Position Paper: Prolepsis on The Problem of Trojan-Horse-Based Integrity Attacks

J. McDermott
Department of Computer Science
James Madison University, Harrisonburg, VA 22807, USA
mcdermot@cs.jmu.edu

1. ABSTRACT

The problem of integrity attacks via Trojan horse applications is so difficult that some computer security researchers and practitioners may object to it as an unreasonable research topic. We agree that the problem is difficult, but we argue that it is reasonable to consider it. We argue that the problem of application-based Trojan horses per se has not been solved; that previous integrity approaches do not offer significant protection in today's architectures and third, solutions that offer significant protection are available. Some of these solutions have been researched, but others have not. We invite other researchers to investigate the problem.

1.1 Keywords

Trojan horse, security, integrity, storage

2. INTRODUCTION

Surreptitious destruction of stored data [4] is a particularly troublesome kind of integrity attack that seeks to disrupt the real-world operations of an organization. The motive for the attack is not fraud, but competitive advantage or simply malice. The most effective means of attack is a Trojan horse. This presents a significant problem for conventional security mechanisms and traditional approaches to data integrity protection. The problem of integrity attacks via Trojan horse applications is so difficult that some computer security researchers and practitioners may object to it as unreasonable. We agree that the problem is difficult, but we argue that it is reasonable to consider it. First of all, the problem of application-based Trojan horses per se has not been solved, so the Trojan horses will be present. Second, previous integrity approaches do not offer significant protection in today's architectures and third, solutions that offer significant protection are available.

We take the Trojan horse problem as a given. No general method of preventing Trojan horses has been established. Security standards like the Orange Book sometimes imply one, but they lack some clearly necessary techniques, such as recursive application of the standards to the programming systems used to generate the trusted code. Most security researchers could come up with a reasonable collection of formal methods, database techniques (i.e. version control), software engineering processes, and management controls that would arguably minimize the chances of picking up a Trojan horse. Unfortunately, current and future market forces rule out even these kinds of approximate implementations. The current trend is toward rapidly developed special purpose applications based on low-cost shrink-wrapped general purpose software [2]. Neither the applications nor the underlying general purpose software are developed with more than a minimal amount of engineering. The best we can hope for in today's market is approximately Trojan horse free implementations of security specific components.

It is difficult to argue that these kinds of attacks are not likely. The National Computer Security Center writes [8]

> "There are many systems in which integrity may be deemed more important than confidentiality (e.g., educational record systems, flight-reservation systems, medical records systems, financial systems, insurance systems, personnel systems.) While it is important in many cases that the confidentiality of information in these types of systems be preserved, it is of crucial importance that this information not be tampered with or modified in unauthorized ways. Also included in this categorization of systems are embedded computer systems. These systems are components incorporated to perform one or more specific (usually control) functions within a larger system. They present a more unique aspect of the importance of integrity as they may often have little or no human interface to aid in providing for correct systems operation."

Since it is difficult to prevent the introduction of Trojan horses and there are systems whose integrity reasonably could be attacked, we should consider how well existing integrity models work. The key limitation is that integrity solutions must work in a climate of rapidly developed spe-

1998 NSPW 9/98 Charlottsville, VA, USA
1-58113-168-2/99/0007...

cial purpose applications based on low-cost shrink-wrapped general purpose software.

3. PREVIOUS INTEGRITY MODELS DO NOT WORK

The concept of tampering[1] with data in transmission is well-understood. Many defenses, both practical and theoretical, have been proposed and some are in use. However, information systems not only transmit data, but also transform and store it. The differences between transmission, transformation, and storage make both the attack and the defense fundamentally different. When we protect transmitted data, we assume that we start with a correct copy of the data. Data that originates from a transformation may not be correct and we may have no easy way to check it. Data is stored for protracted periods of time, so it is difficult to apply the concepts of time, freshness, or session to ensure its authenticity. Furthermore the persistence of stored data restricts the number of bits we can use for overhead.

Data storage in database systems and similar repositories is protected against accidental introduction of bad values by various integrity constraints. Integrity constraints are relatively successful when used this way. They work because they are only related to the bad values in a statistical way. When bad data is introduced by tampering, it can match the integrity constraint up to an arbitrary number of bits, less the one bit needed to make the value invalid. In fact, the only integrity constraints that will detect tampering in general are identity relations. While identity relations may be mathematically simple, they are truly difficult to impose on real-world data, if they can be imposed at all. For example, a check sum and its corresponding value define an integrity constraint that is an equivalence relation. Unfortunately, we cannot use check sums, because they are computed internally. Check sums are not an integrity constraint on real-world data, but on internal representation. Since a Trojan horse has access to both the computation of check sums and the internal representation of data, it can bypass check sums (and digital signatures).

