Using Data-Independence in the Analysis of Intrusion Detection Systems

Gordon Thomas Rohrmair and Gavin Lowe

Oxford University Computing Laboratory,
Wolfson Building, Parks Road, Oxford, OX1 3QD, UK
{gordon.rohrmair, gavin.lowe}@comlab.ox.ac.uk

Abstract. In a previous paper we showed how to use the process algebra CSP to discover de-synchronisation attacks on intrusion detection systems. However, our analysis was not complete: if we failed to find an attack, it was not clear whether that was an artifact of the abstractions used in the model, or whether there really was no attack. In this paper we show how we can perform a more complete analysis, by building a model with a slightly different focus, combined with results taken from the area of data independence.

1 Introduction

In a previous paper [RL02], we showed how to use the process algebra Communicating Sequential Processes (CSP) [Ros97] to discover de-synchronisation attacks on intrusion detection systems. Such attacks occur when the state of the intrusion detection system (IDS) becomes de-synchronised from that of the system it aims to protect, and fails to recognise the attack. In particular, de-synchronisation attacks exist that evade detection by a signature-based network intrusion detection system, even if the IDS recognises all signatures of attacks. Such de-synchronisations are typically caused by interactions between the IDS and the underlying network protocol.

In [RL02], we were able to reproduce the de-synchronisation attacks first described in [Pax99,PN98]. We modelled the systems as CSP processes, and used the model checker FDR to explore the state space looking for states where the target fails, without the IDS raising an alert. We also adapted the models to show how to prevent the de-synchronisation attacks. Furthermore, we identified three de-synchronisation subclasses:

1. De-synchronisation where the IDS and target behave identically, but the input streams are different; this is illustrated in the time-to-live model in Section 2.1.
2. De-synchronisation where the input streams are the same, but the systems behave differently under certain conditions; this is illustrated in the packet reassembly model in Section 4.
3. De-synchronisation where both the input streams and the behaviours of the systems are different.

However, in order to perform this analysis—for example, to keep the state space finite—we had to perform various abstractions, particularly concerning the set of network packets and the set of attack signatures in the model. It is not clear that these abstractions were sound: our failure to find attacks on the adapted models could have been caused by an over-abstraction that lost attacks. To show that our abstractions are correct we have to show that any attack can be reduced to one that would be found by our model. This is the question we address in the current paper.

In the next section, we review the work of [RL02], describing, via an example, how de-synchronisation attacks can be discovered automatically; we briefly review the relevant results from data-independence [Laz97,Ros97]. In Section 3, we adapt the models, so as to take a slightly different view, while still finding de-synchronisation attacks; moreover, we use results from data-independence to show that this analysis is complete, in the sense that it is independent of the underlying types of network packets and attack signatures used in the model. In Section 4 we show the generality of our approach, by applying it to a different example, based upon the packet reassembly algorithm of the Internet Protocol. We sum up in Section 5. A brief introduction to CSP is included as an appendix.

2 Background

An Intrusion Detection System (IDS) is used to detect abuses, misuses and unauthorised uses in a network; more generally, they detect violations against the security policy of a network. They identify intrusions by spotting known patterns—called *signatures*—or by revealing anomalous behaviour of protected resources (e.g., network traffic or main memory usage).

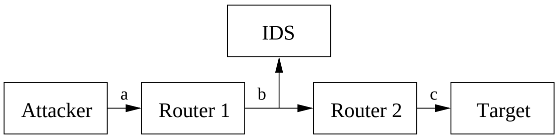
2.1 Detecting de-synchronisation attacks

In this section we review the work of [RL02], so as to demonstrate a simple de-synchronisation attack, and how it can be found using CSP and its model checker FDR [Ros97]. We believe that such attacks could equally be found using other model checkers; an advantage of using CSP is that it has a well-established body of theory, which we will draw upon below.

We consider a signature-based network intrusion detection system. The IDS monitors packets on the network, looking for signatures—i.e. sequences of packets—corresponding to known attacks. We assume the IDS is perfect, i.e., it knows all signatures of attacks that would cause the target to fail.

