Covert channels detection in protocols using scenarios

Loïc Hélouët[1], Claude Jard[2], Marc Zeitoun[3]
[1] Irisa/INRIA, Campus de Beaulieu, F-35042 Rennes, France
[2] Irisa/ENS Cachan-Bretagne, Campus de Ker-Lann,F-35170 Bruz, France
[3] LIAFA, Université Paris 7, 2 place Jussieu, F-75251 Paris, France
{ loic.helouet@irisa.fr, marc.zeitoun@liafa.jussieu.fr, claude.jard@irisa.fr }

Abstract: This paper presents an approach to detect illegal information flows from requirements expressed as high-level scenarios.

1 Introduction

The term *covert channel* has been first introduced in [10] to designate an information flow that violates the security policy of a system. Covert channels are considered as a threat to information systems, as they can be used to provide information leaks, synchronize attacks, or divert a system from its initial use. Their detection is considered as an important task [7, 14] that should be automated as much as possible, in order to be reproducible.

The literature distinguishes several kinds of covert channels: storage channels use a resource of a system (variable, file, ...) to store data that can be read by an unauthorized third party. Timing channels modulate the response time of a system in a noticeable way to transmit information.

It is generally admitted that covert channels cannot be completely eliminated [12, 13]. In fact, closing all covert channels in an information system would require to remove semaphores, shared resources, dynamic adaptation of resource allocation, and even suppress all internal clocks! However, detecting covert information flows and computing their bandwidth remains an essential task. Depending on the bandwidth of a channel, several solutions can be proposed: suppress the resources used (which is not always possible), try to add noise to the covert channel in order to limit its bandwidth, or add monitors to detect illegal uses of a system. Even if a covert channel has a low bandwidth or cannot be closed, [14] recommends to document it with scenarios of use.

Several automatic techniques have been proposed to detect covert channels in information systems [2, 9]. They are based on an abstract representation of a system by a model. However, the difference between a model and an implementation as well as the assumptions made during the analysis may cause some potential channels to be irrealistic in a running environment. This points out the need for testing a covert channel on an implementation when it is discovered. Again, it is important to provide scenarios of use for a potential covert channel. Conversely, model-based approaches can miss some implementation details that can be used to transfer information.

For distributed protocols, the studies performed so far are more empirical, and mainly consist in detecting how information can be hidden in protocol frames [1, 15]. However, the study of protocol frames is not sufficient, as functionalities of the protocol involving several frames can be used to encode and transmit information. The approach proposed in this paper is to perform covert channels analysis for distributed systems at the requirement level, when design decision can still be taken to reduce the bandwidth of a channel at reasonable cost. The research is based on a representation of requirements by scenarios. Covert channels detected during the requirement phase are likely to be present in any implementation of these requirements. They are not due to security holes of an implementation: they represent *structural* information flows. Another advantage in using scenarios is that the model used immediately provides (in an intuitive manner) the scenarios needed to document and test a potential covert channel.

The paper is organized as follows: section 2 describes our scenario model, section 3 shows how covert channels can be detected on a scenario model, and section 4 gives perspectives and concludes this work.

2 Scenarios

Scenario languages have known a growing interest the last decade. They are mainly used to represent behaviors of distributed systems at an abstract level, or to capture requirements in early development stages. Even if scenarios are rather incomplete, they already contain enough information to perform automatic analysis of properties that may already appear at the requirement level. The main idea developed hereafter is that scenarios can be used to detect illegal information flows that are consequences of the design choices at the requirement stage. As scenarios are supposed to represent typical uses of a system, if an illegal information flow is detected at this level, the same flow is likely to appear in an implementation exhibiting the same behavior. Several scenario languages exist (Live sequence Charts, UML sequence diagrams, ...). We shall focus in this paper on Message Sequence Charts [8].