Access controls do not provide protection against integrity attacks. The reasons for this is that access controls block Trojan horses from sharing data, but not from accessing it at its source or sink. As an example, take the Clark-Wilson model. A Trojan horse can either be embedded inside a transformation procedure (TP) or an integrity verification procedure (IVP). A Trojan horse in a transformation procedure can delete values, insert new bogus values, or replace valid updates with their before images. Even if all of the IVP's are free of Trojan horses, they may never detect a bogus value inserted by a Trojan TP. This is because their integrity specifications have the same limitations as integrity constraints: they must be identity relations defined on the real world. A Trojan horse IVP has even more freedom to tamper without detection. If it restricts its tampering to the constrained data items that it is responsible for checking, then its bogus changes will never be detected.

Some readers will object that Clark-Wilson TP's and IVP's are Trojan-horse free, by definition. Clark-Wilson and other models assume that systems of Trojan-horse-free components can be constructed. However, even an approximate implementation of a Trojan-horse-free system is not possible in today's market. What is missing from Clark-Wilson and other integrity models is an assumption that only a handful of system components may be assumed to be relatively Trojan horse free.

Another assumption that does not hold in today's climate is the notion that data with different integrity "levels" (i.e. requirements) will be processed by different programs. This assumption is part of the Clark-Wilson model, but is most apparent in the familiar Biba integrity model. In current systems (and many legacy systems) data processing almost always involves a few large tightly integrated subsystems, for data of all integrity "levels." Differences in processing are handled by rapidly-developed special purpose logic such as scripts, macros, client programs, or other "front-end" mechanisms. The effect is that access control classes do not separate the writing of different classes of data, because all data is written by the same programs.

An integrity model that does work, in principle, is the Byzantine generals problem [10]. Although the original idea was to model a faulty low-level process as a traitor, the model also describes Trojan horses quite nicely. A Byzantine agreement protocol can prevent Trojan horses from tampering with data. Unfortunately, Byzantine generals solutions do not scale well. Byzantine generals problems are usually phrased in terms of agreement on a single atomic value, and often the value must be of a fairly simple type such as Boolean or integer. Cryptographic authentication, non-repudiation, and communication integrity protocols must added to enforce the necessary assumptions. If not, then $3t+1$ copies of the system are needed to detect t Trojan horses. These requirements, and the need for multi-round voting algorithms, make it impractical to apply a Byzantine generals solution to a large high-level components such as database systems.

Since existing models and mechanisms do not work in the present environment, an alternative approach is needed. Our suggestion has been to simulate the presence of an oracle that can predict what values should be recorded. There are several ways to simulate this, not all of which have been investigated.

4. EFFECTIVE SOLUTIONS ARE AVAILABLE

One of the most effective techniques for simulating an oracle is to replicate the data over distinct systems. Ammann, Jajodia, et al. [1] have shown that the kind of replication commonly used in general purpose database systems is not effective. However, McDermott and Froscher have shown how n replicas can defend against n-1 Trojan horses [5].

1. We use the word *tamper* to mean intentional introduction of incorrect data.

NRL has also developed a proprietary defense that allows two copies to defend against n Trojan horses [7]. A platform-specific replication-based defense has been prototyped by McDermott, Gelinas, and Ornstein [6].

Replication in general is problematic in an information warfare context. Under many commercial transaction processing approaches, bogus data can be replicated automatically and precisely to many locations. However replication works as a defense if we use one-copy serializable logical replication over distinct database systems.

4.1 Logical Replication

Many commercial replication mechanisms copy data values from the source data item to its replicas and others only copy the command after an update takes place. However, logical replication copies the command that caused the source data item to change. The command is executed at each replica's site and, because of one-copy serializability, results in the same new value for the replica. If we assume a distinct provenance[1] for the software at each site, then the Trojan horse will not be replicated at all sites. An attack must compromise multiple, possibly heterogeneous, host programs, an unlikely event in practical systems. Even if the attackers can succeed at every site, the attack still may fail. If the Trojan horses are not able to deliberately malfunction in a one-copy serializable fashion, their bogus values will diverge. This can be ensured by restricting communication between the sites to just the protocols needed to carry out the authorized replication. So we can expect a scheme using n replicas to detect up to n-1 cooperating Trojan horses and possibly detect an n-Trojan horse attack.