Packets in the Internet Protocol make use of a *time-to-live* (TTL) field, which records the number of network hops that a packet can make; each router decrements the TTL, and discards the packet if it reaches zero.

We modelled a small network as below.



We modelled packets that contain just a data field and a TTL field. The attacker, nondeterministically chooses packets and injects them into the network:

$$Attacker = \sqcap_{x:TTL, y:DATA} a.x.y \rightarrow Attacker \sqcap STOP.$$

The routers simply receive the packets, decrement the TTL field, and forward the packets if the TTL is not zero:

$$\begin{aligned}
&Router(in, out) = \\
&\quad in?x?y \rightarrow \\
&\quad \text{if } y > 1 \text{ then } out.x.y-1 \rightarrow Router(in, out) \text{ else } Router(in, out).
\end{aligned}$$

The IDS is parameterised by a set *sigs* of attack signatures (a signature is just a sequence of data values); it is also parameterised by a set *vulnerabilities*, each element of which will be a suffix of an attack signature for which the corresponding prefix has been observed. The IDS monitors packets passing on the network, and if it observes an attack signature in *sigs*, it performs the event *alert*, representing an alarm:

$$\begin{aligned}
&IDS(sigs, alerts) = \\
&\quad b?x?y \rightarrow \\
&\quad \text{let } alerts' = \{s \mid \langle x \rangle ^\frown s \in sigs \cup alerts\} \\
&\quad \text{within if } \langle x \rangle \in alerts \text{ then } alert \rightarrow IDS(sigs, alerts') \\
&\qquad\qquad \text{else } IDS(sigs, alerts').
\end{aligned}$$

Similarly, the target is parameterised by the same set *sigs*; if it receives a sequence of packets from *sigs*, it performs the event *fail*, representing a successful attack:

$$\begin{aligned}
&Target(sigs, vulnerabilities) = \\
&\quad c?x?y \rightarrow \\
&\quad \text{let } vulnerabilities' = \{s \mid \langle x \rangle ^\frown s \in sigs \cup vulnerabilities\} \\
&\quad \text{within if } \langle x \rangle \in vulnerabilities \text{ then } fail \rightarrow Target(sigs, vulnerabilities') \\
&\qquad\qquad \text{else } Target(sigs, vulnerabilities').
\end{aligned}$$

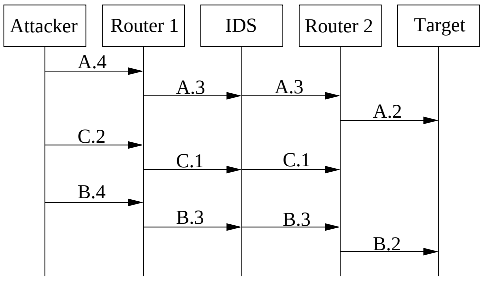
In order to make analysis possible, we had to pick values for the type *DATA* of data packets, and the set *sigs* of attack signatures; we took $DATA = \{A, B, C\}$, and $sigs = \{\langle A, B \rangle\}$.

We combined these processes in parallel, as depicted above, and then hid all events other than *fail* and *alert* (i.e. turned them into internal events). We then used FDR to detect whether an undetected attack is possible, by testing whether the system refines a specification that asserts that every *fail* event is accompanied by a corresponding *alert*:

$$Spec \;=\; alert \rightarrow fail \rightarrow Spec \sqcap fail \rightarrow alert \rightarrow Spec \sqcap STOP.$$

There is some asynchrony in the system, so the *fail* and the *alert* could occur in either order; we test for refinement in the stable failures model [Ros97], to ensure that if a *fail* occurs first then an *alert* must occur.

FDR found several similar attacks; a typical one is depicted below.



The attacker sends three packets with data fields A, C, B, respectively, but such that the TTL field for the second packet is not enough for that packet to reach the target. The target receives the sequence of packets $\langle A, B\rangle$, causing a *fail* event. The IDS does not detect the attack, having seen the sequence of packets $\langle A, C, B\rangle$. (FDR also found several false attacks, where the IDS signalled an *alert* without the target failing, because some of the packets of the attack had a small TTL field so failed to reach the target.)

