Predictive Modelling for Meaningful Security SLAs

Mike Yearworth, Brian Monahan, David Pym
HP Laboratories
HPL-2006-50R1

Keyword(s):
service level agreements, demos2k, simulation, analtics, security, mathematical models

Abstract:
A *meaningful* Service Level Agreement (SLA) is defined as a contractual agreement between a service provider and customer that is valuable, measurable, predictable, understandable, and affordable. In a previous paper we discuss the development of the concept of a meaningful *security* SLA; that is an SLA focussed on the security properties of an information system arising from the interrelation of the infrastructure, processes and security operations staff. In this paper, we focus specifically on the development of a security operations model suitable for predicting performance against possible security SLAs. Although consequential financial losses can arise from lack of availability, confidentiality and integrity of an information system we specifically focus on a model that addresses lack of availability; that is, sources of downtime arising from security vulnerabilities and misalignment. We have introduced misalignment as a catch-all term to describe all change management tasks arising from staff not being able to complete tasks, leading to downtime and consequential financial losses, arising from access control problems; the information system infrastructure is available but is inaccessible due to mis-configuration. Whilst security vulnerabilities are the usual motivation driving security costs, the impact of misalignment is important in understanding the overall cost of security operations.

External Posting Date: October 10, 2008 [Fulltext] - Approved for External Publication

Internal Posting Date: October 10, 2008, 2008 [Fulltext]

Predictive Modelling for Meaningful Security SLAs

Mike Yearworth, Brian Monahan, David Pym
Hewlett-Packard Laboratories,
Filton Road, Stoke Gifford, BS34 8QZ, Bristol, UK.

{mike.yearworth, brian.monahan, david.pym}@hp.com

13th March 2006 (Revised: 8th October 2008)[1]

Abstract

A *meaningful* Service Level Agreement (SLA) is defined as a contractual agreement between a service provider and customer that is valuable, measurable, predictable, understandable, and affordable. In a previous paper we discuss the development of the concept of a meaningful *security* SLA; that is an SLA focussed on the security properties of an information system arising from the interrelation of the infrastructure, processes and security operations staff. In this paper, we focus specifically on the development of a security operations model suitable for predicting performance against possible security SLAs. Although consequential financial losses can arise from lack of availability, confidentiality and integrity of an information system we specifically focus on a model that addresses lack of availability; that is, sources of downtime arising from security vulnerabilities and misalignment. We have introduced misalignment as a catch-all term to describe all change management tasks arising from staff not being able to complete tasks, leading to downtime and consequential financial losses, arising from access control problems; the information system infrastructure is available but is inaccessible due to mis-configuration. Whilst security vulnerabilities are the usual motivation driving security costs, the impact of misalignment is important in understanding the overall cost of security operations.

1. Introduction

ICT security is no longer regarded as a pure technology issue, at least within the sphere of corporate business. It has become truism to say that ICT security is a process – something that happens to create smooth operating conditions for business. Modern corporate management makes essential use of metrics – numerical measures – to demonstrate business performance and hence show impact upon shareholder value. Questions then arise of how to estimate the level of *security performance* that these security-related processes achieve and, indeed, how to specify what their goals should be and how to predict outcomes and impacts.

Another standard corporate management technique is the use of explicit Service Level Agreements (SLAs) between the internal functional units of an organization and, more typically, with external services providers. These SLAs provide a contractual statement for what is to be delivered to the organization by the provider, internal or external. Contractual statements such as these thus formalize, to some extent, business process relationships and attendant expectations. With respect to security, these contractual statements have commonly taken the form of pure *policy compliance* statements e.g., conformance to ISO17799, and such like. These requirements are then audited on a regular basis, the results of which may contribute to a corporate annual report, and thus have impact upon business confidence. As we see it, there is an increasing trend for organisational security policy to extend beyond passive compliance against prevailing best-practice standards towards compliance with security policies that inherently involve SLA-style performance goals to be met.

The approach to understanding the relationship between systems properties and SLAs adopted in this paper, towards developing predictive models for meaningful security SLAs, is fundamentally that of HP Open Analytics, as described by Taylor, Tofts, and Yearworth (2004).