Roughly speaking, a basic Message Sequence Chart (*bMSC* for short) is a graphical representation of interactions in a system, where instances participating are represented by vertical lines, and (asynchronous) message exchanges are represented by arrows from the emitting instances to receiving ones. Formally, a *bMSC* is a tuple $M = \langle E, \leq, A, I, \alpha, \phi, m\rangle$, where E is a set of events, $\leq$ is a causal partial order (events are sequentially ordered along instances axes and a message emission precedes the corresponding message reception), A is a set of action labels, I is a set of instances, α is a function associating an action name to each event , ϕ is a function associating an instance to each event, and m is a mapping that pairs message emissions and receptions.

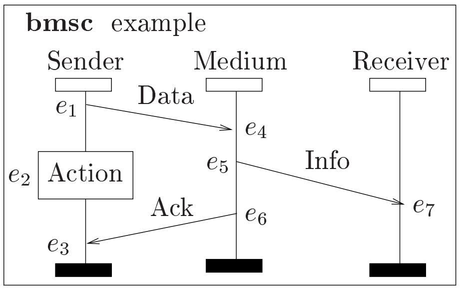

Figure 1: An example of bMSC

The *predecessors* of a set of events X in M is $\downarrow_M(X) = \{e \in E \mid \exists x \in X, e \leq x\}$. The set of *minimal events* $Min(M)$ for a bMSC M is the set of events having a single predecessor: $Min(M) = \big\{e \in E \mid \downarrow_M(\{e\}) = \{e\}\big\}$. A *projection* of a bMSC on an instance $i \in I$ is a sequence of events $\pi_i = e_1.e_2 \ldots e_p$ such that $\{e_1, e_2, \ldots, e_p\} = \phi^{-1}(i)$ and $e_1 < e_2 < \cdots < e_p$. Figure 1 represents a bMSC with seven events. Its unique minimal event is e_1 (emission of the message Data). Its projection on instance Sender is the event sequence $\pi_{sender} = e_1.e_2.e_3$.

Of course, bMSC alone are not powerful enough to represent interesting behaviors. Hence, there is a need for composition operators such as sequence, choice, loop... The *sequential composition* of two message sequence charts M_1 and M_2 is the bMSC $M_1 \circ M_2 = \langle E_1 \uplus E_2, \leq_{1\circ 2}, A_1 \cup A_2, I_1 \cup I_2, \alpha_1 \cup \alpha_2, \phi_1 \cup \phi_2, m_1 \cup m_2\rangle$, where $\leq_{1\circ 2} = \big(\{\leq_1 \uplus \leq_2 \cup \{(e_1, e_2) \in E_1 \times E_2 \mid \phi(e_1) = \phi(e_2)\}\big)^*$. Thus, the sequential composition roughly consists in merging diagrams along common instances and does not impose any synchronization between participating instances. Figure 2 gives an example of sequential composition.