The cause of the attack is clear: the IDS does not take into account the topology of the network. We adapted the model so that the IDS ignores packets with a TTL that causes the packet to not reach the target. FDR then found that, *for this model*, the IDS detects all attacks. However, it is not clear whether the same would be true if we considered different instantiations of the sets $DATA$ and $sigs$, or whether we have over-abstracted; this is the question we consider in Section 3.

2.2 Data-independence

Often the behaviour of a CSP process is dependent on various parameters, such as the underlying types used in events, or the number of nodes in a network. However, analysis using explicit model checking, for example with FDR, can consider only a single value for each parameter at a time. We would like to be able to verify a system for all values of the parameters via a small number of explicit checks; this is often known as the *Parameterised Verification Problem*.

Data independence is a tool that can be used to address this problem. The goal of this approach is so come up with a bound N such that if a refinement holds when a type parameter is instantiated with a type of size N, then the refinement also holds for all larger types.

A process P is said to be data-independent with respect to a type T if the only operations it performs on values of T are to input them, store them, and output them; in particular, P never performs any computation on values of T that constrains what T might be.

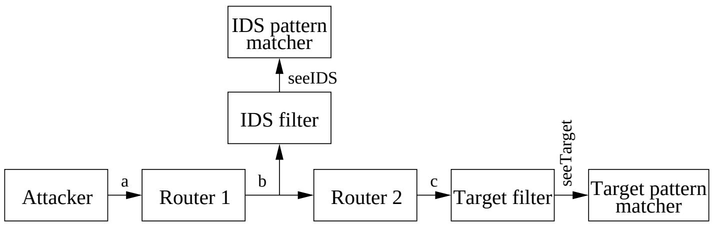
A process satisfies the condition NoEq_T if it performs no test of equality between members of T: such tests could be explicit, for example within 'if-then-else' constructs, or implicit via synchronisations between parallel components on single elements of T. A process satisfies Norm_T if, essentially, it contains no nondeterminism the effects of which are not immediately apparent (see [Laz97,Ros97] for a formal definition).

The following theorem is from [Laz97,Ros97]:

Theorem 1. *Suppose that Spec and Impl are data-independent processes, that both satisfy* NoEq_T*, and that Spec satisfies* Norm_T*. If* $Spec \sqsubseteq Impl$ *when* T *is taken to be of size 2, then* $Spec \sqsubseteq Impl$ *for all finite or infinite values of* T *of size at least 1.*

3 Towards a more complete analysis

We now change the focus of the model from Section 2.1, in order to move towards a more complete analysis, independent of the set *sigs* of attack signatures. We observe that the IDS is really doing two things: filtering out packets that will not reach the target; and performing pattern matching on the remaining packets. It is therefore possible to split the IDS into two different processes corresponding to these functions. In related models, corresponding to different network protocols (e.g. in Section 4, below), the target process similarly performs a combination of filtering and pattern matching, and so it is possible to split the target into two processes. This gives a topology as below.



We can then make the following observation:

Observation 1 *If the stream of packets passed to the pattern matching component of the IDS is the same as the stream of packets passed to the pattern matching component of the target, then the IDS will detect all attacks.*

The hypothesis of Observation 1—that the two components see the same stream of packets—is an easy property to test. In fact it does not even require us to model the two pattern matching components, simply the messages passed to them on the channels *seeIDS* and *seeTarget*. The advantage of this change of focus is that is allows us to remove the parameter *sigs* from the model, thus leading towards a more general verification.

The two filtering processes (where the IDS takes the distance to the target into account) can be modelled by:

$$\begin{aligned} IDS &= b?x?y \to \text{if } dist \leq y \text{ then } seeIDS.x \to IDS \text{ else } IDS, \\ Target &= c?x?y \to seeTarget.x \to Target. \end{aligned}$$

where $dist$ is the distance from the IDS to the target. The rest of the network is unchanged.

