LIMITATIONS OF ENCRYPTION TO ENFORCE MANDATORY SECURITY

Morrie Gasser
MITRE Corporation
Bedford, Mass. 01730

Encryption has long been applied to protection of information in transit between computer systems, and there is little doubt that cryptosystems exist to provide any desired degree of protection for such applications. However, this confidence in the security of the cryptosystem may cause one to overlook the insecurities arising from inappropriate application of encryption.

It is occasionally suggested that encryption may have a role in the protection of information resident within a computer system—a job normally the responsibility of the operating system. We refer here to the protection of a user's information from unauthorized access by another user who also has legitimate access to the system. We seek to prevent unauthorized access by a skillful penetrator, as well as access through the use of a Trojan Horse* implanted in the system acting on behalf of the user whose information is to be compromised.

While modern operating systems today do a reasonable job of protecting files from disclosure to "casual" attackers, determined efforts will generally succeed in finding holes in any existing large-scale general-purpose system. It is quite possible that by adding encryption in various places throughout the system, and suitably distributing the keys, the determined penetrator's job can be made much harder. Whether encryption in this case provides enough added protection to warrant its added cost is a subject of debate that will not be addressed here.

The much more difficult problem is protection against the Trojan Horse or malicious program. Because the malicious program already has access to the user's files, only a "mandatory" security mechanism—one which cannot be bypassed by software with legitimate access to the information—can provide the required protection.

---

*A Trojan Horse is any program whose function is otherwise benign but which secretly performs some unauthorized action when invoked on behalf of an authorized user. For example, an editor might contain a Trojan Horse that secretly copies a user's files to another user's directory. In any system with highly sensitive information, any utility program whose origins or contents cannot be verified should be considered a candidate for a Trojan Horse.

Strict enforcement of mandatory security policies often runs counter to the goal of convenient information sharing, which is what large multiuser systems attempt to provide. If programs activated by a user cannot (accidentally or intentionally) give away information to other users, then the act of implementing sharing when it is desired may require some additional action by the user or other carefully controlled mechanism. Nevertheless, there are many cases where the value of information is such that the inconvenience and cost of implementing mandatory security is justified.

When discussing mandatory security we speak of two classes of software (or, equivalently, hardware): untrusted software, whose origin is not precisely known or controlled and may be suspected of containing Trojan Horses attempting to defeat the mandatory security controls, and trusted software, reponsible for enforcing the mandatory security controls or for bypassing the controls in some specified, trusted manner upon direct request by a user. The vast majority of utility functions performed by a system need not be trusted when mandatory controls are present. Note, of course that "trust" in this context refers to proper behavior with respect to security—clearly if users did not trust a program to perform its desired function they would not use it.

Because the intent of a mandatory security policy is to protect highly sensitive information, considerable expense to secure an operating system may be justified. While such security can, in theory, be implemented through the "usual" application of software/hardware mechanisms, confidence in the correct operation of the trusted mechanisms usually requires costly development and verification techniques, dependent on the degree of confidence desired. Because such systems are not generally available and enhancement of existing systems is not adequate, new systems must be developed. Is it possible that encryption, installed at critical places in a system, might provide a simpler mechanism at far lower cost?

In the abstract, an encryption-based mandatory protection scheme would be constructed in such a way that all outputs generated by untrusted software would involuntarily pass through an encryption box before being assigned to shared storage (figure 1). Inputs to that software might pass through the same or other boxes for decryption, although decryption could be optional and is not necessarily security-relevant. The encryption keys would be loaded under control of some trusted mechanism, determined by the user on behalf of which the untrusted software was running. Untrusted software cannot specify the key used to encrypt its output, nor can it bypass encryption of its output.