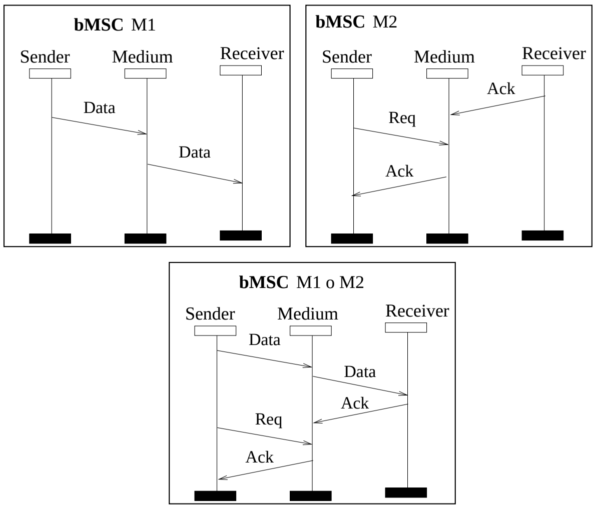

Figure 2: Sequential composition

A *high level message sequence chart* (HMSC for short) is a graph $H = \langle N, \longrightarrow, n_0, \mathcal{M}\rangle$, where N is a set of nodes, $\mathcal{M}$ is a set of bMSCs, n_0 is an initial node, and $\longrightarrow \subseteq N \times \mathcal{M} \times N$ is a transition relation. A *path* of a HMSC is a sequence of transitions $p = (n_1, M_1, n_2).(n_2, M_2, n_3) \ldots (n_{k-1}, M_{k-1}, n_k)$ such that the goal of the i^{th} transition is also the origin of the $i+1^{th}$ one, for all $i \in 1..k-1$. A *circuit* in a HMSC is a path $p = t_1 \ldots t_k$ such that the origin of t_1 and the goal of t_k are the same node. The *order* associated to a path $p = (n_1, M_1, n_2) \ldots (n_{k-1}, M_{k-1}, n_k)$ is the bMSC $O_p = M_1 \circ \cdots \circ M_k$. Figure 3 shows an example of HMSC, depicting a simple communication protocol.

A *choice node* of a HMSC H is a node with more than one successor. A choice node c in a HMSC is *local* iff all paths leaving c have a single minimal event, always located on the same instance: $\exists! i \in I, \forall p = c.n_1 \ldots n_k$, $\phi(min(O_p)) = \{i\}$. We say that the local choice node c is *controlled* by instance i.

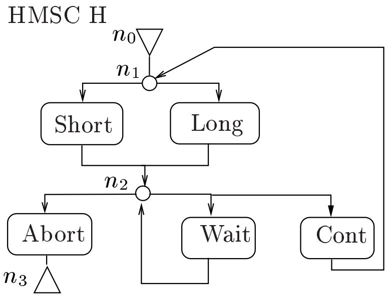

Figure 3: An example of HMSC

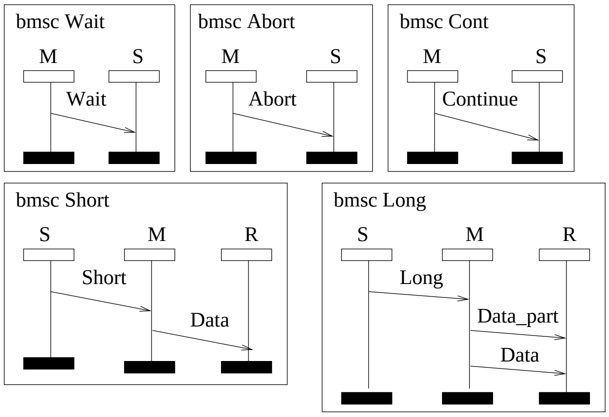

Figure 4: Basic MSCs

3 Covert channels detection

We want to detect illegal information flows in a distributed system. This first supposes that we know authorized information flows. In a first step, this can be given by a Bell & LaPadulla model [3, 4], but we think that scenarios can provide more accurate means for indicating how legal information can be transferred from one instance to another (however this is still ongoing research).

We have to make several assertions to check if illegal information can flow from a sender S to a receiver R. The first assumption is that S and R agree on a protocol for sending and receiving covert information. Covert messages can be of arbitrary length, and we suppose that the same functionality of the diverted protocol is used an arbitrary number of times. This leads to the immediate conclusion that structural information flows are tightly linked to loops in HMSCs.

We say that there is a *potential information flow* from S to R using a choice node c if

- there is a set Q_c of simple circuits from c to c such that for all $p \in Q_c, \pi_R(O_p) \neq \epsilon$ (where ϵ is the empty word);
- c is controlled by S;
- all choice nodes that can enforce a path to leave Q_c are either controlled by S or by R. Formally, for all $q = (n_1, M_1, n_2) \dots (n_k, M_k, n_1) \in Q_c$, if there is a node n_i, $i \in 1..k$, which is not controlled by S nor by R, then for any $\longrightarrow$-transition (n_i, M, n'_i), the path $(n_1, M_1, n_2) \dots (n_{i-1}, M_{i-1}, n_i).(n_i, M, n'_i)$ is a prefix of some path of Q_c.

