, maintained, and decommissioned consistently with that model. If it is not, the underlying assumptions are wrong. The proof may be correct, but it is irrelevant.

An analogy with mathematics may clarify this problem. One proves theorems by constructing a logical sequence of deductions beginning with axioms. If the axioms are incorrect, then the theorems may, or may not, be correct—even if the sequence of deductions is correct. Similarly, one proves systems secure by constructing a set of proofs that the models of the system satisfy specific security requirements (the security policy). But if the "axiom"—the belief that the system abstracts all security-relevant details of the system—is incorrect, then the "theorem"—the claim that the system is secure—may or may not be true. It is certainly unproven.

The structure of security is built upon assumptions. We assume implementations correctly enforce models, that procedures are correct, enforced, and appropriate, and that authorized administrators and users will not compromise the security of the system, either deliberately or accidentally. We assume that configuring systems will cause them to act as configured. We assume that the patches we add to improve security do not conflict with other security components or policies. If our assumptions are wrong, our system has security vulnerabilities we do not realize, and so cannot guard against.

This suggests that students must learn to question assumptions. By doing so, they can discover what security mechanisms, and in some cases policies, are unrealistic. Then they can either change the mechanisms to make more realistic assumptions, or institute procedures to detect attackers trying to violate the assumptions and break into the system.

This paper discusses the importance of assumptions in security. The next section gives several examples of actions and components central to maintaining information and system security. In these examples, scenarios demonstrate the assumptions underlying the claims of security, and show what can happen when the assumptions are erroneous. The third section discusses several techniques for teaching students how to look for assumptions, and question them.

Assumptions and [Non-]Security

We consider three different scenarios, two of which involve break-ins and compromises to information and computers, and the third of which involves a claim of non-security when in fact the system is secure by the required policy.

Firewall and System Configuration

Institutions with files distributed over many systems, or with many workstations and a set of larger servers, often choose to store files on individual systems and make them available to all. One popular protocol, NFS (Network File Protocol), provides this capability by designating certain file systems as exportable. Remote hosts can then mount these file systems, and processes on those hosts interact with the file systems as though they were local. Each exported file system has an associated access control list naming the hosts that can mount that file system, and the rights that each has once it mounts the file system (read, read and write, access as administrator, and so forth). If the access control list is empty, any host may mount the file system and access its contents.

Because of relatively small internal limits on the length of the access control list in many implementations of NFS, it is infeasible to list all hosts in the organization that need to mount the file systems. There are numerous workarounds. One is to allow any host to access the exported file systems. The danger is that, if a computer not part of the organization accesses the exported file system, the NFS server will allow it to do so. Hence this option effectively makes every exported file system available to every computer running an NFS client.

Many institutions with internal networks use firewalls to limit access to their network. That ameliorates the threat of an external host mounting the internal file systems, as the firewall can simply block requests to the NFS servers from the outside network. So, assuming the security policy of the institution disallows external mounting of internal NFS file systems, the firewall's being configured to disallow such communications appears to satisfy the policy.

Now, what assumptions are being made? One is that the firewall is configured correctly and that the implementation supports the given configuration. This involves co-ordination among the system managers running NFS and the system managers running the firewall. In particular, the former must understand that the firewall is blocking NFS messages, and the latter must understand that the internal servers have no access control protection. As Winkler amply demonstrates [Winkler 1997], coordination problems lead to security problems. Less obvious ones are the assumptions that the firewall and NFS

servers are implemented correctly. These assumptions involve trusting the vendor, and most sites have no choice other than doing so. Perhaps the most subtle assumption is that the firewall intercepts all traffic between the internal hosts and the external network. This assumption, rooted in co-operation similar to that required for the firewall, requires that modems and other network devices be placed outside the firewall. Otherwise, everyone who has a modem must provide protection equivalent to that of the firewall, lest an unauthorized remote host dial in to a modem and mount one of the NFS file systems.

