

A Case-Based Approach to Network Intrusion Detection

Daniel G. Schwartz, Sara Stoecklin, and Erbil Yilmaz
Department of Computer Science

Florida State University
Tallahassee, FL 32306-4530

schwartz@cs.fsu.edu, stoeckli@cs.fsu.edu, yilmaz@cs.fsu.edu

Abstract – *This paper reports progress on creating a case-based implementation of the well-known Snort intrusion detection system. Snort is a simple rule-based system that is known to suffer limitations, including both failure to detect certain kinds of intrusions and the frequent raising of false alarms. We believe that a case-based reasoning approach can provide a framework in which to incorporate more sophisticated artificial intelligence techniques that will help overcome some of these limitations. In addition, the present system is intended to apply more generally to other aspects of network security, as well as other domains related to protecting the nation’s critical infrastructure. The system is being built using the modern software engineering technique known as “adaptive” or “reflective architectures,” which will make it easily adaptable to other kinds of problem domain.*¹

Keywords: Information security, intrusion detection, case-based reasoning, snort, infrastructure protection.

1 Introduction

The first line of defense for any organizational network is typically a firewall, and the second line is typically an intrusion detection system (IDS). Contemporary IDSs look at each individual packet as it enters the network and, based on information contained in the packet’s headers together with its payload, tries to determine if the packet represents suspect activity.

Among the existing IDSs, perhaps the best known is the open-source system known as Snort [1]. Snort is attractive for the same reasons as are all open-source projects: the entire system including source code is available free of charge, and there is a dedicated collection of users that continually contribute to its evolution.

¹This work was supported by the US Army Research Office, grant number DAAD19-01-1-0502.

Snort is a rule-based system, with each rule expressing some action to be performed if the packet in question meets certain criteria. A typical such rule is shown in Figure 1. This says that the rule is of the kind that, if fired, will raise an alert, and that this will happen if the protocol is tcp, it is coming from any IP address and any port number, it is going to an IP address in the range of 192.168.1.0 to 192.168.1.255 (a class C network), the destination port number is not greater than 111, and the packet payload contains the hex code 000186a5, in which case the alert to be raised is the message “mound access,” expressing the warning that someone is trying to mount a directory through some service port where this normally should not take place. The current Snort rule set contains approximately 900 such rules.

```
alert tcp any any ->  
192.168.1.0/24!111: (content:  
|“000186a5”|; msg “mound  
access”)
```

Figure 1: Example Snort rule.

Snort is known to suffer from the limitations that it sometimes fails to detect unwanted activity and, perhaps more frustratingly, that it frequently raises false alerts. There has been some recent effort to remedy the latter through introduction of the notion of “flows” [cf. 1] but the results are still not completely satisfying.

This paper reports progress on developing a case-based implementation of Snort. A motivation for this approach is that the case-based reasoning (CBR) approach provides a framework in which one can provide more sophisticated degrees of intelligence, e.g., look-

