

# Cognitive Approaches to Policy Based Intrusion Defence

Catriona M. Kennedy

School of Computer Science,  
University of Birmingham, UK  
C.M.Kennedy@cs.bham.ac.uk

## Abstract

This is a discussion paper on design options for the application of distributed cognitive agents to intrusion defence systems. Intrusion defence includes not only detection but also automated responses and proactive reconfigurations. If the system is to have some autonomy, it needs cognitive capabilities such as sensing, reasoning, planning and acting. Adaptation and focussing of attention are also critically important.

We present a general architecture for detection of intrusions as violations of security policies, where the policies can be defined on different levels of abstraction. Software agents are given responsibility to detect and respond to different kinds of policy violation. Such an architecture has two important requirements: first, the agents must work together to collectively provide the required cognitive capabilities and fault-tolerance; secondly, the high-level policy definitions must be translated into low-level conditions that can be detected by components of a biologically inspired adaptive system such as an artificial immune system. A reverse translation from low level conditions to new or more refined high level policies may also be possible as a result of adaptation.

**Key words:** adaptation, autonomic computing, cognitive agent, fault-tolerance, policy.

## 1 Introduction

In earlier work [10], we presented a cognitive approach to distributed intrusion detection (IDS) and demonstrated a proof-of-concept prototype which successfully detected policy violations in a simplified scenario. However, the architecture was not scalable and the distribution of the IDS was limited to fault-tolerant replication, with no specialisation of its components.

In this report we introduce a more general architecture in which specialised policies are applied on different levels of abstraction and different kinds of autonomous responses are applicable for each kind of policy violation. Software agents are assigned responsibility for different kinds of detection and response.

One important requirement of such an architecture is that the agents must be coordinated so that they act collectively as a coherent cognitive system. For example, they should participate in building a reliable picture of activities on the network (possibly from different perspectives) and their actions should not interfere with each other adversely.

A second requirement is the translation of high level policy specifications into constraints on low level configurations of components so that a violation can be detected very rapidly. In particular, we consider how to translate high level policies expressed in symbolic concepts into low-level signals that can be detected by an artificial immune system [2] or other biologically inspired adaptive system.

In the discussion, we assume for simplicity that the infrastructure is a wired network, but that the same principles would apply to a wireless network. In other words, a wireless network would only require the addition of further architectural features at most; it would not invalidate any of the principles discussed here.

We also assume that the policy based intrusion defence system is coordinated with other tools for detecting and perhaps responding to *misuse* (e.g. Snort) and that it will share information with these tools as necessary. The reason for including a policy-based system is that it should detect unknown intrusions, assuming the policy definitions allow for this.

## 2 Policy-Based Intrusion Defence

Ideally intrusions should be stopped before they begin their initial stages (e.g. stopping a scan or a Trojan from executing). Therefore if policies can be translated into conditions involving low-level events that can be immediately detected, this is clearly beneficial (e.g. disallowed port number, illegal write operation).

Similarly, decisions about policy violations should as far as possible be made locally by processes on a single host and with the minimum of data collection and process communication. Local policy enforcement which involves relatively simple checking of correctness and blocking of illegal activity may be classed as an intrusion prevention system (IPS).

In contrast to low-level policies suitable for IPS, a high level policy specification uses more abstract concepts and relationships (e.g. general rules about the structure of “conversations” between clients and servers, or about roles or capabilities of various processes). Such systems tend to be called “specification-based” intrusion detection and response (e.g. [24]). They are not normally classed as IPS because it would be difficult for them to react fast enough to prevent an intrusion. For example, they may use global correlation of multiple sources of data to provide more flexible responses (e.g. [14, 3]). They may also require detailed analysis to determine if abstract patterns defined in the policy are satisfied (e.g. [18, 11]). We will consider an architecture that will cover a wide spectrum of these sorts of policies. First, we need some concrete examples of what this involves.

### 2.1 Activities constrained by policies

The following are examples of activities that can be restricted by policies:

1. Communication activity (correct network usage): detected by firewalls, TCP wrappers, application wrappers. The following information may be used:
  - (a) connection information: e.g. those recently in “established” state (and who was the remote host, who initiated it etc.), ports listened on etc. Sensors must detect state of connections: e.g. Linux netstat;
  - (b) packet information: e.g. tcp syn, udp, icmp. Can be sensed (and logged) by packet sniffers as well as firewalls themselves.
2. Actions on a host: e.g. what processes should be running, what should be started and stopped in what context? What actions can a process do (file access, starting/stopping other processes)? What files should exist, what software should be installed? Information from sensors may include the following:
  - (a) system calls (e.g. using strace) for selected processes (open, read, write, network access etc.)
  - (b) process information: (whether running, newly started or recently stopped);
  - (c) critical data: installed software, existing files and their states.

Note that policy enforcement is not possible without sensors and effectors. The above information could be used directly by an IPS. The same sources of information may be used by an analysis system using high level policy specifications but the information may also be summarised. For example, it could include general statements about collections of events that appear to be related.

## 2.2 Why do we need analysis on higher levels?

The following question is sometimes raised: if we have different kinds of local policy enforcement (intrusion prevention) directly using the above kinds of information, why do we still need intrusion “detection”? In particular, since “detection” involves analysis on local and non-local levels, this itself can be vulnerable and error-prone. Furthermore, the results of the analysis are usually just logged (in the form of alerts) and frequently there are so many false-positives that an additional layer of analysis is needed.

Intrusion prevention should be used whenever appropriate. However, it has the following limitations:

1. It has to be fast. Therefore, it can only do very limited analysis. Consequently it can only really prevent something if there is a definite violation of a “strict” policy (involving a yes/no answer to a simple condition). It may be damaging to prevent or block an activity if:
  - (a) uncertainty exists as to whether a policy is violated, but there is some evidence that it is or it could happen in the future;
  - (b) only a “soft” policy has been violated (i.e. there is no compromise of integrity, confidentiality etc.), but just a kind of practice that is “insecure” or not “acceptable use”;

Both of the above are due to the “false-positives” problem, which is not overcome by IPS. It could even be more serious for IPS than for detection systems, since an IPS can be exploited as a denial of service tool (by causing it to repeatedly block harmless traffic).