---

[1] Previous report was issued internally to HP as HPL-2006-50

Whilst we require any SLA to be meaningful we found it useful to invoke the control system view of Taylor and Tofts (2003) and to require two basic principles:

1. We must be able to measure the behaviour of the system such that we are able to control it i.e., modify behaviour such that the system does not deviate from desired operation; and
2. We must be able to predict behaviour based on the use of a model of the system.

In a previous paper, Monahan and Yearworth (2005) have provided a framework for discussing meaningful security SLAs but ignored prediction. This paper attempts to develop a preliminary model using Demos2000 (henceforth Demos2k) as developed by Christodolou, Taylor, and Tofts (2000) for (indicative) prediction, based upon phenomenological parameters.

In order to develop a predictive model, we have investigated the sources of security operations costs and financial tools for decision making through an engagement with HP IT[2] and the staff of the Chief Security Officer (CSO) and these are discussed in §1. Via the same engagement we have also investigated existing security metrics used within HP and are discussed in §2 and also putative SLAs developed as part of the HP account relationship between HP IT and HP MS as discussed in §4.

1.1. Sources of primary data

The HP CSO sponsors a number of projects which have contributed primary data to the modelling activity as follows. These sources are listed here together with the sort of data that contributed to the parameterisation of our model.

1. **Cost of security**
    a. **Function**: a wide-ranging measurement of all the costs associated with delivering security in HP conducted by the Business Information Security Managers (BISMs) and with a semi-annual reporting function.
    b. **Data**: number of operations staff including fractional FTE, list of operational activities relating to security and relative amount of time spent on each.
2. **Metrics**
    a. **Function:** reporting a set of measurements about the security state of HP presenting a dashboard to the CSO roughly equating to a threat position.
    b. **Data:** measurements that are not useful parameterisation of predictive models are irrelevant to this activity (but may have other uses). Conversely, rate modification parameters that seem useful in a predictive model may not be measurable. However, those that can be should be proposed as possible new metrics for the organisation.
3. **Cost-based analysis**
    a. **Function**: the financial tool for evaluating Return on Investment (RoI) on security projects based on using Net Present Value (NPV) and roughly in line with the approach of Gordon and Loeb (2005).
    b. **Data**: confirmation of the appropriate financial tools for decision making.

[2] Throughout this report we make reference to HP as the target organisation/system for predictive modelling; however, the approach is intended to be generic and broadly applicable with other large organisations.

4. **SLAs**

    a. **Function**: legacy from the HP MS/HP IT relationship where a broad set of SLAs were in the process of definition. This project was terminated as a consequence of the HP MS/HP IT divorce; however, some of the work is potentially reusable.

    b. **Data**: proposals for meaningful SLAs arising from predictive modelling.

1.2. Guiding principles

In developing a predictive model, we have adopted a number of guiding principles derived from HP Open Analytics practice:

- The model should be not *too* detailed as to be overly complex and slow to develop, execute and maintain; and not *too* simple as to miss important dynamics. The time value of modelling is also important. If this approach is to be replicated in customer engagements then for any model to useful it must be cost effective i.e., deliver timely results useful for making decisions with financial consequences;
- Distinguishing between external elements, such as

    threat environment, rate of discovery of vulnerabilities, speed to exploit, speed to develop patches, signatures,

    and internal elements, such as

    specific tasks undertaken in security operations and the speed with which these tasks are undertaken, propagation models of attacks and attack effectiveness;
- That *entities(classes)* in the Demos2k model should correspond to internal security operations tasks (test patches, clean machines, routine patching) and to external activities (exploit development, patch development);
- That *resources* in the Demos2k model should equate directly to FTE security operations staff, assuming that these dominate over time taken by non security operations staff to perform security activities and that there are no other resources available;
- That *constants* in the Demos2k model should be drawn from distributions should equate to metrics i.e. meaningful measurements of the HP system; either existing metrics, which would be useful, or ones that could be suggested. Rates associated with external elements will have to be best guesses based on intelligence gathering; and
- That *variables* in the Demos2k model should correspond to SLAs or components thereof.

The roles of Demos2k entities, resources, constants, and variables that is described here derives from an understanding of how to model the static and dynamic components of a (security) system described in Monahan and Pym (2006). There, a theoretical framework is proposed in which systems that deliver services are described by structured collections of locations, at which reside structured collections of resources, which together support the execution of processes. The properties of such a system are then described by a system of logic that exploits the structure of the locations and resources in order to account for concepts such as sharing and privacy. We return to these issues briefly in §4.1.