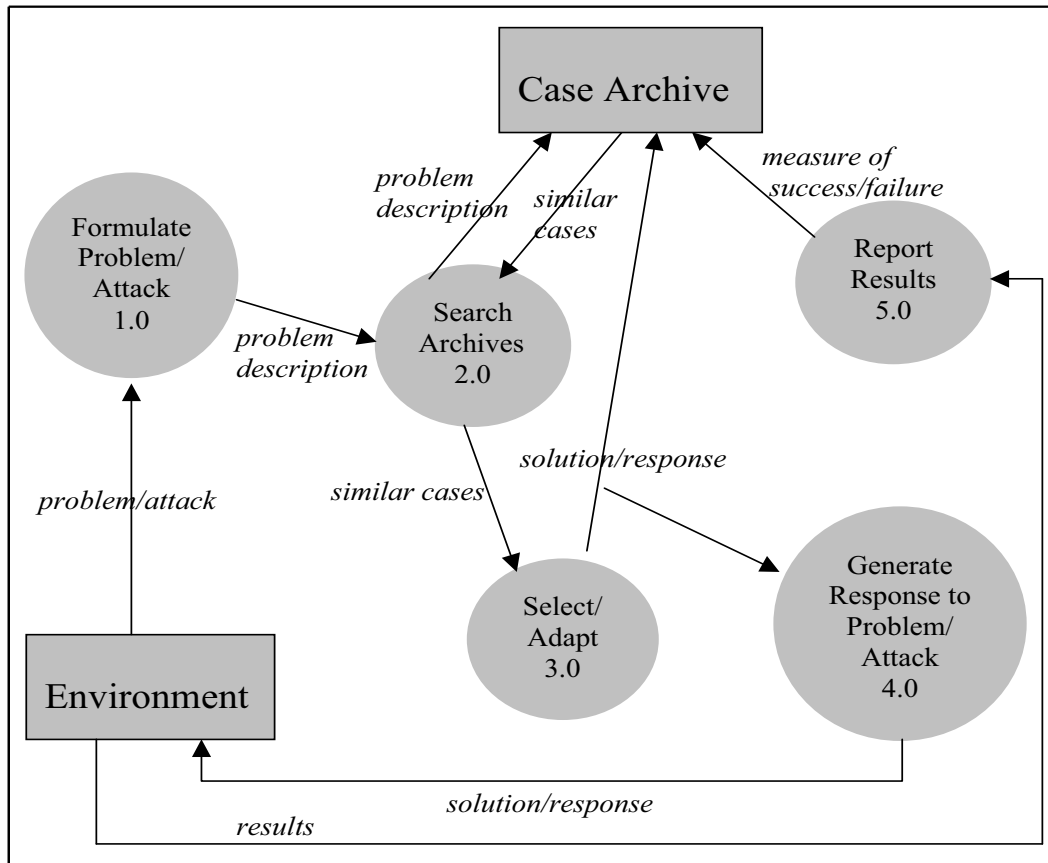


Figure 2: CBR process feedback loop.

ing for patterns among multiple packets, or possibly capturing and examining entire login sessions.

At the same time, this work is aimed at other aspects of network information security, as well as at the broader realms associated with protecting the nation's critical infrastructure. Our CBR system is being developed using the contemporary software engineering technique known as "adaptive" or "reflective architectures," together with some recent advances in using Java with XML. These features enable one to generalize the notion of a case base and thereby facilitate using the same generic case-based reasoning module to create case-based reasoners for other problem domains. In addition, the use of Java and XML will make it easier to extend this to a distributed system and/or to exchange data with other systems across networks. Thus the current work, although presently focusing on Snort, is seen as the first step towards a much larger and more comprehensive set of applications.

2 Case-based reasoning

The basic features of a general CBR system are depicted in Figure 2. The heart of the system is the case

archive, where each entry is a record giving (i) the features characterizing some previously experienced problem situation, and (ii) the action or actions that were taken to remedy this. Then, when some problem is detected in the surrounding environment, this is formulated as a set of case features (step 1.0), and this formulation is fed to a search engine that looks through the archive for cases whose features match those of the given problem (step 2.0). This matching process normally uses a *similarity metric*, which allows for approximate matches. The search engine then returns a collection of matching cases ranked according to their degrees of similarity with the given problem. Then, either some one of these cases is selected (step 3.0), and its recommended action is taken (step 4.0), or the returned cases are used as the basis for formulating a new case, with perhaps some new kind of action (step 3.0), this case is entered into the archive, and the recommended action is taken (step 4.0). In whichever manner an action is decided upon, the results of the action including some measure of its success or failure is entered into the case record (step 5.0). This information then may be used in future applications of

the reasoner in deciding on actions for future problem situations, favoring those case actions that tended to succeed and avoiding those that were known to fail.

Case-based reasoners have been one of the success stories for the field of artificial intelligence (AI), having found a multitude of applications throughout business, industry, government, and the department of defense. Our project began with investigation of a “conversational” CBR system that had been developed at the Navy Research Laboratory, the Navy Conversational Decision Aids Environment (NaCoDAE) [2, 3]. This system supplements the usual similarity metric with an interactive question and answer process by which the user narrows in on the archived cases that are most similar to the current problem situation. In effect, this is a hybrid system that uses features of a rule-based reasoner to assist in the CBR process.

NaCoDAE has several attractive features, but unfortunately does not provide the degree of procedural abstraction needed for the kind of adaptability we sought for the present research. Similar problems seem inherent also in other extant CBR systems. It was for this reason that we decided to build our own system essentially from scratch.

3 The CBR implementation of Snort

Implementing Snort as a case-based system entails treating each Snort rule as a case. This is fairly straightforward. To illustrate, the case that corresponds to the rule given in Figure 1 is shown in Figure 3. What might be regarded as the “premises” of the Snort rule here become case features that are to be matched during the case retrieval process, and what might be regarded as the “conclusion” of the rule becomes the case action.