When asked to analyze this situation, most students will spot the first assumption. A few will realize the effect of implementation problems. Most miss the importance of all external traffic going through the firewall, or simply assume that such traffic will do so. After all, it's a firewall, and that's what a firewall does; what good is a firewall if it does not mediate all traffic between an intranet and an internet? In practice, this is an example of a definition (that of a firewall) hiding an assumption (that of being the only point of contact between the inside network and the outside network).

Spawning Subprocesses

When a UNIX process calls the library function system(3) to run a command as a subprocess, that function executes the command as though it were typed to a command prompt at the user interface. This is a standard function taught to programming students in classes that use the UNIX (or Linux) programming environment and the C programming language. The library function, however, does not directly execute the command. Instead, it calls the user-level command interpreter (called a shell), with two parameters. The first indicates that the next parameter is to be treated as a command typed to the prompt, and the second parameter is the command to be executed, including any of its arguments. The interpreter processes the command as it processes all user inputs.

This function creates a hierarchy of assumptions. The first is that the shell being executed is the same as the shell that the programmer expects. If this assumption is incorrect, the wrong command may be executed. For example, the command "ls –l ~" lists information about files in the user's home directory in one family of shells, but about the file named "~" in another family of shells. Under this assumption lies the assumption that the shell will find the correct command. UNIX and Linux shells look for a command by checking a sequence of directories for an executable file with the same name as the command. The order in which these directories are searched is controlled by a variable named **PATH** or path (depending on the shell). This variable is not a program variable. It is a shell variable that subprocesses inherit and can read. So, if the program uses the system function to spawn the "ls" command, as in the example above, the particular program executed to instantiate that command depends upon the setting of the variable **PATH** (or path) that the program has inherited.

The second assumption is that the user's **PATH** (or path) environment variable is set so that the process will execute the same command for the user as the programmer intended. Otherwise, the command executed may do nefarious things. For example, the loadmodule compromise [CERT 1995] resulted from setting the **PATH** variable to execute a Trojaned version of the dynamic loader. One obvious solution is to reset the value of **PATH** (or path) in the argument to system that invokes the command, for example:

system("PATH=/bin:/usr/bin:/sbin; export PATH; /bin/ld xxx.o").

The assumption is that the **PATH** command changes the search path. Unfortunately, this can be defeated easily, because the shell uses other inherited environment variables to interpret the characters in the parameter to system. Again, the unquestioned (and incorrect) assumption that "PATH=/bin:/usr/bin:/sbin" always resets the value of the **PATH** variable means that an attempt to secure a program fails.1

Security Standards and Best Security Practices

Recently, a plethora of proposed standards and "best practices" have been created for computer and information security (see for example [CIS2004], [ISO2000]). The practices in question here allow one to evaluate a system by running tests, or going through a checklist audit, and generate a based upon the results of the checklist or audit. The higher the score, the more secure the system is. The evaluation criteria under these lists include configuration, maintenance, and interface issues.

The criteria, unfortunately, assume a particular model or security policy. The best practices for a government agency will vary based upon the legal constraints of that agency. A military intelligence organization's security practices would focus on guarding the confidentiality of its information. An agency rating product safety would focus on guarding the integrity of its data to ensure the information is not altered illicitly. Outside government, other policies apply. Applying a checklist or audit analysis based upon a government policy to an academic institution leads to an erroneous assessment of the security of that institution.

As an example, consider the checklist item "users should not have access to other users' directories". This is critical in systems where confidentiality is important, for example on systems where users cannot share data with one another. So, the checklist says that a system on which users can share files has 1 point deducted from its audit score. But on a system in a research lab, users will share data as part of their work. If they cannot, their work is impeded. Thus, denying read access to other users' directories may be a denial of service attack. In this case, using the same checklist will deduct 1 point for a system that prevents one type of denial of service attack, and would add 1 point if in fact the system was secure!