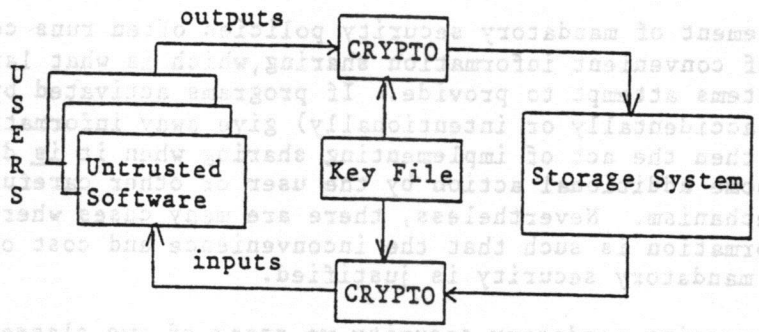

Figure 1. Simple Encryption

If the mechanism works as intended, the problem of protecting information from unauthorized access becomes a problem of key management—presumably much simpler than management of a file system and its directories. The untrusted program could not inadvertently or intentionally insert information into the storage system that could be read by a user other than the one whose encryption key was currently active. In theory, most management of the storage system could be done by untrusted mechanisms subject to the encryption controls. Only where information needs to be kept on a global basis (i.e., allocating storage between users) is a trusted mechanism required.

While there are numerous complexities that might render the above scheme impractical for a large general-purpose operating system, a small special-purpose system, with limited need to share data, may be more suitable. Clearly, if this technique cannot prevent information compromise in a small system, the technique is even more useless for a larger system. Consider the application in figure 1 where the storage system is a large random access memory, and the "users" are processors that can access that memory in an arbitrary fashion. Suppose we simplify the trusted mechanism to the point where the processors operate independently, and the only security mechanism is one that passes a given processor's outputs (data only—not addresses) to memory through the encryption box with a key unique to that processor. There is no need for a trusted memory management mechanism—through appropriate conventions each processor knows where it can read or write. In order to make the system useful (i.e., to share information where necessary), we distribute appropriate decryption keys to the various processors so that a given processor may be able to read another processor's output as required.

Thus we have a static mandatory security policy, implemented through appropriate key distribution, that defines which processors can communicate in which directions. Of course, this architecture may be too simple to be useful (or too expensive to be practical), and numerous issues are not addressed, but the intent here is to provide a "best case" example to prove a point.

A serious problem immediately surfaces regarding the encryption of small units of data (e.g., words) in a random access memory. We can assume this problem is solved by suitable hardware that forces data to be written in some minimum block size, possibly including some random bits so that repeated writing of the same block under the same key will result in different ciphertext for the block. Other techniques may be available, with varying degrees of overhead in memory utilization, but the point is that we are assuming that the cryptosystem in its usage here is secure from cryptanalytic attacks.

We now come to the real problem. While we allow a given processor A to select any block in memory for reading, presumably only those blocks written by processors for which A has the key can be intelligibly deciphered. If B is not allowed to communicate with A, ciphertext written by B will be nonsense to A. However, while the ciphertext itself may be nonsense, the very fact that it is there, or the fact that it does not change on two successive reads, is information that untrusted software in B can communicate to A. One penetration might involve A's writing a block and waiting for B to overwrite it—another could involve A's waiting for B to change a block previously written by B. In fact, through the use of two such blocks, one for synchronization, B can communicate with A at the rate of one bit for every pair of blocks written.

This communication channel can only be closed by providing more explicit allocation of memory using a trusted mechanism, so that there is no way for memory written by B to be accessed (successfully or unsuccessfully) by A. As long as there is any way for B to affect the result of an operation performed by A, protection from the Trojan Horse has not been achieved. Unfortunately, any mechanism that controls memory allocation obviates the need for encryption. On larger systems, where the memory allocation must of necessity be far more complex, and where memory is but one of the means by which processors or processes might communicate, the tradeoff becomes worse. Furthermore, large systems are replete with "control channels" which are not encryptable yet allow covert communication.

The crucial point is that properly enforcing mandatory security requires the elimination of all deterministic cause-and-effect relationships (or information flow) between subjects unauthorized to

communicate. Encryption within a system simply does not address this type of security. Encryption properly applied works on communications links because the relationship is relatively nondeterministic (to those not possessing the key)—information is constantly changing, even independent of whether real data is being transmitted. For certain applications using benign software encryption within a computer system may be appropriate, because exploitation of encryption-based controls requires a Trojan Horse. But for the majority of large systems processing sensitive data, whose software origins are not controlled (including the software provided by the manufacturer), there is nothing encryption can do over that of a trusted security mechanism.

In conclusion, when considering encryption for applications other than serial communications links, we must not lose sight of the threats against which we are trying to protect. In benign, but not necessarily foolproof or bug-free, environments, encryption may have a role. Where the threat is one of malicious software's access to sensitive data, encryption is hard-pressed to provide any additional security.