The direct implementation of Snort as a CBR system, however, uses only part of the foregoing framework. This is because the features of a packet either do or do not match the features of any particular case. In other words, the similarity metric is bivalent, inasmuch as matches are always exact, which means that there is no need either for a similarity ranking of retrieved cases or for case adaptation (Figure 2, step 3.0). Hence there is no need for saving any cases (no new ones are being created), nor is there need to keep records of the results (Figure 2, step 5.0).

Thus, from a CBR standpoint, the present system is quite simple. As was stated, however, the primary aim is not merely to replicate the behavior of Snort, but to use Snort as the starting point for a much more general, adaptive system, which can be used not only to improve on the capabilities of Snort, but also to

Match features:

Protocol: tcp

Source IP address: any

Source port: any

Destination IP address:

from 192.168.1.0

to 192.168.1.255

Destination port: not > 111

Packet contents: 000186a5

Case action:

Output alert “mound
access”

Figure 3: The foregoing Snort rule as a case.

address other, completely different problem domains.

4 System overview

4.1 Generic Case-Based Reasoning

A schematic overview of the current system is shown in Figure 4. A prevailing theme in the creation of this design was to separate those aspects of CBR that are common to all case-based reasoners from those aspects that are specific to the given application domain. In effect, this amounts to defining a generic case-based reasoner, of which each domain specific case-based reasoner will be an instance.

To accomplish this it is first necessary to abstract out the general notion of a case. Moreover, inasmuch as a case is defined by its features (as in Figure 3), this in turn makes it necessary to identify the general notion of a feature. The objective, then, is to create a generic framework within which virtually any type of feature and any type of case can be defined.

Given that this can be accomplished, then it is next necessary to abstract out the general notion of a similarity metric, to be used in the case retrieval process. Since cases are identified by their features, this means that one needs to capture the generic notion of a feature comparator, of which each specific feature comparator is an instance. For example, in the case shown in Figure 3, the *protocol* feature requires an exact match (either the protocol of the incoming packet is “tcp” or it is not), so the requisite comparator would one that calls for an exact match between