2. Recognition of certain kinds of policy violation may require global knowledge of larger parts of the network which in turn requires significant data fusion and analysis. In particular, if an attacker first disables or misuses an IPS on the compromised host, some other part of the system on a different host may be required to detect this.
3. Even if all intrusions could be easily recognisable on the local host and could be stopped immediately, we still need information on what activities have been attempted, and to decide if any action should be taken (e.g. find more information on suspected attacker). In the case of uncertainty or “soft” policy violation, this information needs to be collected and correlated with activity going on elsewhere in the network. The information could also be used by an offline analysis system.
4. New policies can be acquired in a bottom-up fashion by observing features in the data and discovering new patterns. This could partly be done offline by generating more precise policies based on new information, or it could happen online using machine learning or adaptation techniques.

Consequently, IPS is insufficient on its own. Therefore we will talk about “policy-based intrusion defence” to include local and fast IPS (where the intrusion is actually prevented) and non-local and slower IDS (where some stages of the intrusion may already have begun but there is the possibility of limiting or reversing the damage).

### 2.2.1 Scenarios that require higher level analysis

When considering higher level policies, the following classes of scenarios will be kept in mind, as they will not be stopped by simple policy-based IPS:

1. Intrusion does not require any strictly illegal activity, but it could involve a number of “soft” anomalies that requires more global analysis. For example, a simple network IPS such as a firewall may be evaded if only legal port numbers and source/destination addresses are needed to achieve the intrusion goal. A buffer overflow could compromise a vulnerable service by

using legal operations only. The attacker can then use root privileges to read confidential data or modify critical files without activating any illegal communication patterns (e.g. the attacker may use ssh so that it looks as if a legitimate user is reading the data).

2. Intrusion requires illegal activity but is disguised to look legal. This requires a more complex policy, which may need reasoning about the global state of the network, while possibly keeping track of a sequence of events.
3. Intrusion involves strictly illegal activity but the local policy enforcers have been disabled. E.g. firewall rules or TCP wrappers may be circumvented. In order to do this, some critical configuration files may have to be changed, and this could be picked by an access control system or an integrity checking system. However, if the attacker has gained root access without leaving any tracks, they may just look as if they are doing normal administration tasks by updating configuration files. Circumvention may also happen without a trace if the firewall or other IPS was initially misconfigured in error.

### 2.3 Different types of security violation

In addition to the differentiation between levels of abstraction (as well as local and global states), we also need a differentiation between types and “severity” of the security problem - as can be seen from the above scenarios. Different levels of “strictness” may be defined as follows (in roughly descending order):

1. Definitions of confidentiality, integrity, availability and authentication; two levels:
  - (a) *Application domain level*: relating to business or organisation’s goals: e.g. confidentiality of personnel data, integrity of critical financial data, availability of company web server, authentication of staff and customers.
  - (b) *System administrative level*: e.g. “confidentiality” of restricted information (such as password file), integrity of critical software and configuration data, availability of OS and network services, authentication of system users.
2. Defensive security policies defining “good practice”: e.g. only actions that are *essential* for the business tasks may be allowed, and specific ways of doing things may be prohibited if they make an attack easier (e.g. an application may not be allowed to make external network connections after accessing a sensitive file).
3. “Soft” policies: e.g. acceptable use of resources, where violation is just a warning of a possible problem.

#### 2.3.1 “Least privilege” concept

To define administration level policies based on “least privilege” the following classification is useful:

- *Permissions*: set of activities essential by some people some of the time to keep the business operational: this has to be *allowed* by the policy. e.g. web browsing, system admin (including network scanning tools, intrusion detection *and* intrusion responses).
- *Obligations*: set of “always essential” activities; i.e. activities that *must* be done by any correct usage - but difficult for an attacker. Examples include authentication/passwords. Similarly a major intrusion response may require checks such as agreement between different agents before the response can proceed.
- *Not permitted*: set of “never essential” activities i.e. none of these activities are ever required to do the business tasks: This is the complement of the permitted activities and should *not* be allowed by the policy.

- Set of *known* intrusion types and scenarios. Although they can be detected by a misuse system, it is important to understand which policies they might violate, and why.

## 2.4 Policy refinement and adaptation

A violation of an administration level policy does not necessarily mean that an intrusion is happening, but the policies should gradually be refined so that it becomes increasingly unlikely that the policy is violated without malicious intent. The opposite should also be true: it should become increasingly difficult for an attacker to avoid triggering a policy violation.

We can summarise the following potential problems that can arise with policy-based IDS, along with mitigations (assuming the policies are based on “least privilege”):

- False-negatives: intrusions must involve some non-essential activity if they are to be detected by policy-based IDS. (E.g. if only certain port numbers are allowed, can an attack succeed by *only* using legal port numbers?) *Mitigation*: Reduce risk of non-detection by *refining* the policy so that sets of essential and non-essential activities are defined on a lower level (e.g. legal use of a port requires a certain timing pattern in the state transitions).
- False-positives: initial assumptions about essential activities are often wrong. Unexpected kinds of activity may turn out to be essential for the business tasks. The more complex the scenario (e.g. academic networks) the more likely this is to be true. Innocent and essential use of the network (e.g. to try out an unusual academic experiment) may violate policy, thus resulting in a false-positive. *Mitigation*: Policy definition must be a continually evolving and adaptive process. Specifically the set of essential activities will continually expand on a high level but on a lower level the definition of those activities become more precise with time.

Note that the mitigations may involve lower levels of detail or different kinds of abstraction.

## 2.5 Importance of Ontology in Policy Specification

To define a policy, an ontology defining the important entities and relations is needed. For example, it could include host, subnet, router, client, server, web server, mail transfer agent, etc. It is also important to define what kind of events are detected by sensors and how they can form temporal sequences and clusters of spatially correlated events.

Policy-based behaviour specifications can be defined for a subset of entities in the ontology. For example, the correct behaviour of a router or a mail server could be defined. Correct behaviour of security related software such as firewalls and IDS applications should also be specified.

The ontology and policy should focus on those components and interactions that are most critically depended on (and hence introduce the most vulnerability). Knowledge of such potential vulnerabilities will be incomplete but this can improve gradually through testing and operational experience. Simulation and rapid-prototyping can play a key role, and possibly autonomous adaptation by the system itself. Ontologies already developed for intrusion classification are also useful e.g. [14]. Although they are designed with misuse detection in mind, a classification of all known intrusions can help to identify the components and interactions that are typically targeted.

