# An Architecture for Anomaly Detection in Large Complex Critical Infrastructures

David Gamez
Department of Electronic Engineering
Queen Mary, University of London
London, E1 4NS
david.gamez@elec.qmul.ac.uk

John Bigham
Department of Electronic Engineering
Queen Mary, University of London
London, E1 4NS
j.bigham@elec.qmul.ac.uk

Laurie Cuthbert
Department of Electronic Engineering
Queen Mary, University of London
London, E1 4NS
laurie.cuthbert@elec.qmul.ac.uk

## Abstract

Deregulation, cyber-terrorism, and increased interdependency are making large complex critical infrastructures, such as the telecommunications and electricity networks, increasingly vulnerable. Solutions are needed that can provide a rapid automatic response to the known and unknown dangers that threaten them today. This paper outlines the work that is being done at Queen Mary, University of London on the design of an agent-based anomaly detection and repair system that will address this problem. This will build up a model of normality for the telecommunications management network, interact with existing protection mechanisms, diagnose problems and carry out self-healing. The layered safeguards that will be offered by this system will substantially increase the survivability of large complex critical infrastructures in the face of attacks, failures and accidents caused by insiders and outsiders.

## 1. Introduction

Recent increases in physical and cyber terrorism have placed large complex critical infrastructures, such as the electricity and telecoms networks, increasingly under threat. Furthermore, the deregulation of these industries has led to a mobile workforce that is less familiar with its working environment and more likely to take revenge if cost cutting leads to widespread redundancies. All of these problems are compounded by the increasing dependency of large complex critical infrastructures on each other and on the Internet. A major failure in telecommunications is now capable of triggering cascading failures in many other industries.

Safeguard [13] is a European project that is investigating new ways of protecting large complex critical infrastructures against attacks, failures and accidents. It is currently developing a system that will detect anomalies in these infrastructures, interact with existing applications and provide self-healing mechanisms. There has already been a great deal of research done into ways of increasing the reliability of the hardware components of large complex critical infrastructures. Safeguard will not be covering this area and will concentrate instead on the new vulnerabilities that are emerging within the data gathering and control parts of the management networks. Attacks and failures here are the most likely and have the greatest potential for widespread damage. These management networks consist of monitoring and control hardware and software in a constant state of change as adjustments are made to the network. Mechanisms are needed which will detect unusual conditions and automatically repair and defend the network against malicious insiders and outsiders, operator mistakes and hardware and software faults.

A lot of work has been done on the explicit detection of faults, viruses and attacks, based on their signature. However this approach cannot identify unknown dangers and frequent updates are necessary. More recently, research has been investigating how normality can be defined for a system so that an alarm can be raised whenever there is a significant deviation from normality. This can create problems with false alarms, but has the advantage that it can detect and respond to new faults, accidents and threats. Within the deregulated telecommunications industry, information about the normal state of the network could also be extremely useful to network operators who may not have wide experience and need to cope with new technology, regulations and patterns of demand.

At present, most intrusion detection systems are based around the so-called ‘fortress’ architecture. The system is surrounded by barriers with

strict control policies and behind these barriers the whole network lies open. The problem with this approach is that it offers no protection against malicious insiders.[1] One of the Safeguard objectives is to increase the survivability of large complex critical infrastructures by creating a protection system that will work even when an attacker is inside. This will not be an extension of the fortress model with better tools, but a new technology that will build Safeguard defences into the system from the bottom up. Every application and process within the management network will be watched for abnormality and it will be very difficult for a hostile person to damage the system without causing significant deviations at this level. This low-level monitoring will be capable of operating independently of the higher levels to further enhance the robustness of the system.

Although the Safeguard system will be tested on both electricity and telecommunications networks, this paper will use the telecommunications management network to illustrate the problems and proposed solutions. The main focus of this paper will be on the work that is being done at Queen Mary, University of London, on the development of anomaly detecting agents.

## 2. The Telecommunications Management System

A typical telecommunications management system consists of a number of interconnected computers running server, database, firewall and monitoring and control software. Figure 1 shows a typical example, with the different functionalities that are being performed and the connections with the telecommunications network, the corporate network and the Internet. A small telecommunications network could be controlled from one network of this type housed within a single control centre. Larger telecommunications systems will require several control centres.

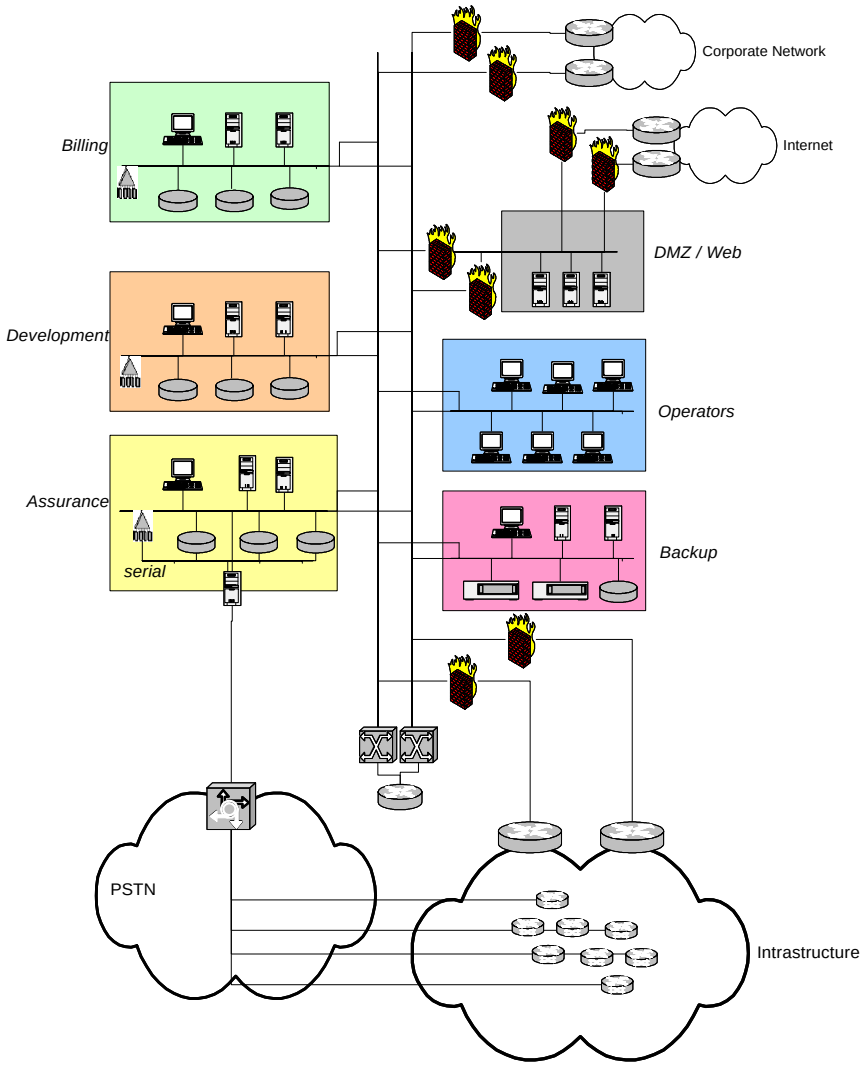


**Figure 1: The telecommunications management network**

[1] The most recent dti survey [6] reports that 48% of large businesses blame their worst security incident on insider activity.