Notice that Q_c does not need to include *all* simple circuits from c to itself. Note also that a choice node n can be controlled by another instance than S or R as long as the decision taken does not prevent from eventually getting back to node c.

Transmitting information through paths that are not controlled by S and R is not always reliable as the covert message can be delayed an arbitrary amount of time, or even interrupted. Consider the HMSC of Figure 3 and the bMSCs of Figure 4. These two figures represent a simple data transmission protocol. A sender can send short data packets to a medium that forwards them to a receiver, or long data packets that are split before being sent. After data transmission, the medium can allow another transmission, become unavailable for a period or even abort the transmission. Obviously, a sender can modulate its use of long and short packets to encode 0 and 1. Node n_1 is controlled by S, but any circuit from n_1 to n_1 passes through node n_2 where instance M can decide to abort the data transmission, or to delay the transmission for an unbounded duration. Therefore, a continuous information flow is not always guaranteed. If we replace all bMSCs by those of Figure 5 (where all nodes are controlled by S), then instances S and R can force the protocol to stay in the scenarios defined by the paths $(n_1, Short, n_2).(n_2, Cont, n_1)$ and $(n_1, Long, n_2).(n_2, Cont, n_1)$. However, the rest of the paper shows that this condition is not sufficient to ensure that information can be transmitted in this way.

Transferring information through system's behavior is one thing, but the information sent must be decodable. Information encoding is done by selecting different decisions performed by S at a choice node. The message received by instance R is the sequence of events labels observed on R. Decoding can be performed if and only if one can find a function mapping the messages received by R to a sequence of integers (the choices of S). A first sufficient condition is to require the words read by instance R to form a code [6]. However, this condition is not always necessary: decoding can be performed, in more cases, by a transducer producing an integer sequence from received messages.

A *transducer* is a tuple $\mathcal{T} = \langle S, \Sigma_1, \Sigma_2, T, S_+, s_0 \rangle$ where S is a set of states, Σ_1 is an input alphabet, Σ_2 is an output alphabet, S_+ is a set of accepting states, s_0 is an initial state, and $T \subseteq S \times \Sigma_1^* \times \Sigma_2^* \times S$ is a transition function. Intuitively, a transducer "reads" words in Σ_1^* and produces words in Σ_2^*. Formally, for $w \in \Sigma_1^*$, the output $\mathcal{T}(w)$ of $\mathcal{T}$ on w is the set of words $v \in \Sigma_2^*$, such that (w, v) is accepted by $\mathcal{T}$ (viewed as an automaton over $\Sigma_1^* \times \Sigma_2^*$). Let F be a finite relation on words. We call $Dom(F) = \{x \mid \exists (x, y) \in F\}$ the *domain* of F and $Img(f) = \{y \mid \exists (x, y) \in F\}$ the *image* of F. The transducer $\mathcal{T}_F$ associated to F is a transducer with a single state s_0 such that $S = S_+ = \{s_0\}$ and $T = \big\{(s_0, \sigma_1, \sigma_2, s_0) \mid (\sigma_1, \sigma_2) \in F\big\}$. A transducer is *functional* iff for any word $w \in \Sigma_2^*$, the output $\mathcal{T}(w)$ is unique. This property of transducers is decidable [5].

If a lexicographic ordering on Σ_2 is given, then it is possible to associate a rank to a word of Σ_2^*, and to build a transducer $\mathcal{T}_{i_F}$ producing integer sequences instead of words on Σ_2^*, by letting $\mathcal{T}_{i_F} = \{(s_0, \sigma_1, Rank(\sigma_2), s_0) \mid (\sigma_1, \sigma_2) \in F\}$.

