The Application of Model Checking for Securing E-Commerce Transactions

Motivation for Model Checking

As firms become progressively more dependent on Internet-based information systems, they are increasingly vulnerable to defects in those systems. These defects can lead to errors, undetected fraud, and malicious intrusion. Information system errors can be catastrophic, whether they occur in market transactions, banking, air traffic control, and so forth. Resulting damages can include lost revenue, lost data, lost trust, and increased costs [1].

Consequently, effective design of e-business processes is essential to avoid defects that can otherwise lead to errors, fraud, and intrusion. Whereas carefully designed e-business protocols can perform effectively within most expected situations, assuring correct processing under all circumstances is exceptionally complex and difficult. Hidden flaws and errors that occur only under unexpected and hard-to-anticipate circumstances can lead to processing errors and potentially ruinous failures. Continued growth of e-business will, in large part, depend on protocols that are designed to ensure that the information exchanged between trading parties is protected from unauthorized disclosure and modification [2].

Verifying that an e-business protocol is robust against hidden flaws and errors is a formidable task. Manual methods are slow and error prone. Even theorem provers, which provide a formal structure for verifying protocol characteristics, may require human intervention and can be time consuming. Moreover, theorem provers generally do not provide much help in locating failure sources. Finally, simulations offer computational power, but they are ad hoc in nature and there is no guarantee they will explore all important contingencies [1].
Conversely, model checking is an evolving technology that offers a platform for effective and efficient evaluation of e-business protocols. Current model checking technology is based on automated techniques that are considerably faster and more robust than other approaches such as simulation or theorem proving. With today’s model checkers, large state spaces can be analyzed in minutes. Additionally, model checkers are able to extend their analysis by supplying counterexamples that identify the precise location where a protocol failure is discovered [3] — see Table 1.

---Insert Table 1 here---

Although still relatively new to the analysis of e-business processes [1] [4], model checking has an established record in the practical analysis of complex hardware and software processes [5] [6]. Elsewhere, we have provided more technical presentations of applications of model checking to e-business processes [7, 8]. The purpose of this paper is to provide an exposition of model checking in the context of e-business that is accessible to managers, auditors, and system developers. We believe that this audience can benefit from an accessible introduction to model checking with a discussion of its potential for enhancing the development of reliable e-processes. The e-processes on which we base this discussion is from the work of Ray and Ray [3], which incorporates processes fundamental to a broad class of e-business operations. These processes include distributed processing, parallelism, concurrency, communication uncertainties, and continuous operations.

**Related Work on Model Checking**

E-business managers, developers, and auditors require robust tools to assure users that e-business systems are secure and reliable [3]. Designing and implementing highly secure and reliable e-processes is challenging, and requires adherence to several specific criteria to be
effective. Adding to the challenge of designing effective e-business protocols, recent research demonstrates that money atomicity, goods atomicity, and validated receipt are critical e-process requirements [3].

Money atomicity ensures that money is neither created nor destroyed in the course of an e-business transaction. Goods atomicity guarantees that a seller receives payment only if the customer receives the product. Validated receipt ensures that the buyer is able to verify the contents of the product about to be received before making payment [3].

Heintze et al. [4] use a model checker to examine the non-security characteristics of e-processes to verify the money and goods atomicity properties of two e-business processes—Digicash [9] and NetBill [10]. This seminal work demonstrates how to model e-business processes and their properties of interest in a process algebra language, CSP, which is the language used in FDR. The model checker determined that the NetBill process does achieve money atomicity and goods atomicity. Conversely, the model checker discovered that Digicash failed to ensure money atomicity—an interesting finding in a commercial software package. In the latter case, a detailed counterexample was generated to illustrate a set of actions leading to a state where money atomicity was not realized.

Wang et al. [1] make a persuasive case that model checking can play a valuable role in designing e-processes and can be an effective method for evaluating and auditing existing e-processes. The authors use a ticket sales application to demonstrate the capabilities of two model checkers. While their application is conceptually sound, it abstracts the fundamental characteristics of many e-business systems—including distributed processing, parallelism, concurrency, communication uncertainties, and nonstop operations. The authors emphasize that
such properties are common to more complex applications, such as online stock trading and retailing.

Ray and Ray [3] extend the basic structures examined by Heintze et al. [4], who assume that neither the NetBill server nor the communication links to the NetBill server ever fail. Ray and Ray provide a more comprehensive treatment of system and communication failures by allowing a communication link among a customer, a merchant, or a trusted third party to fail arbitrarily. Additional mechanisms were developed that ensure that desirable properties are preserved despite such failures. The result is a realistic and practical platform for many e-business applications.

An e-Business Protocol Example

Figure 1 outlines a possible e-business protocol [3]. Messages are exchanged between a customer, a merchant, and a trusted third party (TTP). A merchant has several products to sell. The merchant places a description of each digital product in an online catalog service with a TTP along with a copy of the encrypted product. When a customer finds a product of interest in the catalog, the customer downloads the encrypted product and sends a purchase order (PO) to the merchant. The customer cannot use the product until it is decrypted, and the merchant does not send the decrypting key unless the merchant receives a payment token through the PO process. The customer, in turn, does not pay unless he/she is sure that the correct and complete product has been received. The TTP provides anonymous support for PO validation, payment token approval, and approval of the overall transaction between the customer and the merchant.