Although this system contains firewalls, virus scanners, diagnostic software and intrusion detection systems it remains vulnerable to a number of attacks, failures and accidents.

### 2.1 Threats to the management system

Some of attacks, failures and accidents that threaten the telecommunications management network are as follows:

*Attacks*

- An attacker enters the system by using a buffer overflow exploit on servers connected to the management network. This enables him or her to execute any command on the server which can then act as a launching point for attacks against the rest of the network.
- A virus loaded in software (perhaps a new virus accidentally incorporated into a security patch) spreads across the management network. This could install a rootkit or delete critical system files.
- A Unicode parsing error gives an attacker access to the management network.
- An attacker breaks into the management network through a wireless access point.
- An attacker gains physical access to the management network and connects up a machine that acts as a backdoor into the network.
- A worm breaks into the network using a software vulnerability, replicates itself, overloads system resources and replaces files.
- A denial of service attack overloads system resources.
- A logic bomb is planted by a malicious programmer, damaging the system or giving access to it at a later date.
- Social engineering is used to gain passwords to the management network – perhaps a password controlling out of hours access for operators.
- Social engineering is used to manipulate the network by passing instructions to a gullible operator.
- The network is accessed through an inadequately protected maintenance port.
- The software on the network is attacked using a built-in open service.
- A malicious insider damages the network using the management software.
- The network equipment is attacked physically using bombs, fire, etc.

*Failures*

- Software crashes in the management software or operating system.
- Hardware fails in computers on the management network.
- Network cards fail, or become disconnected.
- Communication links fail.

*Accidents*

- Incompetent installation of the operating system or application software.
- Incompatible software upgrade.
- Accidental damage to hardware caused by maintenance.
- Accidental fire or flooding.
- Mistakes in data entry and the configuration and operation of the network.
- Accidental disconnection of cables by cleaning personnel.
- Aging and environmental processes damage communication lines.

These attacks, failures and accidents can have the following consequences:

- Damage to the hardware and software in the telecommunications network.
- Network breakdown leading to disruption of critical services (911 calls etc.).
- Loss of control over the telecommunications network
- Loss or corruption of data.
- Financial losses caused by damage to reputation.
- Gathering of information that could be used in a future attack.
- Theft of proprietary data such as strategic planning and product development specifications.
- Theft of personal information such as passwords, credit card numbers and bank account information.
- Use of system resources for game playing, password cracking, etc.

Although telecommunications networks continue to operate in the face of these daily threats, new vulnerabilities are emerging all the time, and the increased interdependency of large complex critical infrastructures puts them increasingly in danger of cascading failures that could knock out several services at once. A solution is needed which can detect problems in these networks and respond to them in real time. Safeguard is working to create this solution, and the next section will describe the architecture that has been developed so far.

## 3. The Agent Architecture

The Safeguard architecture is designed to protect the management network against attacks, failures and accidents. At present, the design has not yet been completed and this paper will only give a broad outline of the state of the work at present.

Safeguard is an agent-based system, which will build up a model of the normal operation of the network and respond to anomalies in real time. Problems in the network will be diagnosed by correlation agents, who will bring together information from anomaly detecting agents and agents wrapping existing security and diagnostic applications (such as intrusion detection systems, firewalls and virus checkers). This evaluation of the state of the network will be passed on to action agents for a response, which may include feedback to the anomaly detecting, and wrapper agents. Negotiation agents will communicate with agents in other large complex critical infrastructures to request services and pass on information about security alerts. Figure 4 shows the arrangement and interactions of these agents.

### 3.1 Anomaly detecting agents

The primary role of the anomaly detecting agents is to monitor the network for anomalies and pass this information on to correlation agents for analysis and action. Anomalies will be detected by instrumenting the software in the management network with sensors, which will extract information about the order of the functions being executed and the parameters passed within each function call. During the learning phase a model of the normal execution patterns of the system will be built up by recording which functions are called and their sequence. In the monitoring phase, the function execution patterns will be compared with the model of normal behaviour. The parameters passed in function calls - including numbers, letters and switch settings - will also be checked for known or hypothesised properties that are invariant across all runs of the software. Violations of these invariants could indicate a problem with the network. With both types of analysis, continual learning and adaptation will keep the definition of normality current.

The anomaly detecting agents will be organised into a hierarchy, with low level agents building up a model of the normal operation of the environment within each computer and communicating information about this to higher level agents with a wider view of the network. Each workstation server, firewall and database will have a low level anomaly detecting agent. Higher level anomaly detecting agents will be more widely dispersed and they will integrate and analyse the information from the low level agents and pass it on to the correlation agents. The arrangement of the anomaly detecting agents is illustrated in figure 2 below.[2]

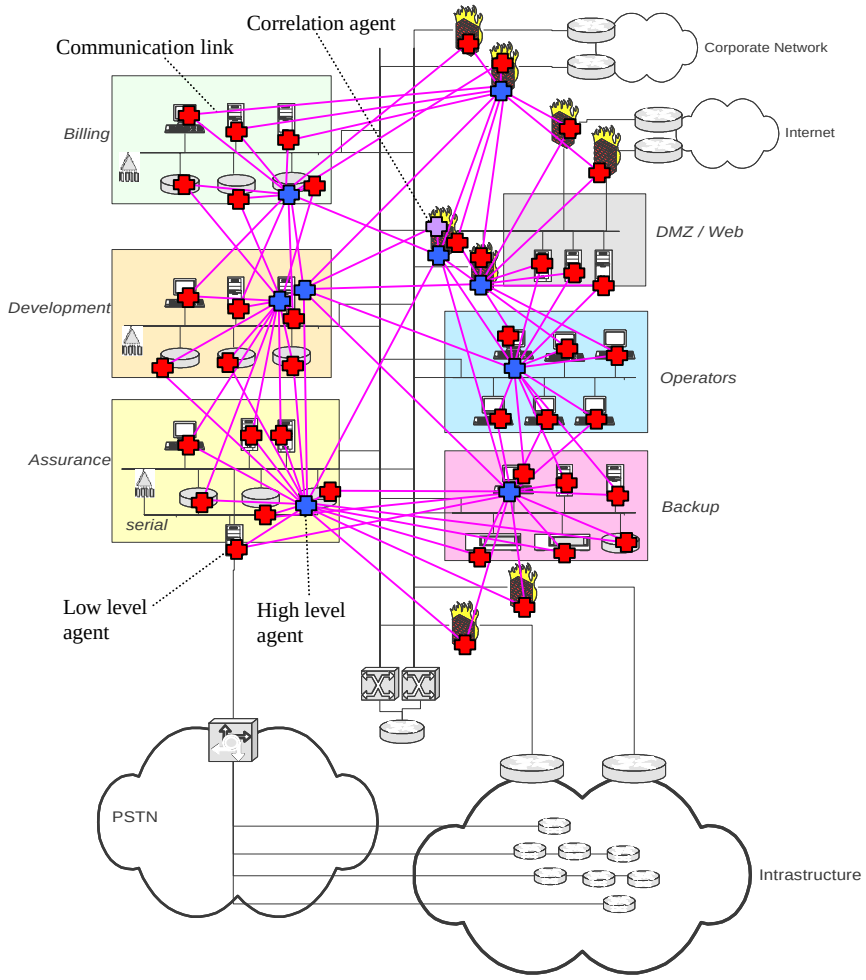


**Figure 2: Location of anomaly detecting agents within the management network**

[2] This arrangement of agents has a lot in common with the AAFID architecture [2]. However AAFID does not include feedback between the high and low level agents.