Policies may be described using temporal or deontic logic, in the form of simple pattern-based idioms such as Dwyer’s Property Patterns [7]. A high level policy language needs to contain the following:

1. Set of objects (subset of ontology above - an “object” may itself be a relation such as a “conversation”);
2. For each object:

- (a) its set of legal state transitions (e.g. using state machine formalism, and possibly temporal logic to specify constraints on timing);
- (b) what subjects are allowed to cause which state transitions on this object? In most cases, the object will also be a subject (unless it is passive data), in which cases it may make changes to itself.

This is just a generalisation of some classic security models such as those in [8]. The main difference is that we want an “operation” (i.e. state transition) to be more general than just “read” or “write” access. Depending on the ontology an operation could include such things as “restart a process” or “issue a challenge to a user”. Similarly, instead of talking about “security levels” (e.g. stipulating who has access to a document), the concepts of “permissions” (authority) and “obligations” (responsibility) of a subject relating to objects is more appropriate for an IDS involving responses and automated configuration changes.

The objects and relations in the ontology must also include the IDS components on various levels, including sensors and effectors. Similarly, critical data has to include any configuration files implementing security policies.

To summarise, a wide range of different kinds of analysis and response is required on different levels. Each of these levels requires a policy-based specification of acceptable states on that level.

### 3 Applying Cognitive Concepts to Distributed Intrusion Defence

A cognitive approach to intrusion defence was introduced in [10] and will be briefly summarised here. A *cognitive system* is an AI-based system in which different mechanisms and layers are integrated together to form a complete architecture for cognition (e.g. [13, 4, 22]). The minimal foundation of a cognitive system is an interaction between *sense*, *decide* and *act* components. We call this elementary unit an *agent* (similar to that in [17]). The sensing includes the sensors and the initial preprocessing and abstraction of sensor data (equivalent to perception); the *decide* component reasons about the abstracted data and makes a decision on any action to be taken; the *act* component transforms the decision into an action on the environment. We are interested in an architecture where the cognitive functionality is distributed among multiple agents (e.g. [13]).

The basic cognitive agent is similar to IBM’s autonomic control loop: MAPE-K (Monitor, Analyze, Plan, and Execute - plus Knowledge) [23]. Autonomic computing is divided into four areas: self-configuration, self-protection, self-healing and self-optimisation. Intrusion defence mostly fits into the self-protection category, but can also involve self-configuration.

Both the cognitive and autonomic paradigms are inspired by living systems with nervous systems. The main difference between an autonomic system and a cognitive system is that the former is usually focused on lower level regulatory aspects that human users are not interested in. In contrast, the cognitive tradition focuses on high-level human-oriented representation and interaction.

#### 3.1 Reactivity, Deliberation and Meta-Management

We use the Birmingham H-Cogaff architecture [21] as a foundation. Key concepts in this architecture have the following meanings in the context of intrusion defence:

*Reactivity* means that a decision depends only on information that is immediately available from the sensors and the required action sequence can be executed immediately (it does not depend on planning). Reactive systems are fast but inflexible (“unthinking”).

*Deliberation* means that a decision depends on hypothetical reasoning and anticipated results. It is not immediate as in a reactive system. Deliberation is important because it is often desirable to classify an intrusion and anticipate its potential impact before selecting a response. The response should then

be finely tuned in accordance with the perceived severity or category of attack. Planning algorithms may be used to construct a sequence of actions. The response selection component should also take into account potential negative effects of a response. This is particularly important if an intrusion is wrongly believed to be taking place, or it is categorised inappropriately. Due to this “false positives” problem, the system must take into account the potential impact of a response based on a wrong diagnosis (for example, would the blocking of legitimate users or processes have the effect of a denial of service?)

*Meta-management* means that an intrusion defence system is included within the system it is protecting. Therefore it must be self-monitoring and potentially self-modifying - in the same way as it monitors and modifies the states of other objects it is protecting (such as applications, servers etc.) There is also some overlap with the requirements of autonomic computing [23], Meta-management is needed to resist attempts to compromise the IDS (subversion resistance), and to periodically reconfigure the IDS on the assumption that attempts are being made to subvert or evade it.

### 3.2 Perception and Action Hierarchies

The H-CogAff architecture is based on a generic schema (CogAff) made up of three “towers”: perception hierarchy, analysis, and action hierarchy. A simplified version of this schema is shown in Figure 1. We have also included the meta-management and deliberation layers in this diagram. The

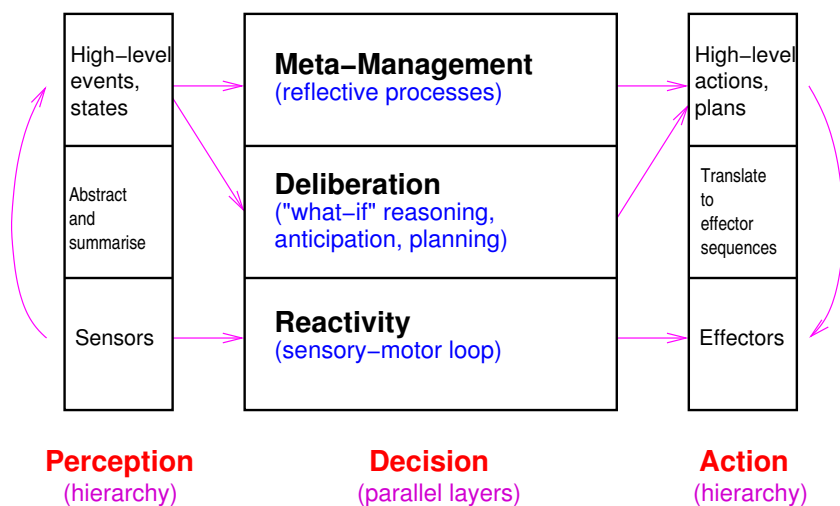


Figure 1: The CogAff Schema - simplified version

perception and action towers have the following interpretations in the domain of intrusion defence. Each has two main components.

Perception hierarchy:

1. Detect low-level events: e.g. SYN packet sent, system call by an application to read a file.
2. Translate recent snapshot of low-level events (or short event sequences) into a high level “state of the world” expressed using the ontology. For example, a “SYN packet sent” event may be associated with a client program (subject) that is in the process of requesting information (object1) from a server (object2). The server’s response will reverse the subject/object roles.