Detection is simple in the replication defense. There is a detection process at each source or replica site. Following changes to protected data, the process at the source site computes a check sum over the changed data and sends it to each replica site, along with the identification of the change. After the logical update is performed at a replica site, the detection process at the replica site compares its results with the results of the primary site. If there is disagreement, there is a problem. Check sums are not essential to the approach and are merely used to facilitate efficient comparison. The granularity of the comparisons or checks is a trade-off between speed and storage. Comparisons over individual data items allow quicker response to attacks but take more storage to perform. We also do not need to check every change, since the insertion of bogus data at some sites will ultimately diverge the copies.

The use of logical replication may allow us to disconnect compromised systems until the Trojan horse can be disabled. If an uncompromised site can act as a data source, it can take over from a compromised source. Replica sites that do not originate data are also easily disconnected.

1. The development, administration, and maintenance of the software is done by distinct sets of people.

4.2 Other Solutions

Logical replication is not the only approach to simulating an oracle that checks stored or transformed data. We will describe two here that have never been proposed or investigated: *session replay* and *pre- and post-condition checks*.

In the session replay technique, only certain user-selected sessions are protected against Trojan-horse-based integrity attacks. The inputs to these sessions are recorded and then replayed on a second system to check the results. The distinction between this approach and logical replication is that session replay provides temporary logical replication of a subset of an application. This conserves resources at the expense of additional complexity. The limitations of this approach are its complexity and its coverage of only selected sessions.

In the pre- and post-condition check approach, only certain user-selected sessions are protected against attacks. A snapshot of the data is taken before the session. A second snapshot is taken at the end of the session. The two snapshots are compared and the differences are mapped back to commands of the session. If a difference fails to map to a command, then some form of attack may have taken place. This technique has the same advantages and limitations as the session replay technique.

Our point here is not to present the techniques, but to establish the existence of solutions, either researched or not investigated.

5. CONCLUSION

The problem of Trojan-horse-based integrity attacks should not be dismissed as too difficult. Trojan horses are unavoidable at present and it is easy to use them to tamper with stored data. Existing security mechanisms based on current paradigms do not provide protection from these kinds of attacks, in the kinds of systems that are built today. On the other hand, there are feasible ways to protect current architectures against these attacks, so it is a worthwhile problem to investigate. We hope that other researchers will become interested in this problem; there may be other ways to solve it.

5.1 REFERENCES

1. Ammann, P., Jajodia, S. McCollum, C. and Blaustine, B. Surviving information warfare attacks on databases. *Proc. of the IEEE Symposium on Research in Security and Privacy*, Oakland, CA, USA, May, 1997.

2. Cusumano, M. and Selby, R. How Microsoft builds software. *CACM*, 40, 6, June 1997, pp. 53-61.

3. Department of Defense. *Department of Defense Trusted Computer System Evaluation Criteria*, DoD-52 00.28-STD, December, 1985.

4. McDermott, J. and Goldschlag, D. Storage jamming. In *Database Security IX: Status and Prospects* (D. Spooner, S. Demurjian, and J. Dobson, eds.), 365-381. Chapman and Hall, 1996.

5. McDermott, J. and Froscher, J. Practical defenses against storage jamming. *Proc. of the 20th National Information Systems Security Conference*, Baltimore, MD, USA, October 1997.

6. McDermott, J. Gelinas, R. and Ornstein, S. Doc, Wyatt, and Virgil: Prototyping storage jamming defenses. *Proc. Annual Computer Security Applications Conference*, San Diego, CA, USA, December 1997.

7. McDermott, J. and Gelinas, R. Record and Disclosure of Invention 5540-092:JPM:jms dated 3 March 1998.

8. National Computer Security Center. *Integrity in Automated Information Systems*, C Technical Report 79-91, September 1991.

9. National Computer Security Center. *Integrity-Oriented Control Objectives: Proposed Revisions to the Trusted Computer System Evaluation Criteria.* C Technical Report 111-91, October 1991.

10. Silberschatz, A. and Galvin, P. *Operating System Concepts*, 4th e.d., Addison-Wesley, 1995, ISBN 0-201-50480-4.