The hierarchical organisation of the anomaly detecting agents can be seen more clearly if they are laid out according to their logical relationships (figure 3).

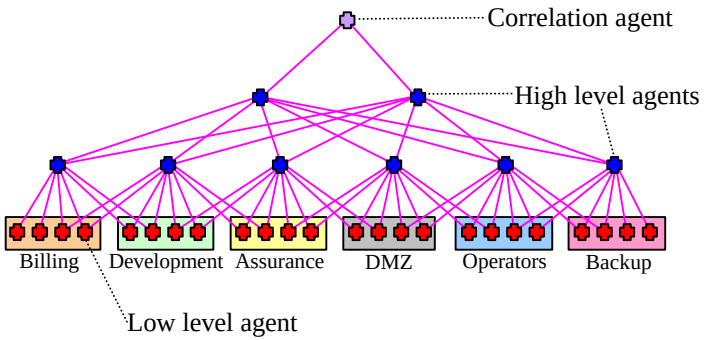


**Figure 3: The logical organisation of the anomaly detecting agents**

The main difference between the correlation agents and the higher level agents is that the latter will only integrate information from other anomaly detecting agents, whereas the former will evaluate the state of the network using information from a number of different sources and initiate actions in response. To reduce the number of false alarms, the high level anomaly detecting agents may also be capable of detecting anomalies in the level of anomaly that is sensed by the low level anomaly detecting agents. Section 4 will cover the technology that will be used inside the anomaly detecting agents in more detail.

### 3.2 Wrapper agents

The wrapper agents will interface with other applications running on the system. These include the intrusion detection system (IDS), firewall, virus checker and diagnostic software. Their task will be to pass information from these applications to the correlation agents and receive feedback from the action agents, in the form of data and policy updates.

### 3.3 Correlation agents

These agents will operate across wide areas of the system. Their main tasks will be to receive information from the anomaly detecting and wrapper agents, to request more information once a problem has been detected, to evaluate the state of the system and to pass this evaluation on to action agents who will carry out an appropriate response. This evaluation will be carried out using case based reasoning (or a similar technique) and the system will learn from past experience and the operators.

In addition to their coordination functions, the correlation agents will look for connections in the information that they receive and use this to update applications on the system. For example, by correlating information about an anomalous process with a positive result from the virus checker they can explicitly identify the anomalous process as a virus and then instruct the anomaly detecting agents to learn its anomaly pattern, so that they can recognise it (and its variants) more quickly in the future. This will be especially useful for polymorphic viruses. The correlation agents could also update the IDS when an anomaly detecting agent discovers the signature of an unknown buffer overflow attack.

### 3.4 Action agents

The action agents will receive a diagnosis from the correlation agents and decide upon an appropriate course of action. To respond effectively to anomalies these agents will have to have detailed knowledge about the most critical areas of the system and the best way to restore it to its normal state. Some of this could be learnt by example from the operators, but a lot may have to be hard coded. This knowledge could be stored and applied using case based reasoning (or a similar technique). The actions of these agents could include:

- Increasing the sensitivity of the anomaly detecting agents (see section 4.4).
- Updating the IDS, diagnosis software, firewall, virus checker and anomaly detector with information derived from other agents.
- Closing off a port or filtering out an IP address that is responsible for a large number of anomalies.
- Killing a process that starts up without instrumentation.
- Killing and restarting a process whose type changes.
- Killing and restarting a highly anomalous process.
- Dropping a user who is responsible for a large number of anomalies (or bringing them to the attention of an operator).
- Querying the execution of functions that have not previously been used in the normal operation of the system.
- Querying the execution of functions that have lead to system failures in the past operation of the system.

- Repairing functions that are consistently associated with system failure. This could be done through the intervention of human operators, or using genetic programming if a way could be found to automatically guide the evolution of a program.
- Reconfiguring the system to work around hardware and software failures. This could involve migrating processes to different machines and instructing negotiation agents to request services (such as electricity or telecommunications) from other large complex critical infrastructures
- Contacting an operator for help. This would be a last resort when a substantial anomaly could not be diagnosed and repaired by the Safeguard system.
- Informing correlation agents in different control centres about the state of the network and passing on solutions to problems that have been diagnosed and successfully repaired.

These self-healing mechanisms will have to be carefully controlled so that they cannot be abused to create a denial of service.[3] To help with this some form of voting could be used on important decisions. This would enable the action agents to combine information from agents with different information and expertise and it would reduce the chances of a compromised agent gaining complete control over the system.

### 3.5 Negotiation agents

These will come into play when the Safeguard system needs to interact with agents in other large complex critical infrastructures. Their tasks will include rerouting and requesting services and sharing information about failures and attacks. If the telecommunications network experiences a major failure, the negotiation agent could arrange for calls to be switched through to the most suitable network at an appropriate price. If the data communications in an electricity network fail, more services could be requested from the telecommunications network.