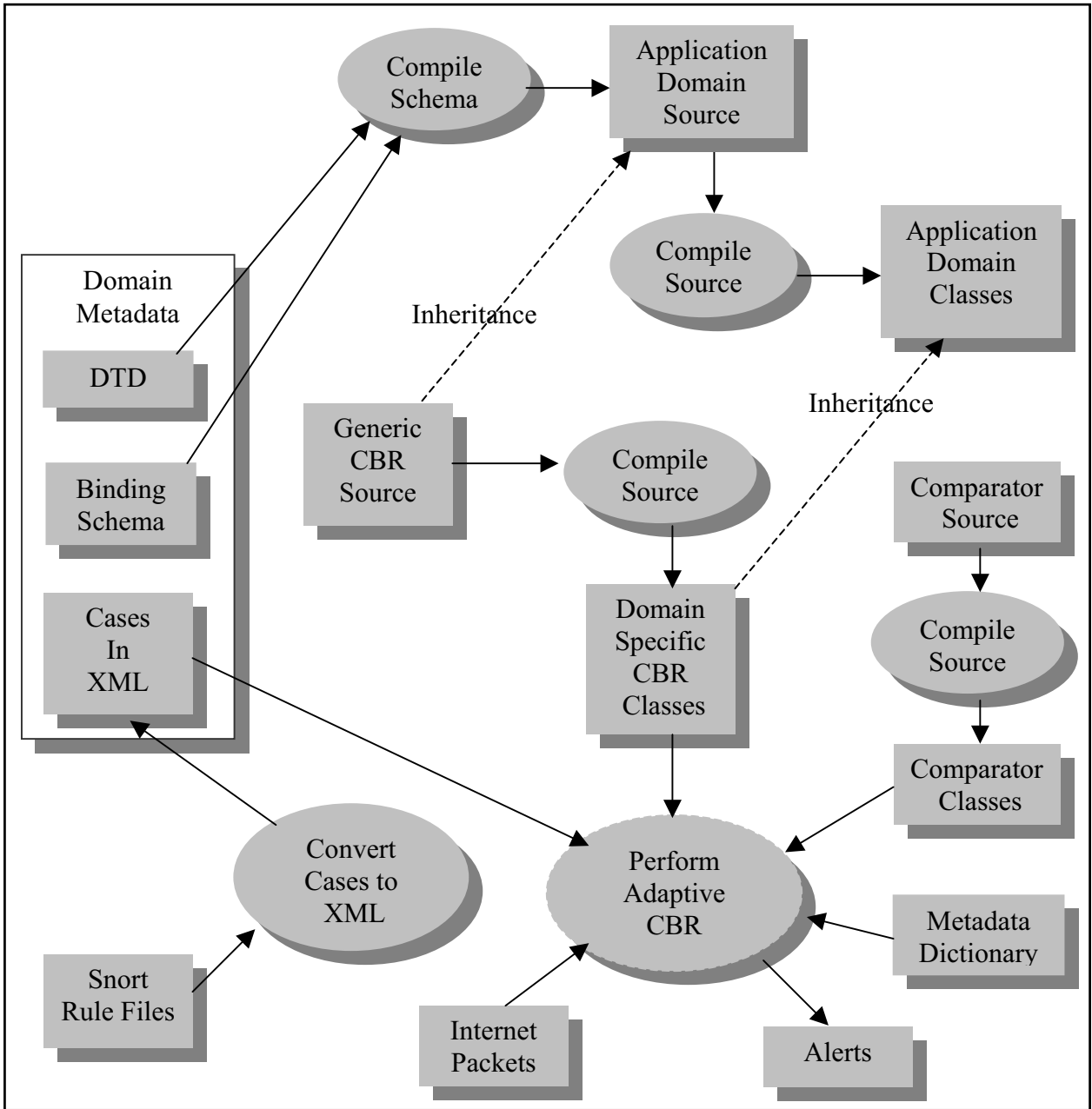


Figure 4: System overview.

character strings. By contrast, the *destination IP* feature requires determining whether the source IP of the incoming packet falls within a certain range of IP addresses, so that here one needs a range comparator for integers. A similarity metric is then defined as a collection of feature comparators, together with a rule specifying whether all or only some of the features need to match, as well as whether some feature need only approximately match, or whether some features may be weighted as to their importance, etc. While the Snort application domain requires a complete match on all features, and all features are regarded as equally important, the generic system accommodates a very broad range of types of comparators.

4.2 Domain Definition

To achieve the desired level of abstraction needed for case definition we have employed XML and Sun Microsystems's Java XML Binding (JAXB) tool kit. This is illustrated on the left side and top of Figure 4. Cases, including case features, are all represented in XML according to the specifications given in a data type definition (DTD). The specific DTD used for Snort cases is shown in Figure 5. For the purposes of this project, a simple parsing program was written that automatically converts an arbitrary Snort rule set into a the corresponding XML representation.

JAXB enables one to automatically generate the Java classes necessary to parse XML documents according to a given DTD. To do this, one need only write a binding schema, which provides additional details about how these classes are to be generated (data conversions, method and class names, etc.). The resulting Java classes can then be used to convert the cases recorded in XML into corresponding Java objects, and vice versa.

The virtue of this approach is that, in order to develop new cases as required by a new problem domain, one need only create a new DTD and binding schema. Then all the source code necessary for dealing with the XML cases can be generated automatically. The use of XML for case representation was inspired in part by an earlier work along these same lines [4].

4.3 Case Feature Comparators

For each new type of case feature one needs also to define a comparator that determines whether, or to what degree, a given problem situation feature matches with the corresponding feature in the case archive. For example, as discussed above, the *protocol* feature requires an exact string comparator, which returns *true* if the string in the problem situation protocol feature exactly matches the string in the corresponding feature on a case in the archive. Similarly, the aforemen-

tioned *source IP* requires an integer range comparator. It should be noted that several different features may use the same comparator, so that in general the number of comparators will be much smaller than the number of different features.

As indicated on the right side of Figure 4, feature comparators comprise a collection of classes that are stored and compiled separately from the other modules. This makes it easy to create new comparators and add them to the collection as desired. In a future system, this module will additionally provide means for specifying how comparators may be combined to form different sorts of similarity metrics.

4.4 Adaptive CBR process

The generic CBR component assumes no knowledge regarding case features or comparison methods. It does, however, inherit from the application domain module, so that the CBR module needs to be recompiled whenever there is a change in the domain definition. This, nonetheless, is all that is required, so that in principle the source code for the generic CBR should never need to be modified. Compilation of this code produces a domain-specific case based reasoner.