1.3. Prediction of performance against SLAs

The security SLAs developed through this work should provide a focus for the cost–benefit analysis of security operations. As a consequence we should be in a position to answer questions such as "what would happen to performance against the security SLAs if the number of security operations FTE

were reduced 10%?"; or "what would happen to the performance against security SLAs if the rate of discovery of security vulnerabilities increased 20%?".

1.4. Objectives of security

The objectives of security are classically stated as protecting the *Confidentiality, Integrity and Availability* of information assets and systems within an organisation. Of these, our focus in this paper is on availability modelling and subsequent cost to the organisation due to lack of availability of the information system leading to negative impact on the performance of business processes, and ultimately on revenue. However, it must be stressed that loss of integrity and breaches of confidentiality also lead to financial losses but these are not discussed in detail in this paper and is the subject of further research.

2. Sources of downtime

Sources of downtime can be aggregated and characterised as follows:

1. **Fix time:** time taken to repair specific components of the system that have been damaged by attack; typically this would be cleaning or rebuilding a computer that has been subject to attack including time taken to recover data. This source of downtime arises from *vulnerabilities*;
2. **Misconfiguration:** lack of access to a significant business component of the system (e.g. a business application like SAP) due to mis-configuration. By misconfiguration we mean any configuration of the system incompatible with operation of business processes (e.g. authentication failure, incorrect assignment of role) which would require operations resources to resolve. Also note the principle of constant change, there is never a correct system configuration. This source of downtime arises from lack of *alignment*;
3. **Intrinsic reliability:** the intrinsic reliability of the components that make up the system, the way in which they are connected and dependent and the operational level agreements in place to break/fix constitute the usual domain of availability analysis and typically the focus of SLAs in current IT Outsourcing (ITO) deals. Of course, downtime from sources 1, 2 and 4 will also contribute to this measure unless explicitly excluded. The means of predicting availability arising from intrinsic reliability is covered by Tofts and Taylor (2003). An area of potential future work would be security of critical infrastructure (e.g., integrity of power supplies to data centres);
4. **Network:** a catch all for lack of availability caused by the network itself being swamped by non authorised, non business related, traffic perhaps due to a Denial of Service (DoS) or worm attack. This source of downtime arises from *vulnerabilities*.

We make an essential distinction between downtime arising from vulnerabilities (1, 4) and alignment (2) ; typical analyses of threat environments and attack profiles concentrate on the former. However, we argue that in terms of a cost of security analysis that *mis*-alignment is actually a greater source of down time and also a greater contributor to operational costs.

2.1. Financial impact on the organisation arising from downtime

With a focus on sources of downtime arising from vulnerabilities and misalignment there is a requirement to financially calibrate downtime. Two measures that could be used are

1. wasted labour costs, and
2. impact on revenue.

We argue that some effort should be made to assess revenue impact arising from downtime. However, an initial first cut measure would be to pro rata company revenue per hour per employee, which for HP would be ~$87Bn/24*365 hours/150,000 employees which is ~$66 per hour per employee. Where the downtime occurs at the point of access to the information system – the desktop – then this

multiplier is fairly easy to use; when the downtime occurs in a server or application then the number of employees impacted is much harder to predict and the subject of further research.

3. Use of existing metrics

The following metrics are currently measured and reported to the CSO as a monthly dashboard:

1. IDS detects;
2. Publicly announced vulnerabilities ("IT-threat");
3. Desktop virus alerts;
4. Exchange virus alerts;
5. HP brand.com compromises;
6. Internal host compromises;
7. New CITSIRT cases;
8. Externally-facing hosts with non-patchable vulnerabilities;
9. Active countermeasures vulnerabilities found and corrected;
10. Awareness SOE training.

Some of these may be directly usable as parameters in a predictive model. Conversely, some parameters which appear to be useful in the model may be candidate metrics that should be measured and reported. This is discussed further in §4.5

4. Developing a predictive model (and modelling framework)

4.1. The conceptual and mathematical framework

The framework in which the availability model has been developed is shown in the diagram below. We refer to the mathematical basis as 'SCRP/MBI + location' for brevity. In fact, this account must be embedded in a stochastic treatment of environmental and operational events. Such a set up provides an account of the semantic basis for Demos2k (see Birtwistle and Tofts (1993, 1995)) using Milner's (S)CCS (1989) as the process-theoretic foundation. Here we propose a framework that incorporates our theory of resource semantics and extends it with purely structural components and access control primitives.