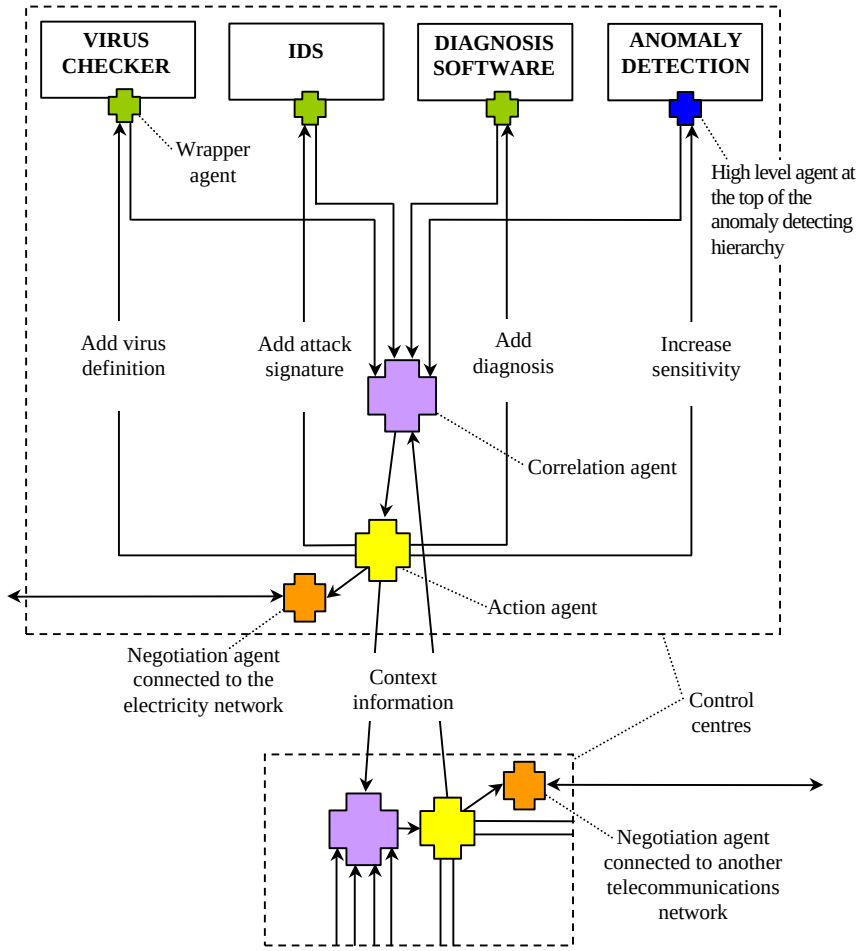


**Figure 4: The interactions between different agents**

[3] For example, by spoofing the source of anomalies an attacker could cause critical ports to be closed by the action agents.

## 4. Anomaly Detection

The anomaly detecting agents will detect anomalies in the following stages:

1. The software is instrumented to extract information about execution patterns and the parameters passed by function calls.
2. The system is monitored whilst it is executing normally.
3. A model is built up of the normal operation of the system. It is assumed that no attacks, failures or accidents take place during this stage.
4. The system is monitored for anomalous behaviour.
5. The normal model is updated to reflect gradual changes in the system over time.

The most critical stages in this process are the instrumentation of the system to extract data and the analysis of this data to determine whether the system is operating normally. These will be covered in more detail.

### 4.1 Software instrumentation

In previous work information about the operation of the system has been gathered using system calls [8], statistical analysis of users and program behaviour [1], monitoring of connections [4] and sensors inserted into the software [10]. In Safeguard it was decided to use the software sensor approach to extract information about the sequence of function calls and the parameters that are passed. This provides a much more comprehensive view of the operation of the system than the other techniques and this method can be used to gather as much information as all of them put together.[4]

Figure 5 illustrates the instrumentation of a system to extract data. Each function in the original application is labelled with a number, which is sent to the Safeguard monitoring system along with the parameters that are passed by the function. Different numbers are used to label different applications and different parts of the operating system, so the sequence of numbers recorded by the anomaly detecting agents gives information about the type of application that the function is from and the interaction between different applications on the system. The process ID of the application that calls the function is also recorded.

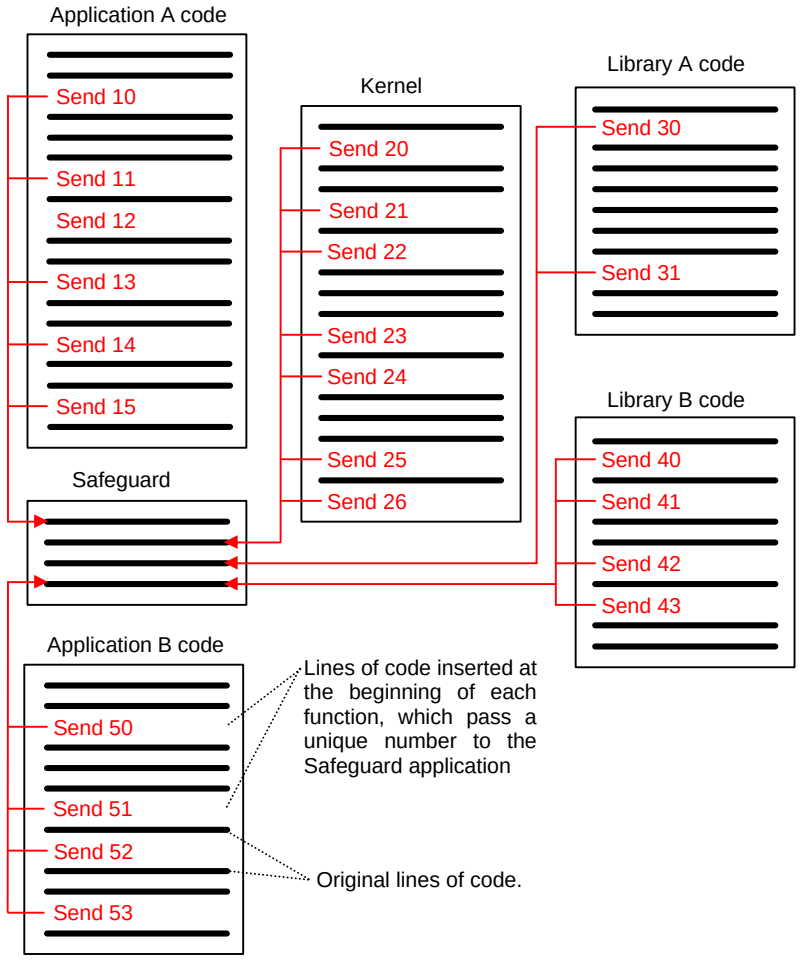


**Figure 5: Instrumentation of software to extract data**

[4] For example, system calls show up within the sequence of function calls as a function call within an application followed by a sequence of function calls in the kernel.

At present this technique is being implemented using source code to test the architecture and processing cost. At a later stage it will be extended by instrumenting binaries in the same way. This should increase the efficiency of the system and it will then be possible to apply this technique to applications where the source code is not available. The instrumentation of binaries has the further advantage that Safeguard can be launched on an existing system without reinstalling everything from scratch.

The instrumentation of software raises important questions about performance degradation. In the previous work that has been done on the instrumentation of operating systems [3, 12, 15], the computer has been slowed down by factors ranging from 4 to 35 at its best performance [3, p.7], and this could have serious effects on time critical applications. The key to managing this performance degradation lies in making the instrumentation as flexible as possible. If the instrumentation of every function in an operating system or application will slow it down excessively, then every other function could be instrumented instead. This trade off between the resources available and the performance of the anomaly detection could be managed dynamically by the agents installing the Safeguard system. This will be the subject of further research.

When the instrumentation is complete, the execution patterns and the function parameters of the applications and the operating system will be passed to the anomaly detecting agents for analysis. In this stream of information, data from different applications will be interleaved, but the association of each piece of data with a process ID will enable it to be traced back to the application that created it. The analysis of the sequence of function calls and function parameters will now be covered in more detail.

### 4.2 Analysis of the sequence of function calls

The stream of numbers that will be sent by the functions as they are executing provides information about:

1. The order of function calls within an application or the operating system.
2. Calls to the operating system from an application.
3. Calls within different components of the operating system.
4. Switches between applications caused by the kernel or user.
5. The type of process that is causing the function call.