---Insert Figure 1 here---

At a more detailed level, the process begins as the customer browses the product catalog located at the TTP and chooses a product. The customer then downloads the encrypted product
along with the product identifier. The product identifier is a file that contains information about
the product, such as its description and its identifier. If the identifier of the encrypted product file
corresponds to the identifier in the product identifier file, the transaction proceeds. If the
identifiers do not match an advice is sent to the TTP and the customer waits for the correct
encrypted product. This process ensures that the customer receives the product that was
requested from the catalog. Next, the customer prepares a PO containing the customer’s identity,
the merchant identifier, the product identifier, and the product price. A cryptographic checksum
is also prepared. The PO along with the cryptographic checksum is then sent to the merchant.
The combination of the PO and cryptographic checksum allows the merchant to ascertain
whether the PO received is complete or whether it was altered while in transit. Upon receipt of
the PO, the merchant examines its contents. If the merchant is satisfied with the PO, the
merchant endorses the PO and digitally signs the cryptographic checksum of the endorsed PO.
This is forwarded to the TTP. The TTP is involved in the process to prevent the merchant from
later rejecting the terms and conditions of the transaction. The merchant also sends a single use
decrypting key for the product to the TTP. The merchant next sends a copy of the encrypted
product to the customer, together with a signed cryptographic checksum. The signed
cryptographic checksum establishes origin of the product and also provides a check to signify
whether the product has been corrupted during transit.

Upon receipt of the second copy of the encrypted product, the customer validates that the
first and second copies of the product are identical. Through this process, customers can be
assured that they received the product ordered. The customer then requests the decrypting key
from the TTP. To do this, the customer forwards to the TTP the PO and a signed payment token,
together with its cryptographic checksum. The payment token contains the customer’s identity,
the identity of the customer’s financial institution, the customer’s bank account number with the financial institution, and the amount to be debited from the customer’s account.

To verify the transaction, the TTP first compares the digest included in PO from the customer, with the digest of the same from the merchant. If the two do not match, the TTP aborts the transaction. Otherwise the TTP proceeds by validating the payment token with the customer’s financial institution by presenting the token and the sale price. The financial institution validates the token. If the token is not validated, the TTP aborts the transaction and advises the merchant accordingly. If the token is validated, the TTP sends the decrypting key to the customer and the payment token to the merchant, both digitally signed with the TTP’s private key.

Secure channels guarantee the confidentiality of all messages throughout this protocol. The protocol ensures money atomicity if the payment token generated by the customer contains the amount to be debited from the customer’s account and credited to the merchants account. Consequently, no money is created or destroyed in the system by this protocol.

Goods atomicity is guaranteed if the TTP hands over the payment token only when the customer acknowledges the receipt of the product. The process also ensures that the product is actually available to the customer for use—only when the customer gives the go-ahead for payment by acknowledging the receipt of the good.

Delivery verification is guaranteed if the TTP receives a cryptographic checksum of the product from the merchant. Also, the customer independently generates a checksum of the product received and sends it to the TTP. Using these two copies of the checksums, which are available at the TTP, both the merchant and the consumer demonstrate proof of the contents of the delivered goods.
Example of Model Checking

The e-commerce system described in the preceding section can be modeled and tested in a model checker, such as FDR. FDR has been used in a variety of hardware and software applications, and has been successfully applied to testing for atomicity in two commercial e-commerce payment systems, as noted above [1]. Recall that three of the most important requirements of e-commerce trading are that:

1. Money is neither created nor destroyed in the course of an e-commerce transaction. A transaction should ensure the transfer of funds from one party to another without the possibility of the creation or destruction of money. No viable e-commerce payment method can exist without supporting this property [11].

2. Both the customer and merchant should receive evidence that the goods sent (or received) are those to which both parties agreed. Particularly when dealing with goods that can be transferred electronically (as we do here), the combination both (1) and (2) is essential [12].

3. The customer is able to verify the contents of the product about to be received before making payment. The customer must be able to verify that the product about to be received from the merchant is the same as the product that was ordered, before the customer pays for the product.

Model checking provides verification of whether or not our system guarantees these requirements by writing them as specifications in the model checker. If a specification is not satisfied, the model checker returns a counterexample, which allows the analyst to immediately see how the requirement can be violated. This, in turn, aids in modifying the system to resolve the problem. As an example, we consider a specification for requirement (1) above.
**Specification Requirement_1**

IF NOT transaction_trace = STOP
OR
{customer sends paymentToken to TTP;
 merchant receives paymentToken from TTP;
 STOP;}
OR
{customer sends paymentToken to TTP;
 customer receives transactionAborted message from TTP;
 STOP;}
THEN
Requirement_1 is violated;
End requirement_1;

The above specification asserts the following:

1. The customer must send payment to the TTP and the merchant must subsequently receive that payment from the TTP, or

2. The customer can send payment to the TTP and (in the case where the payment is invalid) the customer then receives a transaction aborted message from the TTP.

3. If neither (1) nor (2) is satisfied, the requirement is violated.

The following is an example of what the model checker returns as a counterexample when the implementation fails to satisfy specification requirement_1:

1. The customer receives the encrypted goods from the TTP, and then sends a PO to the merchant.

2. The merchant then sends the encrypted goods to the customer and sends a key to the TTP, after which the encrypted goods are received by the customer who sends a payment token to the TTP.

3. The TTP then receives the key from the merchant and the payment token from the customer.
4. The TTP then sends the payment token to the merchant, after which the customer receives the key from the TTP. The process then stops. Since the next step should have been the receipt of the payment token by the merchant (from the TTP), we know where to look to examine the failure.

In the above case, the model checker showed that the failure occurred in the e-process controlling the sending of an encryption key from the merchant to the TTP. Once this was identified, the problem was rectified and proven in a subsequent run of the model checker.

**Concluding Remarks**

As e-commerce transactions become increasingly complex, coupled with increased regulations and liability exposure, the need for assurance in e-commerce protocols will likely grow. A potentially valuable development would be to adapt model checking to dependability auditing of e-business processes prior to implementation. The assurance afforded by model-checking can justify greater confidence in e-processes and thereby increase the acceptance and legitimacy of e-business.

**References**