Analysis

The three examples underscore the role of assumptions in securing systems. Assumptions must reflect the reality of the systems. The assumptions must hold in the specific environment, under the specific installation and operation of the particular system. Otherwise, any analysis of the security of the system is faulty, because it is based upon hypotheses that are false. As any student of propositional logic knows, if your hypothesis is false, you may prove any conclusion. Assurance experts recognize this role of assumptions when they discuss the importance of security requirements analysis. The requirements embody the assumptions. The security policy reflects them. But if the design, implementation, maintenance, and operation of the system do not meet the requirements, the system is not secure. For these reasons, students must learn how to

[1] The specific method is to define the **IFS** shell variable to include the character 'P'. Then that character is treated as a blank, so the variable **ATH** is set. The original search path is still used. This works on some shells, but not all.

uncover these assumptions. Once they can find the assumptions, they can then attempt to validate them. If the assumptions are valid, further analysis can proceed as appropriate. If not, then the analysis must proceed under new, validated, assumptions. Further, the security analysts charged with defending the system can institute countermeasures to detect attempts to exploit the invalid assumptions, thereby detecting attempts to attack the system.

Teaching Assumptions

Perhaps our students show implicit trust in the technology due to contextual factors. They operate in a learning environment where there is a certain amount of protection. Making mistakes is part of the learning experience and a university provides a buffer zone from the impact of such errors and miscalculations. It is our (the academic's) responsibility to move the students out of this zone of comfort and into a zone of reality. But how do we do this effectively?

In many cases students are not aware of the complete problem, nor of the factors that constrain any solution to that problem. There is no vision beyond the task at hand and ultimate trust that all other variables will remain static and predictable. Unfortunately, computers are not predictable and the technology is constantly changing, with progress based upon sophisticated, ill-tested, but widely distributed hardware and software products. Students need to be made aware of the questionable trustworthiness of the technology, and the full extent of the risks inherent in computer systems and networks.

A key question is how one teaches analyzing a problem, situation, or system for hidden assumptions. Four methods are discussed in the remainder of this section. These are (1) relating the ideas and methods to non-technical fields [Bishop 2004]; (2) use examples, some from the field of computers, and others from different fields; (3) use various stakeholder views to build a big picture; and (4) reverse assumptions.

Relating Methods to Non-Technical Fields

Failure to question assumptions can be disastrous in other areas. Lucius Quintilius Varrus assumed that his Roman legions could easily defeat the Cherusci, whom he considered undisciplined warriors. As a result of this assumption of incompetence, the three Roman legions he commanded were destroyed in a battle in the Teutoburgian Forest in 9 AD, the most humiliating defeat that Rome had suffered. Had Varrus been less credulous, he would have questioned whether his informant was truthful in his suggestions and descriptions of the Cherusci. More recently, Adolph Hitler assumed that the Soviet Union would surrender when he attacked it. He was mistaken, and historians credit his defeat in large part to the Soviet Union's defense, and subsequent advances, against Germany. Had Hitler questioned his assumption that the Soviet Union was weak and would collapse, the history of the world would be very different.

One amusing way to teach the need to question assumptions is to use mystery stories. The television show Columbo presents open mysteries—the viewer sees the murder; the fun is watching Lt. Columbo solve the crime. He does so by asking questions that challenge the assumptions of others. Through this dialogue, he finds little details that others miss, and that they cannot explain. These details guide him to discover the killer. Also useful are the questions Columbo asks: "Why?"; "How do you know …?"; "I can't explain why …"; and so forth. These demonstrate how he is questioning the assumptions,

and probing for other, underlying assumptions. If students are not familiar with that show, other fictional detectives such as Sherlock Holmes or Nero Wolfe offer similar opportunities.

Using Examples

This leads to examples drawn from history as well as computer science. Consider Sherman's march to the sea through the state of Georgia during the United States Civil War. Alinsky describes it when discussing his rules for political organizers:

The third rule is: *Whenever possible go outside of the experience of the enemy. Here you want to cause confusion, fear, and retreat.*

General William T. Sherman, whose name still causes a frenzied reaction throughout the South, provided a classic example of going outside the enemy's experience. Until Sherman, military tactics and strategies were based on standard patterns. All armies had fronts, rears, flanks, lines of communication, and lines of supply. Military campaigns were aimed at such standard objectives as rolling up the flanks of the enemy army or cutting the lines of supply or lines of communication, or moving around to attack from the rear. When Sherman cut loose on his famous March to the Sea, he had no front or rear lines of supplies or any other lines. He was on the loose and living on the land. The South, confronted with this new form of military invasion, reacted with confusion, panic, terror, and collapse. Sherman swept on to inevitable victory. It was the same tactic that, years later in the early days of World War II, the Nazi Panzer tank divisions emulated in their far-flung sweeps into enemy territory, as did our own General Patton with the American Third Armored Division.2