Table 1 shows some simplified examples of the data sequences that would be obtained if a system were instrumented as shown in figure 5.

| System events | Example data |
|---|---|
| Sequence of function calls within application A. | 10 12 10 14 13 15 12 |
| Application A calls a function in the operating system kernel. | 10 12 22 23 13 15 10 |
| Switch between application A and application B caused by time slicing in the kernel (25 and 26 are the kernel functions responsible for time slicing). | 10 12 25 26 51 53 53 |
| Switch between application A and application B caused by a user action (21 and 20 are the kernel functions responsible for context switching). | 10 12 21 20 51 53 53 |

**Table 1: Data sequences from applications and the operating system**

These sequences will be analysed for the numbers they contain (alphabet analysis) and the sequence of numbers (time series analysis). The process ID associated with each function call will also be tracked.

Alphabet analysis will look at *which* functions are being called in the normal operation of the system. In the work that has been done on system calls so far, the monitoring of the alphabet has not played a significant role because an alphabet of a few hundred system calls is shared amongst all the applications on the system. Within Safeguard the alphabet is expected to play a much more important role since each application will be instrumented with a unique set of numbers and it is anticipated that only a limited proportion of this alphabet will be used during normal operation. Attackers are likely to activate functions that are not part of normal use and this will easily be detected. This alphabet analysis fits in with the work of Neumann [11], who advocates the use of thin operating systems to avoid the risks associated with the unnecessary functionality that is usually included within most modern operating systems. This surplus functionality introduces vulnerabilities, increases the probability of obscure faults and often provides tools (telnet or a compiler, for example) that make it easier for an attacker to abuse the system. By monitoring the functionality in use Safeguard can harden an operating system by turning off functions that are never called. A complex operating system becomes a simple one with functionality tailored to its own particular context.

The time series analysis will look for relationships *between* the numbers that are passed to the anomaly detecting agent. This will enable the agent to recognise when one program is masquerading as another and the takeover of a program (in a buffer overflow attack for example). One method, which could be applied to the time series analysis, is that developed by Forrest et al. in their work on system calls [8]. This uses a sliding window, usually of length six, which moves along the sequence of system calls and records each pattern within the window. During the learning phase, the sequences of system calls are stored in a database to build up a model of the normal operation of the system. When the system is being monitored for anomalies, the sequences within the sliding window are compared with the database and non-matching sequences are flagged as anomalous. To apply this approach within Safeguard it will have to be adapted to match the data from function calls. One of the most important factors that will need to be adjusted is the length of the sliding window. If this is too short it will be unable to detect some attacks. If it is too long, the system will slow down and a larger database will be needed to store the normal model. In their experiments Forrest et al. found that the optimum window length was six. However, Maxion and Tan [17] have shown that this window length is an artefact of Forrest's data, which contains an attack requiring a sequence of six system calls to identify it correctly. More research is needed to fine tune this parameter for the Safeguard data. It may be possible to use a window length of 2 (along with Stillerman et al. [16]) or, if the minimum attack length is longer than six, the finite state machine model suggested by Marceau [9] could be employed.

A second issue connected with time series analysis is the amount of pre-processing that needs to be applied to the data. Some of the data will consist of thousands of calls to a single function and it makes sense to filter at least some of this out before passing it on to the anomaly detecting agent. When one function call is always followed by another, the first function call could also be eliminated from the analysed data.[5] There are also questions about the balance between local and centralised processing of information. The instrumentation will be adding code to applications and the operating system to manage the transfer of data to the anomaly detecting agent. The amount of data that is passed could be reduced if this code carries out some local processing on the data before passing it on to the Safeguard agent.

The alphabet analysis and the time series analysis will be brought together to build up a model of the normal operation of the system. However, to achieve this a number of problems need to be overcome. To begin with, given the finite processing power of the machine, there is likely to be a trade-off between a large alphabet, i.e. a lot of instrumentation points, and the complexity of the time series analysis. Many instrumentation points gives you a better view of which functions are operating, but this leaves less system resources available to analyse the relationships between functions. This balance will have to be worked out in detail. A second problem is the difficulty of building up a normal model for large complex applications using the sliding window approach. Forrest's system successfully built up a stable profile for login, sendmail and cron, but it had difficulty profiling more complex applications, such as Netscape and Emacs. Although the increased alphabet in Safeguard should make profiling easier,[6] the complexity of data may have to be managed through a layered approach, which would:

1. Build up a simple profile of the system by monitoring the interleaved sequence of function calls from applications and the operating system. This could apply a lot of pre-processing to the data to eliminate repetitions and functions that always call the same function.
2. Build up a more sophisticated profile of each application. This processing could be done within each application before its data is sent to the Safeguard agent.
3. Build up a profile of the subcomponents within each type of application. Large applications could be broken down into smaller parts, perhaps using high level information about the application's behaviour to build up a stable profile. This processing could also be done within each application.

This layered approach is based upon the architecture of the components being analysed, moving from the interoperation of the system as a whole down to the operation of subcomponents. It would also be possible to develop a layered approach based upon the significance of the anomalies that are likely to be detected and the cost of looking for these anomalies. This would look for anomalies in the following order, dropping the analysis of less important layers if the processing power is not available and expanding the analysis if the sensitivity increases (see section 4.4):

[5] This is the technique used by Munson [10] to reduce 3000 instrumentation points in the Linux kernel to 100 virtual functions.

[6] Forrest et al. tried to build up a profile of Netscape using a few hundred system calls and a sliding window length of six. More data will be available in Safeguard, which will instrument all 60,000 of Mozilla's functions. Furthermore, in the Safeguard data numbers close together are likely to be functionally related, which is not the case in sequences of system calls.

1. Alphabet anomalies - a function is called which is not normally used in the operation of the system.
2. Anomalous transitions between applications and the operating system.
3. Anomalous transitions within applications or the operating system.
4. Anomalous transitions between applications.[7]

This subsection has covered the analysis of which functions are being called and the relationships between these calls. The next section will cover the analysis of the parameters that are passed in the function calls. Section 5 will give more information about how attacks, failures and accidents can be detected by these methods.

### 4.3 Analysis of function parameters

Execution patterns give a good picture of the functionality that is in use while the network is operating. However they do not give information about whether normal functions are being used for abnormal ends. The control software in the management network can be used normally to configure a router, but if this configuration is done incorrectly, there can be substantial disruption of the network. During normal operation, the operating system utilises functions to delete files. However if the same functions are used to delete critical files, the entire system can collapse.

To protect the telecommunications management network against this kind of misuse, a second kind of anomaly detection is needed which will monitor the values of data and control signals passing within the system. This type of monitoring has been missing from the anomaly detecting systems that have been developed so far. The best way to track this kind of information is to extract the parameters that are passed within function calls and analyse them to detect their invariant properties. This approach is similar to that used in Michael Ernst's Daikon application [7], which inserts code into a program's functions to extract the values of the global and local variables that are in scope. The program is then run over a test suite and Daikon looks for invariant properties of the extracted variables, such as 'x > 10', 'x = y + 4', 'array A contains no duplicates', and so on.