In order to match case features with their associated comparators, the system additionally employs a metadata dictionary. Specifically, for each feature, this gives the feature's name, its value type, and the name of the associated comparator.

Given these various components, the performance of the CBR process may be described as follows. A packet is received from the outside world (Internet) and fed to the CBR module. This in turn scans the XML representation of the case archive, and for each case does the following. For each case feature as described by the DTD, it looks up this case by name in the metadata dictionary, extracts the value type and comparator name for this feature, creates an instance of the needed comparator using reflection on the compiled comparator class having this name, and then applies this comparator to determine whether the packet feature matches the corresponding feature of that case. It thus proceeds in this manner through the entire case library, retrieving all cases that match the given input packet, and performing the prescribed case action. In the case of Snort, this is typically (but not always) the outputting of an alert message.

This use of a metadata dictionary to separate out the nongeneric components and methods (features and comparators) is an example of what is more generally known as an adaptive, or reflective, architecture [5]. In this context, such metadata is sometimes referred to as the "knowledge level." The term "reflective" stems from the use of Java's capacity for run-time reflection,

```

<!ELEMENT snortcasevector (snortcase+) >
<!ELEMENT snortcase (caseid, requiredfeature+, optionalfeature*, solution)>

  <!ELEMENT caseid (#PCDATA)>

  <!ELEMENT requiredfeature (featurevalue+)>
    <!ATTLIST requiredfeature
      featurename ( protocol | sourceip | sourceport | destinationip
|destinationport ) #REQUIRED
      negation (NOT) #IMPLIED>

  <!ELEMENT optionalfeature (featurevalue+)>
    <!ATTLIST optionalfeature
      featurename ( ttl | tos | flags | id | dsize | message |
sequencenumber | acknowledgement | content | offset ) #REQUIRED
      negation (NOT) #IMPLIED>

  <!ELEMENT featurevalue (#PCDATA)>
  <!ATTLIST featurevalue
      negation (NOT) #IMPLIED>

  <!ELEMENT solution (actiontype, message?)>

    <!ELEMENT actiontype (#PCDATA)>
    <!ELEMENT message (#PCDATA)>

```

Figure 5: Sample XML data type definition.

where it can dynamically determine the types of the objects it is acting upon and so also dynamically create the methods (in this case the comparators) to apply based on the object's type.

The fact that this architecture leads to very flexible software systems is the motivation for also calling it "adaptive." We hope through the use of this architecture to build a CBR system which can in this manner be easily adapted to other problem domains.

5 Future work

The coding for the core functionality of the Snort implementation is now complete, but there remains work to be done on the user interface. We expect this to be completed by the time this paper appears in print. A near term future effort will focus on building in higher degrees of intelligence so as to improve on the system's capabilities for network intrusion detection. One potential area of research is login session modeling, which looks at collections of packets and detects suspect behavior patterns. This may be combined with a facility for user profiling. In effect, this is to implement policies that state that certain kinds of activities are permissible for certain users, but for other they are not.

In addition we plan to explore other aspects of network security, as well as other problem domains related to the nation's critical infrastructure. Such other domains, as described by the ARO program supporting this work, include transportation, communications, electrical power grids, and water supply systems.

These effort will entail devising new kinds of cases, together with appropriate similarity metrics. They may also entail building in capabilities for use interaction during case retrieval process along the lines of NaCoDAE [2, 3]. Our objective is to design a system for which this adaptation to new problem domains will be as painless as possible.

References

- [1] *Snort, The Open Source Network Intrusion Detection System*, <http://www.snort.org/>.
- [2] Aha, D.W., *Navy Conversational Decision Aids Environment*, <http://www.aic.nrl.navy.mil/aha>.
- [3] Aha, D.W., Breslow, L.A., and Munoz-Avila, Conversational case-based reasoning, *Applied Intelligence*, 14(1), 2001, 9-32.
- [4] Shimazu, H., A textual case-based reasoning system using XML on the world-wide web, in P. Smythe and B. Cunningham (eds.), *EWCBR'98, Lecture Notes in Computer Science*, Vol. 1488, 1998, 274-285.
- [5] Martin Fowler, M., *Analysis Patterns: Reusable Object Models*, Addison-Wesley, 1997.