Sherman questioned the assumption that all armies needed supply lines. He did this quite consciously, informing President Lincoln that his army could live off the land, and by doing so he could "make Georgia howl". His army was much larger and better equipped than the Confederate army he faced. The result was that the South could not cut his supply lines, and could not stop his army—which hastened the collapse of the Confederacy.

Sherman questioned the need for supply lines. Jeb Stuart Magruder questioned the assumption that the Union army, having hundreds of thousands of soldiers, would attack Yorktown, which was defended by less than 10,000 soldiers. The Confederate army was retreating, and the Union Army of the Potomac, led by General McClellan, was poised to attack. Magruder was ordered to delay the attack. He had been an actor at one time, and put his thespian talents to good use. One morning he sent a column along a road that was heavily wooded except for a single gap in plain view of the enemy outposts. All day the gray files swept past in seemingly endless array, an army gathering in thousands among the pines for an offensive. They were no such thing, of course. Like a low-budgeted theatrical director producing the effect with an army of supernumeraries, Magruder was marching a single battalion round and around, past the gap, then around under cover, and past the gap again.[3]

Everyone expected McClellan to attack immediately. Magruder knew that his opponent was very cautious, so the assumption that the overwhelming strength of the Union army would defeat the Confederate forces was valid only if the Union army

[2] [Alinsky 1972], pp. 127–128.
[3] [Foote 1958], p. 399

actually attacked. Hence his show. McClellan dug in, ordered siege guns brought up, and prepared to bombard Yorktown to drive the massive army of defenders out. The night before the bombardment was to begin, the Confederates simply walked away, and the next morning Union troops entered the town with no resistance—no shelling necessary.

These two examples are most useful when students have studied some military history or political science, because they then understand many of the tactics under discussion. The discussion begins with enough history to set the stage for the events; it is best to pick a well-known time period or set of events so little discussion is necessary. Then the scenario is presented, as above, and students are asked why the events unfolded the way they did, with an emphasis on what assumptions were likely being made by all parties, and what role those assumptions played in the events.

Modern political events are another good place to draw on examples. Consider electronic voting, a controversial topic in the United States preceding the November 2004 election. The issue was whether "direct recording electronic" voting machines, called DREs, are trustworthy systems. The specific problem is that DREs do not provide a "voter verifiable audit trail", so the only record of votes is stored in the DRE and on flash memory. If a recount is required, the election officials print copies of the ballots stored in the DRE memory (or on the associated flash cards), and manually count those paper copies. Of course, if the votes are misrecorded in the memory, the paper copies reflect those incorrect votes.

The problem of trust in elections that involve DREs raises many interesting assumptions, among which are:

- The vendors of DREs have programmed their systems to count votes correctly;
- The independent testing authorities (ITAs) will find any problems with the DREs;
- The DREs have not been compromised in any way between elections;
- The vote counting server counts votes correctly;
- Procedural controls during the election prevent compromise of these systems and ensure that votes are counted correctly; and
- Only software certified in accordance with the law will be used on these systems.

A good classroom exercise is to ask what happens when one or more of these assumptions fail. A better one is to ask what assumptions underlie these; for example, the second assumption above is based on the ITAs being competent to do adequate testing, and the standard against which they test are also adequate. This leads to a discussion of the nature of requirements analysis, for example.

Moving on to specific computer security scenarios, consider a forensic analysis of a break-in. The computer may have been compromised, because it is acting oddly; but there is no other indication of problems. The system has been rebooted[4] but continues to function oddly. How would you proceed with the analysis? The usual answers focus on trying different programs to analyze the system, possibly loading them from CDs. But the key assumption is that the kernel is operating correctly. This assumption may be bogus, because certain versions of *rootkit*, a system modification tool that compromises system calls to hide processes and files, exists as a kernel loadable module. Thus, the only way to detect it is to boot from a clean installation or analysis CD. This exercise focuses the

[4] In a forensic analysis, rebooting destroys evidence of which processes are running; but it is one of the first steps people take when their machine appears to malfunction.

students on the problems of forensic analysis and how the "virtual world" can be easily hidden.