It is easy to capture the hypothesis of Observation 1 as a refinement assertion; there is a certain amount of buffering in the system, and the specification has to take this into account:

$$\begin{aligned} Spec &= \bigsqcap_{x:T} (seeIDS.x \to Spec'(x) \sqcap seeTarget.x \to seeIDS.x \to Spec) \\ &\quad \sqcap \\ &\quad STOP, \\ Spec'(x) &= seeTarget.x \to Spec \\ &\quad \sqcap \\ &\quad \bigsqcap_{y:T} seeIDS.y \to seeTarget.x \to Spec'(y). \end{aligned}$$

The specification captures the property that the IDS and target see the same stream of packets, except that the IDS might at any point have seen up to two more packets than the target, or the target might have seen one more packet that the IDS.

We can then use FDR to check that the system failures-refines $Spec$. The refinement holds, and we can then use Observation 1 to deduce that the IDS detects all attacks *for all possible sets of attack signatures*.

However, this appears to leave us only slightly better off than before: we can verify the system for a fixed type $DATA$; but does this tell us anything about systems with different values for $DATA$? It turns out that we can use Theorem 1 to show that this is indeed the case. The system and specification processes are both data independent with respect to the type $DATA$ (when the pattern matching against attack signatures was included, the IDS and target were not data independent); both satisfy NoEq_{DATA}; and the specification satisfies Norm_{DATA}. The theorem therefore tells us that we have only to check the refinement for a type $DATA$ of size 2, say $DATA = \{A, B\}$, to have verified it for all values of $DATA$.

4 The packet reassembly model

In this section we consider another example, so as to demonstrate the general nature of our technique.

Sometimes an Internet Protocol (IP) packet has to be routed through different networks. Not all networks have the same properties. Therefore, a packet might have to be split up into fragments, tagged with their position in the original packet (fragment offset); this process is termed *fragmentation*. The target receives an increased supply of smaller fragments instead of one IP packet, and therefore has to reconstruct the initial packet; this process is called *reassembly*. The algorithm in [dR81] collects the fragments and puts them into the right place of the reserved buffer, before passing the packet to the layer above.

Sometimes data is received at the same fragment offset as a previously received fragment. In such a case, a decision has to be made whether to favour the old or the new data. RFC 791 [dR81] leaves unspecified which should be preferred, but the recommendation is to prefer new data, so that if the algorithm receives data from the same position twice, the new data will overwrite the old. However, not all implementations follow this recommendation.

If the IDS favours old data, and the target favours new data—or vice versa—then there is a de-synchronisation attack: the attacker can send a fragment with the same offset as a previous fragment, which will be accepted by the target but not by the IDS—or vice versa—leading to them reassembling different packets.

In [RL02] we showed how to detect this attack using CSP and FDR. We built CSP models of the target and IDS, which accumulated packets in a buffer, favouring old or new data according to their policies, and which when the buffer was full, compared the packet with the attack signatures, performing a *fail* or *alert* event if appropriate.

We also showed that if the IDS and target follow the same policy, then there is no de-synchronisation attack, at least for the choice of $DATA$ and $sigs$ used in that model.

We can adapt the techniques of the previous section to this setting. We can model the reassembly algorithms, at both the target and IDS, as separate processes. We can then ask whether the stream of packets passed to the pattern-matching components are the same, using essentially the same specification as for the time-to-live model; this generalises our analysis to all possible sets of attack signatures. We can then use Theorem 1 to show that we need only perform this analysis for a type $DATA$ of size 2 to deduce that the same result holds for all larger types.

5 Conclusion

In this paper we have described how to perform a more complete analysis of the deployment of intrusion detection systems, so as to discover—or show the absence of—de-synchronisation attacks. We have changed the focus of our previous models so as to ask whether the IDS and target, after appropriate filtering, see the same stream of packets; this makes our analysis independent of the set of attack signatures in question. We have then used a result from data independence to show that the analysis is independent of the underlying type of data.

The converse of Observation 1 is not, in general, true: if the attacker can cause the target and IDS to see different streams of packets, this does not necessarily mean that a de-synchronisation attack is possible. However, we believe that the converse of Observation 1 is true in all but contrived examples. Even if this is not the case, the worst that will happen is that we detect a false attack, which will give us a better understanding of the system.

There are other abstractions that we have made in our models, which we briefly discuss now.