Let c be a choice node controlled by S. The simple algorithm below can help finding a covert information flow from S to R associated to c.

Algorithm $\mathbf{Covert(c, S, R)}$

$Q = \{$ simple circuits from c to c forming a potential information flow from S to $R\}$

$$F = \Big\{ \big(\ \pi_S(O_q) \cap {\downarrow_{O_q}}(\pi_R(O_q))\ ,\ \alpha(\pi_R(O_q))\ \big) \mid q \in Q \Big\}$$

while $\exists y \in Img(F)$ s.t. $|F^{-1}(y)\}| > 1$ **do**
  choose $x \in F^{-1}(y)$
  $F = F - \{(z1, z2) \mid z1 \in F^{-1}(y) - x\}$
end while
/* Hence, now, we know that $\forall x_1, x_2 \in Dom(F)^2, x_1 \neq x_2 \Rightarrow F(x_1) \neq F(x_2)$ */
Build $T_{i_{F^{-1}}}$
if $|Dom(F)| \geq 2$ and $T_{i_{F^{-1}}}$ functional **then**
  there is an information flow from S to R with $k = |Dom(F)|$ different values
end if

So, if a set of circuits in a HMSC H can be used to transmit information, and if this information can be decoded by a functional transducer $T_{i_{F^{-1}}}$, then there is a structural covert channel in the protocol depicted by H that allows the transmission of k values. Note that the domain of the relation F is $\pi_S(O_q) \cap {\downarrow_{O_q}}(\pi_R(O_q))$ and not $\pi_S(O_q)$, as there must be a causal dependency between what is executed by instance S and the events observed on R.

Let us consider again the description provided by the HMSC of Figure 3 and bMSCs of Figure 5.

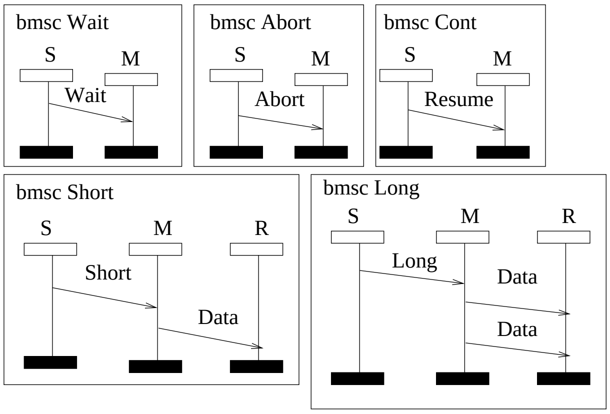

Figure 5: Other examples of bMSCs

Choose the choice node n_1, and a set $Q_{n_1} = \{(n_1, Short, n_2).(n_2, Cont, n_1);$ $(n_1, Long, n_2).(n_2, Cont, n_1)\}$ of paths leaving n_1. Clearly, Q_{n_1} provides a potential information flow, as the instance M can not disturb the scenarios imposed by S and R. The actions performed on S are the emission of messages $Short$ and $Long$, $Abort$, $Wait$ and $Continue$, which will be respectively represented by events e_1, e_2, e_3, e_4, e_5. The relation built from the HMSC and Q_{n_1} is $F = \{(e_1.e5, ?Data), (e_2.e5, ?Data.?Data)\}$. When the word $?Data.?Data.?Data$ is observed on R, then it is impossible to know how it has been produced. Let us replace the bMSC $Long$ by the bMSC of Figure 6.

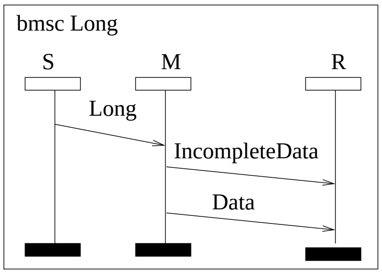

Figure 6: A small variation

For the same node n_1 and the same set of circuits Q_{n_1}, we find a relation $F = \{(e_1.e_5, ?Data), (e_2.e_5, ?IncompleteData.?Data)\}$. This relation can be used to transmit two observable values. Note that a similar covert channel can be detected from node n_2, taking into account the emission on message *Continue*, but with the same outputs on R.