Lateral and Holistic Thinking

A method used in the study of history is lateral thinking, or taking on the viewpoint of others involved in the situation under consideration, and looking at the impact of possible outcomes. This approach views the situation at hand not just as an isolated task or incident, but as a part of a larger picture, encompassing a much wider view. Edward de Bono [de Bono 1971] introduced the term "lateral thinking" but the technique has been used for thousands of years. Seeing the situation from several viewpoints forms part of numerous problem-solving approaches (for example, see [Adams 2001], [Foster 1996], and [Michalko 2001]).

In order to make informed decisions, perspectives are drawn from the views of all stakeholders involved in the given situation. A stakeholder is anyone or anything that forms part of a given situation, contributes to that situation, or could be affected either positively or negatively from the proposed action. Stakeholders can be both objects and subjects, and may include individuals, organizations, devices, code, data and the like. In the firewall and system configuration example in Section 3.1 above, stakeholders would include the remote hosts, NFS servers, modems, firewall, internal servers, traffic, and the like. Some stakeholders, such as an attacker, will not be available. Students must take on the mindset of that stakeholder. Each stakeholder's perspective includes the goals or objectives of that stakeholder, plus the assumptions that stakeholder makes. Based on these assumptions, students also study the consequences of the intended action on each stakeholder.

A holistic perspective is developed from the combination of the stakeholders' views, presenting a comprehensive rich picture. Not only does it encourage discovery of assumptions held by each party, but also the amalgamation of the expected outcomes arising from the different stakeholders' goals is enlightening and often does not reflect the overall goal of the action. In addition, detailing the assumptions made by each party aids in identifying expectations. These can be mapped back to the original requirements. Thus the compiling of the big picture in this way paints a more informative portrait of the entire situation and highlights potential or hidden problems.

Students can work in a group, each taking the view of a different stakeholder and the group piecing together the big picture combining all views and assumptions. Overlaps and conflicts can be easily spotted in the bigger picture.

Assumption Reversal

Reversing the identified assumptions sheds an entirely new light on the situation and often presents a worst-case scenario. Assumption identification and reversal is a technique used extensively in creative thinking and problem solving, and aids in risk identification and management in a security context. Assumption reversal can be used as a tool to foster assumption identification and also to remove the blinkers to our thinking in problem-solving. Using this technique, all assumptions are identified within the problem situation at hand, then reversed. The reversal, in effect, dissipates traditional thinking patterns, allowing information to link in provocative new ways [Michalko 2001].

Combined with other techniques above, assumption reversal can be a powerful technique when applied to the area of computer security. To see an impending threat requires people to challenge their assumptions [Sherman & Schultz 1998]. In assumption reversal, this involves not only a search for, and identification of, those assumptions but also a question "why?" and then reversing those assumptions and asking "how likely?". In an approach similar to risk analysis, the implications of each assumption and its reversal are determined.

Take again the example of the firewall and system configuration issue in Section 3.1. One assumption is that the firewall intercepts all traffic between the internal hosts and external networks. Once this assumption has been identified, it is then questioned (e.g. "does it?" "why?") and reversed ("assume the firewall does not intercept all traffic between the internal hosts and external networks"). If an assumption cannot be tested and verified, then the reverse of that assumption must hold. For example, if we cannot test whether the firewall does actually intercept all traffic, then we must assume it does not. What are the implications of these assumptions on our system? How can we ensure the security solution works and achieves what is required?

This teaching technique can be used for any situation ranging from the application of an operating system service pack to the installation of a biometric authentication system. It teaches students to think beyond the accepted boundaries and place intended actions into a much larger perspective. It encourages students to seek out the bigger picture before making decisions and highlights the impact of actions on all stakeholders before proceeding.