[7] Since a program can be interrupted at any point, a large amount of data needs to be stored to record this type of information. These transitions might be relevant since a large number of user-induced application switches could be highly abnormal. However, given the innocuousness of this type of activity and its potential for creating false anomalies, it may not be worth recording it at all.

Ernst's approach is promising, but a number of modifications need to be made to integrate it better with Safeguard. To begin with, Ernst aims to find a complete set of invariants for the program by running it over as comprehensive a test suite as possible. Safeguard would not use a test suite; instead, programs would be run in their usual environment and invariants would be found relative to this situation. This would give information both about the operation of the program and the context it is running in. This creates an important difference between the invariants discovered by Daikon, which are program invariants, and those detected by Safeguard, which will be context bound invariants. Daikon only has to test a program once over a comprehensive test suite to discover the program invariants, whereas Safeguard will have to monitor the invariants continually to detect changes that are caused by abnormal behaviour of the system – unusual switch settings, anomalously long data entries, etc. A second modification would be to use more sophisticated techniques to identify invariants. Daikon looks for fairly simple properties of the variables, but neural networks, genetic programming or similar techniques could also be used to approximate the invariant behaviour of variables and the relationships between them. A potentially interesting approach would be to search for statistical relationships, rather than predicates requiring absolute compliance, and use the degree of deviation as an indicator of potential problems.

To manage the cost of processing this type of information, the invariant monitoring could be divided into two levels:

1. Monitoring of function parameters that are known to be important or have proved themselves to be important in the past. This could include parameters in critical parts of each computer and the parameters passed between applications and across the network. This would protect the switch readings and settings in a telecommunications network, which are often subject to 5% error.
2. Monitoring of more speculative parameters. Only a proportion of the possible parameters would be monitored at this level. This proportion would be selected randomly and periodically a different random selection would be made. The size of the proportion could be adjusted dynamically to match the processing load on the system. Parameters that are found to be good indicators of problems could be upgraded to level 1 monitoring.

These parameter monitoring policies will be set by the action agents under the guidance of the correlation agents. They will also be affected by the sensitivity level of the system (see next section).

**4.4 Sensitivity**

To reduce the number of false alarms and to manage the processing load on the system, each anomaly detecting agent will contain a variable that will control its degree of sensitivity. If everything is normal, the sensitivity will be low and many anomaly checks will not be performed on the system. If the level of anomaly increases in one part of the network, a message will be sent to some or all of the agents to adjust the amount of checking that they do and the types of anomaly that are examined. If nothing happens over a period of time, this sensitisation will gradually die away.

The level of sensitivity could affect:

- The level of detail of the analysis of the sequences of function calls.
- The proportion of level 2 parameters that are analysed for anomalies.
- The type of level 2 parameters that are analysed for anomalies. If there are a lot of anomalies in strings then the proportion of level 2 strings that are analysed should be increased.
- The locations that are analysed for anomalies. Anomalies in applications that are physically or logically connected with a misbehaving application should be watched more closely and responded to more vigorously than anomalies in other parts of the network.

False alarms could also be controlled by learning gradual shifts in the behaviour of the system over time and switching into learning mode when legitimate changes are made to the system. Care will be needed to prevent these techniques from introducing security weaknesses.

## 5. The Operation of the Safeguard System

Once the Safeguard system has been installed, the anomaly detecting agents will monitor their local environment and pass information about its anomaly level to the correlation agents. To minimise network traffic, these messages will not include much detail about the anomalies that are detected. The wrapper agents will also send information to the correlation agents about the status of the IDS, virus checker, firewall and diagnosis applications. Using the information from these sources, the correlation agents will generate a map of the network showing the activity in each of the nodes they are connected to. When an anomaly arises in the area they are monitoring, they will ask the local agents for more information – for example the source of the anomaly (the process ID), the type of anomaly (whether the anomaly was in the function call parameters or the execution patterns) and information about viruses or intrusions that have been detected on the system. Once the problem has been identified, an action agent will be instructed to repair the anomaly and make system adjustments, using the mechanisms described in section 3.4.

The response of Safeguard to some common attacks, failures and accidents will now be covered in detail.

*Unknown buffer overflow attack*

Buffer overflow exploits are one of the most common ways in which worms spread and attackers gain root access to a system. These will be identified by the anomaly detecting agents in a number of different ways:

1. The parameter monitoring will detect that an anomalous parameter has been passed and reject the function call. The parameters passed in a buffer overflow attack are always anomalously long.
2. If an attacker uses a buffer overflow exploit to launch a different application with the process ID and privileges of the compromised process, there will be anomalous calls to the kernel from the new application and the application type will change. For example, if the server is instrumented with numbers ranging from 400 to 800 and the shell is instrumented with numbers ranging from 1000 to 1200 and the server is compromised to launch a shell with root privileges, then the functions called by the process will switch from the range 400-800 to the range 1000-1200. This can easily be detected.
3. If an attack is carried out purely by using the attacker's own code, then any system calls that are generated by this code will cause anomalous function sequences within the operating system.
4. If the attacker's code avoids system calls altogether, or restricts itself to system calls that are normal for the system, then anything abnormal, which the attacker does, will be detected by Safeguard. Safeguard is designed to protect against malicious insiders as well as outsiders, and it should detect all abnormal manipulations of the system, whatever their source.

Processes that are compromised by a buffer overflow exploit will be killed and restarted by an action agent. The correlation agent will know that this was an unknown buffer overflow exploit since it will not have received an alert from the IDS about it. Once the attack has been dealt with the correlation agent will send the signature of the exploit to the IDS to enable a faster and more targeted response to

the attack in the future. Known attacks could be dealt with directly by the IDS under the control of policies set by the correlation agent.

*Unknown virus*

If the virus's code is not instrumented, it will be detected and killed immediately as a process without instrumentation starting on the system. To fake instrumentation, the virus will have to have access to the communication channels within Safeguard, it will need to know the numbers used by an application and it will need to know the system calls made by the application. Randomisation of the numbers that are used to instrument each system and encryption of the communication channels will make it harder to guess an instrumentation pattern. Furthermore, even if the virus fakes instrumentation, it will not be able to do anything anomalous to the system without being detected and destroyed.

The correlation agent will know that the virus was unknown since it will not have received an alert from the virus checker about the problem. The signature of the virus could be used to update the virus checker to ensure a more rapid and targeted response in the future. Known viruses could be dealt with directly by the virus checker under the control of policies set by the correlation agent.

*Unknown rootkit*

The installation of an unknown rootkit will be detected since it will use functions within the kernel and its libraries. This will generate anomalous function calls within the kernel, which should be detected by the Safeguard system. Removal of a rootkit might require reinstallation of the operating system and an operator should be informed about this type of anomaly.