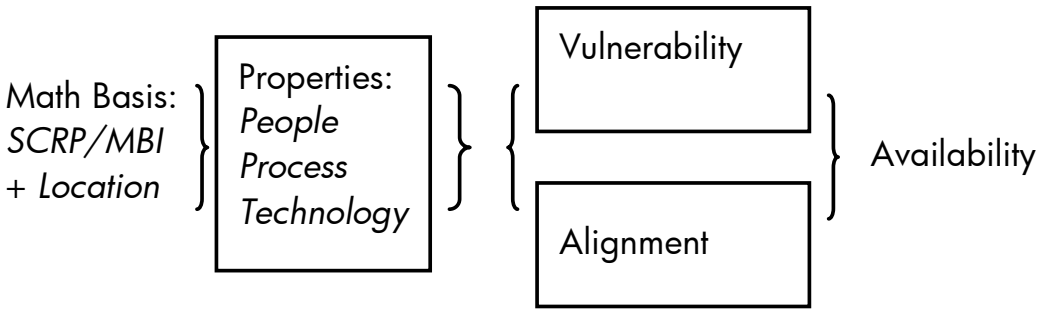

Figure 1. Framework used to derive availability model.

Demos2k has been used to develop the model described in this paper. Demos2k embodies the stochastic treatment of environmental and operational events but has just an 'atomic' notion of resource lacking the structural content of resources that allows resource composition and comparison. The integration of these concepts into a basis for Demos2k-like tool is the purpose of further

theoretical work underway in the context of the calculus and logic of resources and processes, SCRP/MBI, introduced by Pym and Tofts (2004), and begin developed by Monahan and Pym (2006) to include the concept of location and, potentially, a located analysis of concepts in access control such as roles and impersonation.

We initially focussed on threat modelling and internal mitigation processes associated with vulnerabilities such as patching and fixing compromised machines. Misalignment modelling is relatively less well understood and ripe for further research effort. Internal alignment processes include the following configuring new users, new devices, new customers, and new partners; updating configurations based on change requests, and non standard business critical fixes. Instead of modelling many individual processes, however, we have currently abstracted away into a single 'align' process that consumes operations resource for an average job time set by the change management request rate. The average job time and change management rate are two operations metrics that would be useful to measure within the organisation.

This abstraction to highly aggregated components of the system has advantages, such as conciseness, and efficiency. Moreover, such a modelling style is strongly suggested by the form of Demos2k with its lack, as discussed above, substantial structural components (of course, we could code them in for any given model, but there would be significant loss of clarity, efficiency, and modelling rigour).

In the security domain, however, a concept of location, or place, arises naturally and essentially. For example, an infection starts at some place(s) then spreads around the system following an epidemiological pattern that depends, among other things, on the connectivity of the system components. Understanding the topological properties of the system such as connectivity is therefore essential to determining the appropriate deployment of countermeasures. The integration into our modelling framework of a well-founded account of location is thus a further strand of our ongoing research.

The remaining active strand of research in this area concerns the key security concept of access control. We can consider a user or a group of users, or role, that is to say, a *principal*, to be a process that is being executed on a system, i.e., relative to some resources that are situated at locations. Building on some ideas introduced by Abadi et al (1993) we are able to make mathematically precise their conception that impersonation — hence groups, roles, etc — is a form of concurrent composition of the impersonated and impersonating principals. Using the ideas from logic to which we alluded in §1.2. These ideas are sketched in Monahan and Pym (2006).