In the time-to-live model, the TTL field can, in principle, take arbitrarily large values (in the Internet Protocol it is implemented as an 8-bit number). However, for a network with diameter d (counted in TTL-decreasing hops), it is clear that there is no point in considering packets with time-to-live field greater than $d+1$, for such packets display identical behaviour to those with time-to-live field equal to $d+1$.

We now argue that in the reassembly model, it is sufficient to consider a buffer of size three, in the sense that if there is an attack upon a system that uses a larger buffer, then there is also an attack upon a system that uses a buffer of size three. To see this, consider an attack upon the system that uses a larger buffer, in which the IDS and target reassemble packets that first disagree in position k. We map this onto an attack upon the system that uses a buffer of size three by remapping fragment offsets as follows: all fragment offsets less than k are mapped onto 0; fragment offsets of k are mapped onto 1; and all fragment offsets greater than k are mapped onto 2. If is then clear that in the system with buffer size three, this will correspond to an attack where the IDS and target reassemble packets that disagree in position 1 (and maybe elsewhere). (The above remapping works only for $k > 0$, but there is a simpler remapping that works when $k = 0$.)

We have considered only a single network topology. This turns out to be an important consideration. If there are two or more routes between the IDS and the target, then it is possible for the IDS and target to see fragments in a different order, which opens up the possibility of another de-synchronisation attack. However, most local networks have only a single route from the gateway to each host, which prevents such attacks. Under this condition, we believe that our abstraction has not lost any attacks:

- It is safe to abstract away from the parts of the network before the IDS, because the attacker can effectively choose what fragments are sent past the IDS (channel b in the earlier models).
- It is also safe to abstract away those parts of the local network not on the direct route from the IDS to the target, because they do not affect what the target sees.
- The remaining issue is the number of routers between the IDS and the target; there does not seem to be any intrinsic difference between one router reducing the TTL field by one, and N routers reducing the TTL by N, because the attacker is able to choose the TTL appropriately; however, the number of routers does affect the buffering of the system and hence the appropriate specification process; it is not clear how to formalise this argument.

Finally, we have, of course, abstracted away from many of the details of the underlying network protocol. Our long-term aim is to remove these abstractions, so as to model the whole of the Internet Protocol version 6.

References

[dR81] Marina del Rey. *RFC 791 Internet Protocol: DARPA Internet program protocol specification*, 1981.

[Laz97] Ranko Lazić. *A Semantic Study of Data-Independence with Applications to the Mechanical Verification of Concurrent Systems.* D.Phil., Oxford University, 1997.

[Pax99] Vern Paxton. BRO: A system for detecting network intruders in real-time. *Computer Networks*, 31:2435–2463, 1999.

[PN98] Thomas H. Ptacek and Timothy N. Newsham. Insertion, evasion, and denial of service: Eluding network intrusion detection. *Secure Networks*, 1998.

[RL02] Gordon Rohrmair and Gavin Lowe. Using CSP to detect insertion and evasion possibilities within the intrusion detection area. In *Proceedings of BCS Workshop on Formal Aspects of Security*, 2002.

[Ros97] A. W. Roscoe. *The Theory and Practice of Concurrency.* Prentice Hall, 1997.

A CSP notation

An event represents an atomic communication; this might either be between two processes or between a process and the environment. Channels carry sets of events; for example, $c.5$ is an event of channel c.

The process $a \rightarrow P$ can perform the event a, and then act like P. The process $c?x \rightarrow P_x$ inputs a value x from channel c and then acts like P_x.

The process $P \sqcap Q$ represents an internal or nondeterministic choice between P and Q; the process can act like either P or Q, with the choice being made according to some criteria that we do not model. The process $\bigsqcap_{i:I} P_i$ represents a replicated nondeterministic choices, indexed over set I.

A trace is a sequence of events that a process can perform. A failure of a process is a pair (tr, X), representing that the process can perform the trace tr to reach a stable state (i.e. where no internal activity is possible), where none of the events from X can be performed. Process P is failures-refined by process Q if all the failures of Q are also failures of P:

$$P \sqsubseteq Q \Leftrightarrow failures(P) \supseteq failures(Q).$$