Action hierarchy:

1. Select high level action (based on ontology). This could be a simple table lookup or it could involve planning.
2. Translate high level action into an effector activation (or sequence of effector activations). For example, this may involve a sequence of simple operations such as stopping some processes that may be compromised, deleting files they have created, re-initialising their configuration files and restarting the relevant processes.

The analysis tower is divided into reactive, deliberative and meta-management layers. The perception/action hierarchy is independent of reactivity and deliberation. For example, a deliberative or meta-management layer may reason about low-level events as well as high level. Similarly a reactive layer may select an immediate response to a high level state. However, the deliberative layer will in most cases use information that is already translated into a high level ontology and the reactive layer will typically respond immediately to low-level events (e.g. based on a policy that is formulated on a low level) before these events are fused together and translated into the ontology.

### **3.2.1 Agent actions and responsibilities**

The agent should select actions that move the system state closer to a satisfaction of the policy requirements and should not have adverse side effects on the service delivery. The agent's actions can be applied to any object that it has authority over (usually its domain of responsibility) which could be the whole network or just a single host or a single application. Actions against a suspected attacker are assumed to be defensive only and may, for example include redirection to a honeypot or slowing down a connection or process while gathering information on it.

Meta-management is required to ensure that the IDS components are also subject to policy requirements. In contrast to deliberation, actions are internally directed (e.g. interrupt a response sequence that is having unintended negative side effects).

### **3.3 Mapping the Architecture onto a Distributed System**

The diagram in Figure 1 can be interpreted as a single multi-layer agent or it can involve multiple agents. We are applying it to a multi-agent system, which should act as a distributed intrusion defence system.

The simplest form of distribution is hierarchical. If we return to multi-level policy specifications, we can define agents to do specialised policy-checking on the different levels. The fast IPS decisions are suited to agents that are mostly reactive, while the higher level policy analysis requires deliberation. A reactive agent may act as an IPS if it responds externally to stop the intrusion.

To map the multi-agent system onto a physically distributed system, we have to think about where they are placed with respect to the different hosts and subnets. Agent roles in a network can be divided as follows:

1. Reactive local agents (single-host only) which monitor one local source of data: if one condition is satisfied (misuse-based) or violated (policy-based) then respond locally if appropriate and send an alert to next higher level.
2. Deliberative local agents that do some detailed analysis (e.g. reasoning about uncertainty): one kind of agent may fuse together different data sources from one host (e.g. application system calls, open port numbers etc.); other kinds may just be focused on a single source (e.g. an untrusted process). A response may involve the planning of a sequence of steps but is only applied locally.
3. Non-local agents with fusion of different data sources from different hosts (or different subnets). Their responses may also involve some non-local aspects (e.g. broadcasting to local

agents, telling them how to respond). The non-local agents may also include varying degrees of reactivity and deliberation but are expected to be more deliberative, and only respond occasionally when necessary. (This is to minimise use of the network by the IDS, thus minimising the inefficiency and added vulnerability associated with this).

“Local” does not necessarily mean low-level or reactive. A local IDS agent may do some abstract reasoning in the background, but its source of information is limited to one or more applications or services on a single host. Its domain of responsibility is also local (i.e. it can only change things that it knows about). “Global” usually has to be abstract (involving summarised statements about events rather than the actual concrete events).

The local reactive agents should also act as “sensors” for the higher level agents, passing on summarised reports of events and indicating whether they were legal or not.

## 4 Distributed Meta-Management and Fault-Tolerance

We now consider how to implement meta-management in an intrusion defence system. If we return to the CogAff schema in Figure 1, the meta-management layer looks like an additional layer of abstraction. However, this is not necessarily true. Figure 2 shows an alternative representation of CogAff as a “reflective” agent. The “meta-level” monitors and intervenes in the system internally, while the “base level” interacts with the external environment. The meta-level may have its own independent levels of abstraction, along with reactivity and deliberation. The meta-level plays the role of meta-

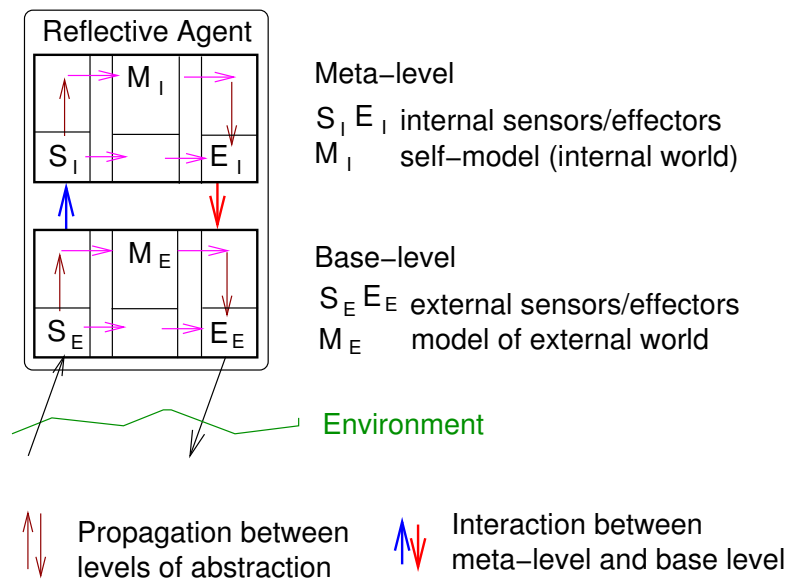


Figure 2: An alternative representation of CogAff showing a meta-level

management and allows the agent to detect policy violations in the behaviour of its own components and to intervene to correct them if possible. It may also intervene in other ways (e.g. interrupting any unproductive analysis process).

As mentioned earlier in Section 2.5, the policy definitions must include the IDS components along with other objects in the ontology. This means that all layers of the cognitive agent also require an acceptable behaviour specification (possibly as a set of constraints) to allow potentially hostile behaviour to be detected. For example, if the agent’s response planning has been compromised, it should deviate from its correct behaviour specification based on the policy (assuming that the policy is

well designed). The agent’s meta-management should detect the potentially damaging behaviour and intervene to stop it. However, if the meta-management itself is compromised, there is no independent monitoring system to detect this, although the meta-management should also have a correct behaviour specification.