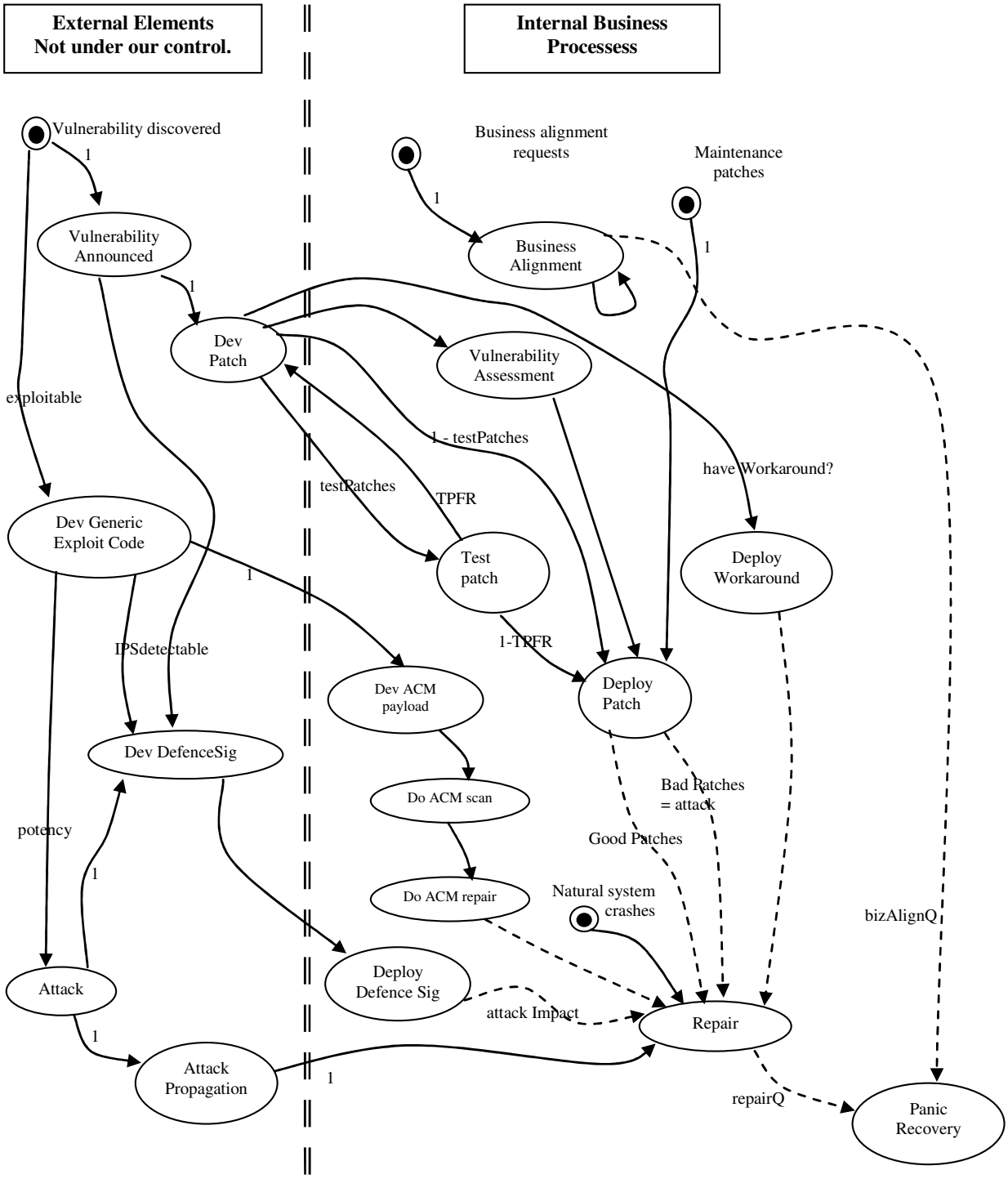

Figure 2. High-level process view of causative spawning dependencies in the model. A (full) arrow from process A to process B indicates that process A spawns process B; where several processes may be (concurrently) spawned by a process. Also, spawning may be probabilistic - numbers on arrows generally indicate probability of spawning. Importantly, note that the probabilities on arrows issuing from a process tend not to add up to 1 - this is because spawning is often not exclusive and yet concurrent. Furthermore, spawned processes could overlap in time with their parents - but often succession is intended. Finally, the dotted arrows indicate other significant dependencies tying processes together, possibly labelled by important attributes (note: - they are not probabilities).

4.2. Description of the model

The Demos2k model corresponding to Figure 2 is shown in Appendix A.