*An attacker or insider without technical knowledge gains access to the system*

Someone with little technical knowledge would set switches and routers to unusual values, activate functionalities that are not used in the normal operation of the system and use the normal functions in abnormal ways. All of these changes will be detected by Safeguard. Attempts to disable Safeguard will also be detected (see section 6 on the protection of the Safeguard system). More than one operator needs to be informed about this type of anomaly to make sure that the operator responsible for the anomalies is not the only one informed about them.

*An attacker or insider with technical knowledge gains access to the system*

Any damage caused by an attacker will probably involve changing settings in abnormal ways that would be detected by Safeguard. However a skilled operator could probably damage the system incrementally over a long period of time and escape detection. The key to securing the management network against this kind of attack is to progressively extend the physical time that is needed to change the network without detection so that it is not worth attacking the system in this way. If incremental changes do lead up to a catastrophic result, then the anomaly detection and self-healing will play a crucial role in buying time and limiting the spread of damage as problems develop. More than one operator should be informed about this type of anomaly.

*Software or hardware failures*

Software errors and hardware failures will affect the sequence of functions within the system and their parameters. This will be noticed by the anomaly detecting agents. The correlation agent could then request more information about the anomaly and then run diagnosis software to identify the problem. Once the problem had been fully diagnosed a limited amount of software repair could be carried out and both hardware and software failures could be worked around, perhaps by migrating processes to different parts of the system. Operators could also be informed about the problem.

The most important feature of these responses to attacks, failures and accidents is the multi-level survivability that is provided by Safeguard. Safeguard agents will secure the perimeters of the telecommunications management system – just as fortress-model intrusion detection systems do today – but their distributed software sensors will continue to protect the system once an intruder is inside. Safeguard is a fortress that can lose its walls without significantly compromising its security. This kind of total defence protects against both attackers and insiders and it will become the next step forward in survivable architecture.

## 6. Protection of the Safeguard System

The Safeguard protection has to be maintained even if an attacker gains full access to the system. A security system that can be switched off easily is of little use in this context and so the self-monitoring and protection of the Safeguard agents is extremely important. This will be achieved in a number of different ways:

- *Monitoring of critical files and processes*. Any attempt to modify Safeguard – alter its log files, kill its processes, etc. - will be detected as abnormal and prevented. Agents could also explicitly watch, in the manner of *Tripwire* [18], the critical parts of the Safeguard system and intercept unauthorised attempts to modify them before they take effect.
- *Instrumentation of Safeguard agents*. Safeguard agents could be instrumented to detect abnormal patterns in their own behaviour. The

problem with this approach is that an agent can only detect its own failure while its monitoring is intact. If the monitoring is compromised or fails then Safeguard will be unable to detect its own failure. This can be partially circumvented by distributing some of the Safeguard processing amongst the applications that are being monitored and establishing redundant communication channels between applications and the outside world. If the Safeguard server goes down in one machine, then the applications and the operating system could inform agents in other machines about the problem and increase their sensitivity to abnormal behaviour until the problem is resolved.

- *Communication between agents in different parts of the system*. Safeguard agents in different parts of the system will monitor each other. If agents in one part of the system fall silent, the other agents will realise that there is a problem and take action.
- *Concealment of Safeguard agents*. Research is being done into the use of invisibility as a protection mechanism for Safeguard. It will be much harder to modify or compromise Safeguard if it is made completely invisible to the local user (by hiding processes, files, open sockets, etc.).

## 7. Preliminary Results

The development of the Safeguard system is still at an early stage and the main focus has been on the threats that need to be addressed and the design of the architecture. The anomaly detecting part of this architecture has been tested on a small scale by instrumenting the source code of simple applications so that they monitor and process their execution patterns and send them to a central server for analysis. This server will become one of the components of the anomaly detecting agents when the Safeguard system is constructed. Work is currently under way on the instrumentation of Mozilla[8] to test the performance of the data extraction on a more demanding application and improve the data analysis and filtering. In the experiments so far, the data analysis has been limited to the alphabet that is used by each process. This has proved to be an effective method of detecting the execution of functions that are not used in the normal operation of the system.

[8] An open source browser similar to Netscape.

## 8. Related Work

The design of the Safeguard architecture has been influenced by the work that has been done on anomaly detection, survivability and the dynamic discovery of invariants.

### 8.1 Anomaly Detection

In recent years a great deal of research has been done on anomaly detection. The two systems that are most closely related to the Safeguard project are *pH* and *Watcher*:

*pH*

pH is a system developed by Forrest et al. [8] that resides in the Linux kernel and detects buffer overflow attacks by monitoring the system call patterns of each application. During the learning phase a sliding window is moved along the sequences of system calls and the patterns are recorded into a database. The system is monitored by comparing the learnt sequences representing normal behaviour with the sequences emitted by the application. To protect against attacks, anomalous system calls are delayed by an amount proportional to the observed degree of anomaly. An application with a large number of anomalies slows down to the point at which it is unusable by an attacker. Whilst pH is a promising system, this approach has a number of limitations:

1. The source code of the kernel is required to install pH and monitor system calls. For Linux this is not a problem, but there is no easy way of porting pH to Windows. Many of the management systems of large complex critical infrastructures are based around Windows.
2. Whilst pH can develop a stable normal profile for simple applications, such as login, sendmail and cron, it has difficulty profiling more complex applications, such as Netscape and Emacs. Safeguard will have to monitor programs that are potentially as complex as Netscape and Emacs and a way needs to be found around this problem.
3. pH uses delay to respond to anomalous system calls. This will not work with Safeguard since an attacker could deliberately cause delays and create a denial of service attack against time-critical systems.

*Watcher*

This is a system developed by Munson [10] and now marketed by Cylant [5]. It works by instrumenting the source code of an application or operating system with sensors that extract information about the execution patterns of the

program. To reduce the amount of data that is needed to record normal behaviour, the instrumentation points are compressed into a smaller number of virtual functions (in the analysis of data from the Linux kernel, 3000 instrumentation points were reduced to 100 virtual functions). During the learning phase, the number of times that each of these virtual functions is called in a given time period is recorded to build up a profile of the kernel's normal behaviour. Abnormalities can then be detected and responded to by dropping users or restarting processes. Munson's system was tested by protecting a vulnerable Linux kernel and inviting people to attack it over the Internet. Although the machine was eventually compromised it survived 13 000 attacks within the first two months of its operation.

Safeguard plans to extend Munson's approach by taking the relationship between processes more into account, freeing it from its dependency on source code and extending its analysis of the time relationships between functions.

### 8.2 Dynamic discovery of invariants

The discovery of invariants in the parameters passed in function calls is influenced by Michael Ernst's work on the dynamic detection of program invariants [7]. Ernst's system, Daikon, identifies invariant properties in the variables within a program by instrumenting its source code and running it over a test bed to extract data. This data is then analysed for invariant properties, such as $x > 10$, $x + y = 35$, etc. Ernst's approach is orientated towards debugging applications, but it can be adapted to build up a normal profile of the parameters passed in each function call by instrumenting binaries and extracting invariants for the context in which the program is running. Violations of these invariants indicate a deviation from the program's normal behaviour.