Distribution is necessary to avoid a “single point of failure” as well as to enable independent monitoring of the meta-management layer. More details are in [9]. Figure 3 shows distributed meta-management with two agents. Distribution allows the agents to “reflect” mutually on each other’s decisions and responses in addition to their own. In practice, it may be difficult to implement a “pure” reflective system since it requires an agent to have the same degree of access to another agent’s reasoning as it has to its own. This is possible in experimental agent toolkits such as SimAgent [20] but unlikely in a real world system. However, “external” forms of monitoring may be sufficiently effective. It all depends on the ontology and policy definitions. Figure 4 shows a three agent version.

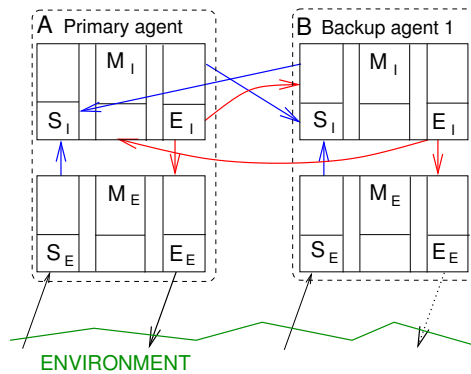


Figure 3: Distributed meta-management - two agents

In both figures 3 and 4, the primary agent is the one that can act on the environment. The backup

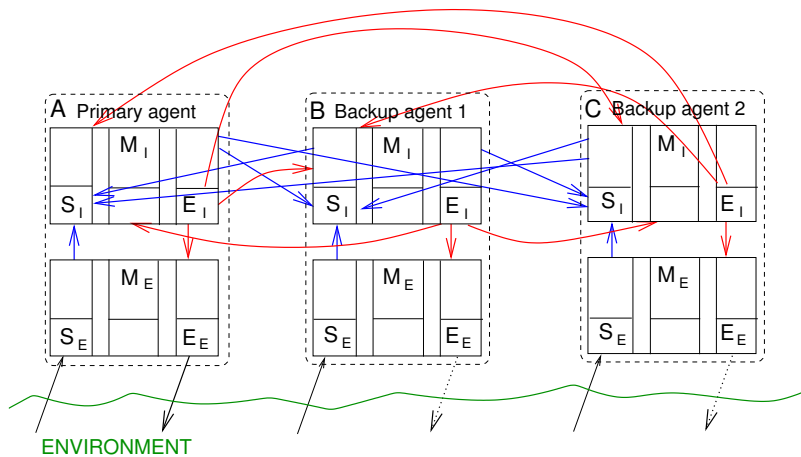


Figure 4: Distributed meta-management - three agents

agents only simulate the actions internally and anticipate the results. The simplest form of distribution is replication, which we discuss in the next section.

## 4.1 Design steps

In a system with distributed meta-management, the “perception hierarchy” includes the following types of connectivity:

- hierarchical, one-way connections: distributed sensing from around the network; data fusion and summarisation to produce assertions about the state of the network and events in it.
- non-hierarchical mutual monitoring and checking between agents to ensure the reliability of the information being summarised and propagated. (There are also other ways of promoting reliability, such as sensor redundancy and diversity).

The distribution required for mutual monitoring can be divided into the following categories of design:

1. Single agent at the highest level: this means that there is only one way of fusing the data from different parts of the network and reasoning about it; there are two possibilities:
  - (a) agent is not replicated: simple hierarchical system with one agent always in control - this is the simplest case with no distributed meta-management (shown in Figure 2);
  - (b) agent is replicated, with mutual monitoring and protection between replicated versions (Figures 3 and 4).
2. Multiple diverse agents at the highest level: each agent represents a different way of interpreting the data and reasoning about it. Agents monitor each other and replicated groups may exist. Two possibilities:
  - (a) Exclusive domains: Exactly one agent (or group) is responsible for one domain;
  - (b) Overlapping domains: Each agent or group is responsible for a domain but may share responsibility for all or part of it with other agents;
3. Dynamically changing responsibility and authority relations.

These subdivisions provide a design trajectory (sequence of steps) to build a subversion-resistant system. Design 1(a) can be the first step. Although it is not subversion-resistant in itself, it provides an essential building block. The next step is to replicate the highest level agent (as in 1(b)) and to develop the connectivity required for mutual meta-management. Design 1(b) involves one viewpoint of the network with dependency on a single software design. The lack of diversity in 1(b) means that a weakness in one method used by an agent can also be exploited in all others. Therefore the third step is to add diversity (2) to the replicated system in 1(b). Finally, the static system in 2(a) and (b) is allowed to change dynamically.

This is only one suggested design trajectory. Note also that “development” of connectivity in 1(b) and the dynamically changing responsibility assignments in 3 may involve some degree of learning as well as software design. Similarly the introduction of diversity may be partially automated (e.g. with evolutionary methods).

We look at these stages in detail.

### 4.1.1 Single high-level agent: non-replicated

In the purely hierarchical system of 1(a), we only have one deliberative agent with a single global view of the network. Then the reactive agents can act as sensors and effectors for this higher level agent. The reactive agents will already have responded by the time their information is received by the deliberative agent. If the higher level agent’s selected actions differ from those of the reactive agent, the deliberative responses will override them. (The higher level agent could also tell it to start or stop responding). The meta-management in this version corresponds to Figure 2 and has the same limitations.

### 4.1.2 Single high-level agent: replicated

Replication allows the meta-management layer to be distributed as in Figures 3 and 4, thus removing the single point of failure. Replication makes the assumption that agents have a degree of independence of each other so that if one is compromised, the other will remain intact for at least long enough to detect and respond to a problem.

Autonomous action becomes more difficult with replicated agents. So far we have only considered how to add autonomous action and responses to a single agent. The action hierarchy in a cognitive agent assumes a single top-level decision-maker. In a non-hierarchical system, the actions must be coordinated.

Replicated agents monitor each other in addition to the other objects in the network. The information from the reactive agents may be broadcast to all the replicas, enabling them to share the same information (although there may be some specialisation so that they do not completely share information sources).