Summary

Each of the four methods has its own use. The first method, relating computer security methods to non-technical fields, is most useful in an introductory class in computer security because it relates technical problems to problems familiar to the students. Examples make the methods come alive. Experienced instructors know that the best way to engage students is not by going from theory to examples, but by generalizing examples to demonstrate theory, and then present the theory. Because of this, presenting examples is critical to the other three methods, and engages the students most effectively. Lateral and holistic thinking teaches students to stand in others' shoes, and think as they would. This forces them to consider viewpoints that they might not otherwise think of. Finally, reversing assumptions helps highlight what could happen if an assumption is wrong, and is extremely useful when discussing the ideas underlying defense in depth.

The role of these methods is to make students aware of how the superstructure of security depends upon the underlying assumptions, and how important it is for students of security to identify and challenge these assumptions. Students learn that applying these techniques lead to a more informed decision making process, and therefore more robust designs and implementations of security solutions.

Conclusion

The security of information and systems is rooted upon assumptions. Formal methods assert that the system satisfies specific security properties if preconditions (also called "assumptions") are satisfied. Less formal security analyses rely upon the assertions of policy, mechanisms, and procedures, all of which are assumptions or rely upon still

more assumptions. If the assumptions are wrong, then the proofs or arguments of security are meaningless. Thus, to determine whether a system is "secure" requires determining how correct the assumptions are, which in turn requires knowing what those assumptions are.

Students must be taught to question assumptions. Unless they know how to find those assumptions, and then challenge them, they cannot provide a realistic analysis of a system. Attackers routinely do this, so those who are learning to defend systems, or design systems that are to be difficult to break into, must also do this to uncover the weak points, and strengthen them.

This paper proposed several teaching methods to help teach students the importance of verifying and validating assumptions. These methods have been applied in other fields, and indeed in classes of computer security, with great success. These methods also broaden the students in those classes, because the students learn how assumptions constrain our thinking. By applying these ideas to other courses, they grasp the ubiquity of assumptions, and how much fun it is to challenge accepted ideas.

References

[Adams 2001] James L. Adams, Conceptual Blockbusting, Perseus Publishing, Massachusetts (2001).

[Alinsky 1972] S. Alinsky, Rules for Radicals, Random House, Inc., New York, NY (1972).

[Bishop 2004] M. Bishop, "Teaching Context in Information Security," Avoiding Fear, Uncertainty, and Doubt Through Effective Computer Security Education: Proceedings of the Sixth Workshop on Education in Computer Security pp. 29-36 (July 2004).

[CERT 1995] CERT, Sun 4.1.X Loadmodule Vulnerability, CERT Advisory CA-1995-12 (Oct. 1995). Available at http://www.cert.org/advisories/CA-1995-12.html.

[CIS2004] Center for Internet Security, Windows Server 2003 Operating System: Legacy, Enterprise, and Specialized Security Benchmark Consensus Security Settings for Domain Controllers, Version 1.1 (Oct. 2004). Available at http://www.cisecurity.com/bench_win2000.html.

[de Bono 1971] Edward de Bono, Lateral Thinking for Management, Penguin Books, Middlesex England (1971).

[Denning 1999] D. Denning, "The Limits of Formal Security Models," National Computer Systems Security Award Acceptance Speech (Oct. 1999). Available at http://www.cs.georgetown.edu/~denning/infosec/award.html.

[Foote 1958] S. Foote, The Civil War: A Narrative from Sumter to Perryville, Vol. 1, Random House, NY (1958).

[Foster 1996] J. Foster, How to Get Ideas, Berrett-Koehler Publishers, San Francisco (1996).

[ISO2000] International Standards Organization, The International Security Standard, ISO 17799 (Dec. 2000).

[Michalko 2001] M. Michalko, Cracking Creativity: The Secrets of Creative Genius, Ten Speed Press, Berkeley, CA (2001).

[Sherman & Shultz 1998] H. Sherman and R. Shultz, Open Boundaries: Creating Business Innovation Through Complexity, Perseus Publishing, Massachusetts (1998).

[Winkler 1997] I. Winkler, Corporate Espionage, Prima Publishing, Rocklin, CA (1997).