### 8.3 Survivability

Peter Neumann's work on survivable systems [11] has been extremely useful for the design of the Safeguard architecture. One of the problems that Neumann has identified in modern operating systems is the large amount of code they contain and the potential for errors and exploits within their complex structure. This contains a large number of functions that are never used in the ordinary running of the system, which increases the potential for security vulnerabilities and other failures:

> Windows 2000 (N 5.0) reportedly will have something in excess of 50 million lines of source code (most of which appears to be kernel code), with another 7.5 million lines of associated test code. … Unfortunately, the totality of code on which survivability and security depend is essentially the kernel and operating system plus potentially all the application software that can be loaded at any time. That represents an enormous amount of code that must be trusted (because it is not trustworthy) in any critical application. (Recall the divide-by-zero in an NT application that brought the Yorktown Aegis missile cruiser to a halt, in Section 1.6.) [11, p.82]

Neumann's solution to this problem is to develop 'thin' operating systems that are precisely tuned to the user's needs:

> One approach to establishing highly survivable systems and networks involves architectures in which subsystems (e.g., servers and end-user platforms) are stripped of not only their unneeded functionality but also the corresponding undesirable vulnerabilities that the superfluous functionality entails. [11, pp.108-109]

Although thin client operating systems are available, and new ones can be developed, it is a potentially expensive and complex matter to adapt industrial software to run on them and so this technique is unlikely to be applied to real world systems. Safeguard's solution is to dynamically adapt 'bloated' operating systems, such as Windows and Unix, by switching off the functions that are not used in the ordinary running of the system.

## 9. Conclusion

This paper has outlined the design of an agent-based architecture that will increase the survivability of the management networks of large complex critical infrastructures such as the telecommunications or electricity networks. This architecture will defend against unknown attacks, failures and accidents by identifying anomalies in the system, requesting information from diagnosis and security applications and then reasoning about an appropriate self-healing response. This architecture could be installed on any platform - it could even run on the routers and switches in a telecommunications network. This protection has a processing cost, but this can be managed by matching the level of anomaly detection to the system that it is being protected. This paper has only outlined the initial design of the Safeguard architecture, and it is hoped that tests will soon demonstrate the substantial increase in survivability that will be offered by this type of agent-based anomaly detecting system.

## 10. Acknowledgements

We would like to acknowledge the other members of the Safeguard project team for their help with the preparation of this paper. These include Wes Carter (Queen Mary, University of London), Stefan Burschka (Swisscom), Simin Nadjm-Tehrani (Linköping University), Sandro Bologna (ENEA), Claudio Balducelli (ENEA) and Carlos López Ullod (AIA). We would also like to thank the European Information Society Technologies Programme for their support of this project.

## 11. References

[1] Anderson, D., Teresa F. Lunt, Harold Javitz, Ann Tamaru, Alfonso Valdes, 'Detecting Unusual Program Behaviour Using the Statistical Component of the Next-generation Intrusion Detection Expert System (NIDES)' SRI International technical report May 1995 SRI-CSL-95-06.

[2] Balasubramaniyan, J. S., J. O. Garcia-Fernandez, D. Isacoff, E. Spafford and D. Zamboni, 'An Architecture for Intrusion Detection using Autonomous Agents', COAST Technical Report 98/ 05, June 11, 1998.

[3] Casmira, Jason P., David P. Hunter and David R.Kaeli, "Tracing and Characterization of Windows NT-based System Workloads", *Digital Technical Journal*, Vol. 10, No. 1, (1998).

[4] Cheung, Stephen, Rick Crawford, Mark Dilger, Jeremy Frank, Jim Hoagland, Karl Levitt, Jeff Rowe, Stuart Staniford-Chen, Raymond Yip, Dan Zerkle, 'The Design of GrIDS: A Graph-Based Intrusion Detection System', UCD Technical Report CSE-99-2, January, 1999.

[5] Cylant web site: www.cylant.com.

[6] dti (Department of Trade and Industry, UK), 'Information Security Breaches Survey 2002', available at: https://www.security-survey.gov.uk/ isbs2002_detailedreport.pdf.

[7] Ernst, Michael, *Dynamically Discovering Likely Program Invariants*, PhD Thesis, University of Washington 2000.

[8] Forrest, Stephanie and Anil Somayaji, 'Automated Response Using System-Call Delays', Proceedings of the 9th USENIX Security Symposium, Denver, Colorado, 14-17th August 2000.

[9] Marceau, Carla, 'Characterizing the Behavior of a Program Using Multiple-Length N-grams', Proceedings of the New Security Paradigms Workshop 2000, Cork, Ireland, Sept. 19-21, 2000.

[10] Munson, John C., 'Watcher: The Missing Piece of the Security Puzzle', Proceedings of the 17th Annual Computer Security Applications Conference, New Orleans, Louisiana, December 14, 2001.

[11] Neumann, Peter G., 'Practical Architectures for Survivable Systems and Networks', SRI Report, 2000.

[12] Romer, T., Geoff Voelker, Dennis Lee, Alec Wolman, Wayne Wong, Hank Levy, Brian Bershad, 'Instrumentation and Optimization of Win32/Intel Executables Using Etch' USENIX Windows NT Workshop, 1997.

[13] Safeguard web site: www.ist-safeguard.org.

[14] Skoukis, E., *Counter Hack*, New Jersey: Prentice-Hall, 2001.

[15] Srivastava, Amitabh and Alan Eustace, 'ATOM: A System for Building Customised Program Analysis Tools', Western Research Laboratory (WRL) Research Report 94/2.

[16] Stillerman, Matthew, Carla Marceau and Maureen Stillman, 'Intrusion Detection Distributed', *Communications of the ACM*, Vol. 42, No. 7, July 1999.

[17] Tan, Kymie M. C. and Roy A. Maxion, '"Why 6?" Defining the Operational Limits of stide, an Anomaly-Based Intrusion Detector", IEEE Symposium on Security and Privacy: Berkeley, California, 12-15 May 2002.

[18] Tripwire website: www.tripwire.org.

**David Gamez** is a Research Assistant working on the Safeguard project. His research interests include artificial immune systems and biologically structured neural networks.

**John Bigham** is a Reader in the Department and heads the Intelligent Systems group, which carries out applied and basic research into a broad range of issues related to the deployment of agent technology in real-world applications. John has worked on a variety of EU-funded projects, including UNITE (Integration of Uncertain and Temporal reasoning) and DRUMS (Defeasible Reasoning and Uncertainty Management) within the ESPRIT Programme. Under ACTS he has participated in GEMA, AIM and IMPACT and in the 5th Framework is working on the IST projects SHUFFLE and SAFEGUARD.

**Laurie Cuthbert** is a Professor and Head of the Department. He has been active in European research, leading a task group in RACE R1022 and then work packages and work package groups in RACE EXPLOIT and BAF, and in the ACTS project EXPERT. He has provided the technical management for the ACTS project IMPACT as well as supervising the work within Queen Mary on agent management of channel allocation in AMPS networks. He is active in the IST SHUFFLE and SAFEGUARD projects.