Clearly, the control of a reactive agent's responses cannot be shared in this way. The simplest solution is that a single agent in the replicated group has "response privileges" at one time. This means that it controls all the reactive agents sending it data. This is a classic fault-tolerance solution and the agent that has control is often called the "leader". In Figures 3 and 4 the terms "primary" and "backup" are used (where the primary agent is the "leader").

The agent's tools for monitoring and reasoning may be integrated with existing tools for fault-tolerant distributed communication and agreement [15, 5, 12, 16]. For example, an action might only be permitted if there is a majority vote in favour of it. Interventions by meta-management would be governed by the same principle, except that the actions are directed at the IDS itself.

At any one time, a replicated group has a leader with authority to respond to intrusions or to periodically reconfigure the system (this can be integrated with the "self-configuration" role of an autonomic system). The entities being proactively reconfigured will include components of the IDS itself (as part of the meta-management function). Such actions are particularly important to increase subversion resistance. A simple mechanism for this may be implemented using a rotating leadership scheme, where the current leader gives up its leadership to a randomly selected agent after a random interval. Additionally, reactive agents may be restarted at random intervals. If an attacker uses malicious code to compromise an agent, this kind of self-reconfiguration may be difficult to emulate without obstructing or slowing down the intended intrusion (due to the use of randomness and giving up of power to a non-compromised agent).

Note: The distributed "meta-management layer" is a reflective layer within each replica agent. However, a separate layer of "meta-level" agents may be appropriate in some cases. The meta-level agents would then *only* monitor each other, as well as the base level agents below them; they would not monitor other objects. This solution adds more complexity and hence potential vulnerability.

### 4.1.3 Multiple diverse high-level agents

In the above variants 1(a) and 1(b), specialisation only occurred on the level of the low-level agents. In a scalable architecture we need specialisation on multiple levels. We cannot just have a single replicated agent at the top, since there might be hundreds of low-level agents or sensors sending all replicas the same input (for example if one or more sensor is running on each host).

In addition, pure replication is not very subversion-resistant, since a vulnerability in one agent can be exploited in all. Replication merely makes disrupting the IDS more difficult because all agents would have to be subverted simultaneously (or at least faster than the time it takes for other replicas to detect any problem).

Therefore, instead of agent replicas looking at the same data and using the same ontology to describe the network, the agents can be specialists monitoring different kinds of data and possibly using different ontologies (2(a)).

Specialist agents may themselves be replicated. In that case, we talk about a specialised agent *group* instead of just a specialist agent.

One form of specialisation is to divide the network into spatial partitions. Each agent (or replicated group) is then responsible for its partition but may also monitor some aspects of other partitions. The following are important design issues:

1. how to divide the network into partitions: a partition may be a collection of hosts or applications on a single host for which an agent group is responsible and from which it receives data.
2. how to specify the agent/sensor connectivity (which also implies agent/effector connectivity): The default is full connectivity between all sensors collecting data on a partition and the agent group assigned to it.

The simplest architecture is a hierarchy of levels, in which agents at one level act as sensors for those at the level above. For example, we may have a reactive layer at the lowest level, specialist agent groups on an intermediate level and a top level agent group to fuse together information on the whole network and interact with the administrator.

#### **4.1.4 Combining redundancy with diversity**

If there is no overlap between the domains of responsibility of the specialist agents, they have no redundancy and hence less fault-tolerance. If there is an overlap between the domains (2(b)), the specialist can provide multiple viewpoints of the same objects. Multiple agents can monitor the same object using different sensors or they can interpret the same sensor data in different ways.

Partitioning and specialisation increases the diversity and thus creates more fault-tolerance. There is also more coverage of events, making it more difficult for intrusions to avoid detection (assuming that we have appropriate policies that specify the acceptable form of these events). The disadvantage is that the development of such software is complex and time-consuming. This problem may be partially solved by exploiting the natural diversity and fault-tolerance inherent in biologically inspired algorithms.

#### **4.1.5 Dynamically changing partitions and connectivity**

The partitions and connectivity need not be static. For subversion-resistance it is best if it continually reconfigures and evolves. There are several possibilities:

- The agents could agree to modify their partition allocation periodically. For example, a group leader may give up control of some of the objects under its responsibility and “offer” them to another randomly selected agent group.
- Instead of full static connectivity between higher and lower level agents (e.g. for perception and action), some agents or sensors may be randomly switched on and off so that the configuration for a given partition is continually changing.

#### **4.1.6 Dynamically changing authority relations**

The agent with the most authority is by default the group leader on the highest level and should also be subject to the most scrutiny by other agents. In addition to its monitoring by replicated agents within its group, it may be monitored by selected agents in lower level groups. If they disagree with the top-level agent’s actions their “opinion” may have some weight in determining the final vote in support of the action.

Instead of having a static framework of intergroup meta-management in which the same agents and groups are always participating, the agents whose votes are taken onto account to support an

action may be periodically changing. As with rotating leadership, a participating agent may give up its own privileges to a randomly selected agent.

#### 4.1.7 Summary of design options

The above design options can be successive stages of an evolutionary path to a subversion-resistant system. It can be characterised as a trajectory through a multi-dimensional *design space* [19]. Some transitions between stages may be assisted by autonomous adaptation (such as learning or evolutionary computation). Table 1 summarises the trajectory.

	Distributed?	Diversified?	Multi-viewpoint?	Dynamic relations?
1(a): Non-replicated	No	No	No	No
1(b): Replicated	Yes	No	No	No
2(a): Exclusive domains	Yes	Yes	No	No
2(b): Overlapping domains	Yes	Yes	Yes	No
3: Self-reconfiguring	Yes	Yes	Yes	Yes

Table 1: Design trajectory from a single agent to a distributed subversion-resistant system

The distributed self-configuring architecture can be applicable to any situation in which a reliable global picture of a complex system is required, and where multiple viewpoints of the same data may be beneficial (not just limited to intrusion defence). It is particularly useful where masses of data have to be fused and summarised to fit into ontological concepts, and where serious errors or security problems are possible (such as fraud or identity theft).

The main disadvantage of the self-configuring architecture is the added complexity which may introduce new vulnerabilities, for example due to unintended interactions between agents. One way of improving robustness is to ensure that the system gradually evolves, with progression to the next stage only when the previous one is established and fully trusted. Another is to limit autonomous intrusion responses to defensive reconfiguration and increased monitoring or isolation of the suspiciously behaving processes. Autonomous reconfiguration actions will be necessary and are also a part of established research in autonomic computing.