4 Conclusion

This article has shown how structural information flows can be detected on a simple scenario language. The main advantages of this technique is that it provides immediately an user with scenarios for using a potential covert channel, and a decoder for covert messages, given as a transducer. With this material, it should be easy to test for the effective presence of a given covert channel. Message sequence charts also contain more elaborated constructs such as data, guards, and so on, that are often used to describe requirements. Taking such constructs into account is possible if one can translate them to simpler ones. For instance, a bMSC where a parameter can be set to 1,2 or 3 can be translated into a HMSC with three choices. Of course, dealing explicitly with all possible values would be quite inefficient, and we are investigating how covert channel analysis can be performed with symbolic values.

This paper has only considered a coding and transmission strategy using a single choice node. Finding more elaborated strategies allowing data transmition is also an ongoing work. So far, we have only considered covert data transmission. The establishment of a connexion between covert channel users and the initalisation of the channel have been ignored, but we think that this can be also studied in our scenario framework.

Finally, we have not considered timing issues in this paper. However, they are central to covert channel analysis, as a low bandwidth channel can be ignored and a channel with high bandwidth must be treated. By providing information on message transmission time and duration of events, it could be very simple to adapt the work of [11] to approximate the bandwidth of a channel.

References

[1] K. Ahsan and D. Kundur. Practical data hiding in TCP/IP. In *Workshop on Multimedia Security at ACM Multimedia '02*, Dec. 2002.

[2] G.R. Andrews and R.P. Reitmans. An axiomatic approach to information flows in programs. *ACM transactions on Programming languages and Systems*, 2(1):56–76, January 1980.

[3] D.E. Bell and J.J. LaPadulla. Secure computer systems: a mathematical model. MITRE Technical Report 2547, MITRE, May 1973. Vol II.

[4] D.E. Bell and J.J. LaPadulla. Secure computer systems: mathematical foundations. MITRE Technical report 2547, MITRE, March 1973. Vol I.

[5] J. Berstel. *Transductions and Context-Free-Languages*. B.G. Teubner, Stuttgart, 1979.

[6] Jean Berstel and Dominique Perrin. *Theory of codes.* Revised version, 2002.

[7] Common Criteria. Common criteria for information technology security evaluation part 3: Security assurance requirements. Technical Report CCIMB-99-033, CCIMB, 1999.

[8] ITU-TS. *ITU-TS Recommendation Z.120: Message Sequence Chart (MSC)*. ITU-TS, Geneva, September 1993.

[9] R.A. Kemmerer. Shared ressources matrix methodology: an approach to indentifying storage and timing channels. *ACM transactions on Computer systems*, 1(3):256–277, 1983.

[10] B. Lampson. A note on the confinement problem. *Communication of the ACM*, 16(10):613–615, Oct. 1973.

[11] P. Le Maigat. A (max,+) approach for time in message sequence charts. *5th Workshop on Discrete Event Systems (WODES 2000)*, 2000.

[12] S.B. Lipner. A comment on the confinement problem. In *Proceedings of the fifth symposium on Operating systems principles*, 1975.

[13] I. Moskowitz and M. Kang. Covert channels - here to stay? In *Proceedings of COMPASS'94*, pages 235–243. IEE Press, 1994.

[14] NCSC. *A guide to Understanding Covert Channel Analysis of Trusted Systems.* Number NCSC-TG-030 [Light Pink Book] in Rainbow Series. NSA/NCSC, Nov. 1993.

[15] C.H. Rowland. Covert channels in the TCP/IP protocol suite. Technical Report Tech. Rep. 5, First Monday, Peer Reviewed Journal on the Internet, July 1997.