## 5 Translation of High-Level Policies to Low-Level Conditions

As mentioned earlier in Section 2, intrusion detection should as far as possible be done locally and with minimal processing, with only the “difficult” situations being handled at the higher levels. Therefore we need to translate the high level policy requirements into low-level conditions that can be detected rapidly and processed locally (with minimum requirement for correlation and fusion from different sources).

This translation can be defined as a configuration process and incorporated within the “self-configuration” component of autonomic computing. Some degree of automated configuration is essential because it is extremely error-prone at low levels and its consistency with policy requirements has to be guaranteed.

Components that can be configured by an autonomic management system include the following:

1. Services on different levels (web servers, mail servers, applications, network management tools)
2. COTS security products (firewalls, COTS IDS such as Snort, access control systems (e.g. NFS).
3. Low-level reactive agents that are part of the IDS;

#### 4. High level IDS agents.

(3) and (4) are components of the IDS system. In particular, (4) is part of the autonomic management system itself (although concerned with self-protection).

### 5.1 Autonomic architecture

The policy translation process from abstract requirements to simple conditions is a configuration of the low-level reactive IDS agents, which may act both as IPS systems and sensors for the high level agents. (Note that “reactive agent” is used as a generic term here and may include passive sensors and agents with some analysis capability in addition to the “IPS” type agent). Configuration of the high level IDS agents may also be automated in this way.

We will assume that the configuration is done by a “configuration agent” which is closely coupled with the high level IDS agents and shares information with them. Consequently, an agent with responsibility for configuring a set of IDS agents may also know about alerts and other information from these same agents and it can use this information as necessary.

In a real-world autonomic system it is likely that the self-configuration functions will be separate from the self-protect functions (intrusion response). However, it is important that they are closely coupled and that they can share information. In our conceptual architecture based on the CogAff schema, it is useful to combine the intrusion responses with the configuration functions, since the actions required by both are often similar and the same software components are likely to be used. We can think of both functions as belonging to the planning and action hierarchy of Figures 1 and 2, except that there will be a human in the loop in most cases.

The ability to vary the autonomy of the system is important. One extreme is a fully automated system which is entirely self-configuring and self-protecting, but with the ability to allow user requests to override as necessary. On the other end of the spectrum, all intrusion responses and configuration changes are requested by a human user. The longer term objective is that the system should be able to initiate responses and adaptive configuration changes of its own when this is required.

#### 5.1.1 Dynamic attention focusing

The close coupling of intrusion defence with configuration means that dynamic configuration changes may be possible in the form of attention focusing. If a high level IDS agent detects some “soft” policy violations indicating that an intrusion may be beginning, it can direct the configuration agent to increase the level of alertness of the reactive agents. Similarly, there may be uncertainty about whether an intrusion is in progress due to insufficient information. The reactive agents (as well as passive sensors) may be temporarily re-directed so that they collect more data from sources that are most relevant. (E.g. a firewall could log more detailed information on packets that appear to come from a particular source).

In addition, the low-level conditions that can trigger alerts may be made more “strict”. For example, a firewall log entry that would normally not be significant can be meaningful in the context of an intrusion. Therefore a log listener could be configured to raise an alert, which normally it would ignore. Such dynamic focusing of attention (re-allocation of resources) can be regarded as a temporary change in policy, requiring a temporary configuration change. This is also a kind of intrusion response.

### 5.2 Learning and adaptation

As discussed in Section 2.4, an initial policy specification will probably not define perfectly the boundary between “intrusion” and “non-intrusion” because of false negatives (intrusions that do not violate the rules) and false positives (due to unintended violations of policy caused by innocent behaviour). Therefore it is important to refine the policies or learn new ones by taking account of the context of different kinds of policy violations.

### 5.2.1 Example: Integrating an immune system into the autonomic architecture

An artificial immune system can learn to recognise simple conditions that are expected to be hostile. Such conditions are the occurrence of strings belonging to the “nonself” set. A “string” can be an activity pattern such as a short sequence of system calls. The discrimination between “self” and “nonself” strings can be learned using the negative selection algorithm [6]. One problem with negative selection is that it is purely *pattern*-based (i.e. normal patterns vs anomalous). There is no semantic information on whether a security requirement is being violated. Therefore the false positive rate is often higher than it is for policy-based IDS.

However, many features of an immune system are valuable for enhancing security. In particular, its adaptive capability is important as well as its intrinsic fault-tolerance due to the distributed nature and inherent diversity of its component cells. Recent work by [1] is based on the Danger Model of immunology and aims to overcome the limitations of the early artificial immune systems. In the Danger Model an immune reaction only happens if there is a “danger signal” present in addition to abnormal patterns. A danger signal is a “distress signal” sent out by a cell that is being destroyed by a pathogen.

A danger signal can be defined as a violation of a condition required by a security policy. Therefore it is possible for the policy-based IDS agents defined in previous sections to send danger signals to the immune cells of a Danger Model. Different classes of “danger” can be defined, which may correspond to the three different levels of policy defined in Section 2.3.

In order for the Danger Model cells to learn new policies, they will also need details of the context in which a condition was violated (or satisfied). This information need not be very different from the information that is sent to higher level agents in the autonomic architecture. The immune system is just an additional interface. New policies may be suggested by the immune system to the configuration agent, which is also interacting with the user. The configuration agent may then execute configuration change requests that implement the new policy, which may again be subject to refinement in the same way.

## 6 Summary and Conclusions

A layered cognitive architecture such as CogAff can be very useful in policy-based intrusion defence because data fusion and analysis (a process similar to perception) is often required, as well as deliberative planned responses (Section 2.2). It is not enough just to have local intrusion prevention systems. Moreover, the focusing of attention on relevant sources of data is important, as is adaptation to changing situations.

To enhance fault-tolerance, cognitive functionality can be provided collectively by multiple agents distributed over a network. In particular, distributed meta-management is necessary for subversion resistance. Connections between agents include those that are hierarchical (data-fusion upwards and configurations/directions downwards) and symmetric (mutual meta-management and information sharing). In addition to the architecture itself, the *design trajectory* by which the architecture evolves is also important. We have considered a simple sequence of design steps towards a subversion-resistant system. One of the main uncertainties that needs further investigation is the possibility of unintended side-effects from interactions between agents.

The CogAff architecture can be implemented as an autonomic system in which self-protection and self-configuration are closely coupled. The self-configuration can involve focusing of attention on the source of a suspected threat or it can respond to a configuration change request as a result of a policy modification. An autonomous response to an intrusion can also be a reconfiguration.

Translation from high level policies to low-level conditions is important in order to exploit the potential of biologically inspired adaptive systems. The policies first have to be translated into configurations of reactive agents that can check if the conditions are being violated. This belongs to the self-configuration component of an autonomic system. As an example, we have considered the

interface between the autonomic system and an artificial immune system.

## Acknowledgements

This research was supported by the Paul and Yaunbi Ramsey Fund. I would also like to thank Uwe Aickelin and Jamie Twycross of Nottingham University for their explanation of the Danger Model of the immune system.

## References

- [1] U. Aickelin, P. Bentley, S. Cayzer, J. Kim, and J. McLeod. Danger Theory: The Link between AIS and IDS? In *Proceedings ICARIS-2003, 2nd International Conference on Artificial Immune Systems, LNCS 2787*, pages 147–155, Edinburgh, UK, 2003. Springer-Verlag.
- [2] Uwe Aickelin and Steve Cayzer. The Danger Theory and its Application to Artificial Immune Systems. In *First International Conference on Artificial Immune Systems*, University of Kent, Canterbury, UK, September 2002.
- [3] Ivan Balepin, Sergei Maltsev, Jeff Rowe, and Karl Levitt. Using Specification-Based Intrusion Detection for Automated Response. In *Sixth International Symposium on Recent Advances in Intrusion Detection (RAID 2003)*, Pittsburgh, PA, USA, September 2003.
- [4] Luc P. Beaudoin. *Goal Processing in Autonomous Agents*. PhD thesis, University of Birmingham, 1994.
- [5] Flaviu Cristian. Understanding fault-tolerant distributed systems. *Communications of the ACM*, 34(2):56–78, 1991.
- [6] D. Dasgupta and S. Forrest. Novelty-Detection in Time Series Data using Ideas from Immunology. In *Proceedings of the International Conference on Intelligent Systems*, Reno, Nevada, 1996.
- [7] Matthew B. Dwyer, George S. Avrunin, and James C. Corbett. Patterns in Property Specifications for Finite-state Verification. In *Proceedings of the 21st International Conference on Software Engineering*, May 1999.
- [8] Dieter Gollmann. *Computer Security*. Wiley, Chichester, England, 1999.
- [9] C. M. Kennedy. *Distributed Reflective Architectures for Anomaly Detection and Autonomous Recovery*. PhD thesis, University Of Birmingham, Birmingham, UK, July 2003. Available online at:  
<http://www.cs.bham.ac.uk/research/cogaff/0-INDEX03.html#03-03>.
- [10] C. M. Kennedy. Distributed Intrusion Detection based on Cognitive Architectures, 2004. Technical Report CSR-04-13, School of Computer Science, University of Birmingham.
- [11] Calvin Ko, Paul Brutch, Jeff Rowe, Guy Tsafnat, and Karl Levitt. System health and intrusion monitoring using a hierarchy of constraints. In *Recent Advances in Intrusion Detection (RAID 2001)*, LNCS, Davis, California, October 2001. Springer.
- [12] Jean-Claude Laprie. Dependability - Its Attributes, Impairments and Means. In B. Randell, J.-C. Laprie, H. Kopetz, and B. Littlewoods, editors, *Predictably Dependable Computing Systems*. Springer Basic Research Series, 1995.
- [13] Marvin Minsky. *The Society of Mind*. Simon and Schuster, New York, 1986.

- [14] Maria Papadaki. *Classifying and Responding to Network Intrusions*. PhD thesis, University of Plymouth, 2004.
- [15] D. Powell, G. Bonn, D. Seaton, P. Verissimo, and F. Waeselynck. The Delta-4 Approach to Dependability in Open Distributed Computing Systems. In *Proceedings of the 18th International Symposium on Fault-Tolerant Computing Systems (FTCS-18)*, Tokyo, Japan, 1988. IEEE CS Press.
- [16] David Powell and Robert Stroud. Malicious- and Accidental-Fault Tolerance for Internet Applications (MAFTIA) – conceptual model and architecture, 2003. Project deliverable - D21.
- [17] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall International, Englewood Cliffs, NJ, USA, 1995.
- [18] R. Sekar, A. Gupta, J. Frullo, T. Shanbhag, A. Tiwari, H. Yang, and S. Zhou. Specification-based anomaly detection: a new approach for detecting network intrusions. In *Proceedings of the 9th ACM conference on Computer and communications security (CCS'02)*, pages 265–274, 2002.
- [19] A. Sloman. “Semantics” of Evolution: Trajectories and Trade-offs in Design Space and Niche Space. In Helder Coelho, editor, *Progress in Artificial Intelligence*, Lecture Notes in Artificial Intelligence, pages 27–38. Springer-Verlag, 1998.
- [20] A. Sloman and R. Poli. SIM\_AGENT: A toolkit for exploring agent designs. In Joerg Mueller Mike Wooldridge and Milind Tambe, editors, *Intelligent Agents Vol II, Workshop on Agent Theories, Architectures, and Languages (ATAL-95) at IJCAI-95*, pages 392–407. Springer-Verlag, 1995.
- [21] Aaron Sloman. Varieties of affect and the cogaff architecture schema. In *Symposium on Emotion, Cognition, and Affective Computing at the AISB'01 Convention*, March 2001.
- [22] Aaron Sloman and Brian Logan. Building Cognitively Rich Agents using the SIM\_AGENT Toolkit. *Communications of the Association of Computing Machinery (CACM)*, 43(2):71–77, March 1999.
- [23] Daniel H. Steinberg. What you need to know now about autonomic computing, parts 1 and 2 at: <http://www-106.ibm.com/developerworks/ibm/library/i-autonom2/>, Last consulted: February 2005.
- [24] Prem Uppuluri and R. Sekar. Experiences with specification-based intrusion detection. In *Recent Advances in Intrusion Detection (RAID 2001)*, LNCS, Davis, California, October 2001. Springer.