4.2.1. Processes consuming security operations staff resources

The following internal processes consume security operations staff resources:

1. Continuous change management (relationship between users, resources and processes) which has been aggregated into a single business alignment process;
2. Repair compromised machines;
3. Processes associated with HP Active Counter Measures such as develop payload, perform scan and take repair actions arising from 'forced' disconnections;
4. Processes associated with patch management including testing and deployment;
5. Implementing workarounds as mitigating actions;
6. Deployment of IPS signatures;
7. Vulnerability assessment – determining corporate patching deadlines and associated risk levels.

4.2.2. Development of the threat model

The threat model provides the stimulus to all of the security operations tasks indicated in §4.2.1. It is an area of modelling that requires further development in its own right and is potentially useful outside the context of this paper; however we have made the following assumptions:

- All vulnerabilities are naturally present in all *unpatched* systems – they do not need to be 'caught' or 'infected';
- Viruses, Worms and Trojans (VWT) exploit vulnerabilities that remain unpatched. These VWT are the *infection agents*;
- Typically, there has to be some external access for VWT to become effective;
- Patching systems will eliminate the vulnerabilities *and* remove the effects of the infective agents.

In order to avoid situations where our model suffers from the runaway affects of an attack leading to an implausible number of unavailable machines, we have introduced a damping mechanism to model spread effects. The number of machines requiring repair and the number of machines on the repair queue will grow under attack at two different rates, where the time constant of the repair queue is slower than the number of machines requiring repair. Since the length of the repair queue would be monitored we introduce a 'panic' threshold. At the time the panic occurs the following take place:

- The organisation is effectively down for a number of days;
- Utilisation of security operations staff during this down time is 100%; and
- At the end of the down time the number of machines on the repair queue becomes equal to the number of machines that needed repair at the time of the panic.

4.3. Candidate SLAs arising from the model

Some of the constants calculated in the measurement process used in the model given in Appendix A are candidates for use as SLAs. However, the model has been used just to provide availability and this is used as the SLA for analysis.

4.4. Requirements for further data

Some of the parameters used in the model could be verified by including a corresponding measurement into the cost of security project activity undertaken by the BISMs in HP IT. The following measurements are suggested:

1. The aggregation of individual security operations staff, and FTEs, into teams that govern the concurrency limits within the model:

   Parameters: `repairTeams`, `testPatchTeams` and `bizAlignTeams`;

2. The average amount of time taken to undertake tasks and the number of security operations staff typically assigned to solve the problem:

   Parameters: `bizAlignInterval`, `bizAlignStaff` and `bizAlignTime;`

3. The average time and resources associated with the development and deployment of HP Active Countermeasures payloads:

   Parameters: `devACMpayloadTime`, `deployACMpayloadTime` and `acmScanTime;`

4. the average times taken to test patches and to fix systems:

   Parameters: `testPatchTime` and `repairSysTime`.

4.5. Candidate Metrics arising from the model

The following is a suggested list of new metrics for HP IT to deploy. Not all of these may be measurable in practice but are included here for discussion. Conceivably some of these measures may be derivable from the existing dashboard.

1. The amount of time taken to assess the potential impact of a new vulnerability for which a patch exists before deciding to deploy:

   Parameter: vulnAssessmentTime;

2. The effectiveness of attacks, patching, workarounds and HP Active Countermeasures:

   Parameters: `attackEffectiveness, badPatchEffectiveness, patchEffectiveness, IPSsigEffectiveness, acmEffectiveness,` and `workaroundEffectiveness;`

3. Deployment coverage measurements associated with patch management, HP Active Countermeasures, Virus signature distribution and workarounds;

   Parameters: `patchDeploymentCoverage, sigDeploymentCoverage, acmDeploymentCoverage` and `workaroundDeploymentCoverage;`

4. The characteristics of specific attacks:

   Parameters: `exploitable, potency, IPSdetectable` and `needsWorkaround;`

5. The characteristics of the patching process:

   Parameters: `probPatchIsBad, probTestRejectsBadPatch, probTestRejectsOKPatch` and `probPatchTest.`

5. Results

The results from the model presented here should be considered as preliminary demonstrating the basic feasibility of the approach.

The model has been used to investigate two key questions; (i) the impact on predicted availability of a system and utilisation of reducing the number of security operations staff resources, and (ii) the impact on availability and utilisation if the threat environment changes. Some of the runs initially produced what could only be regarded as devastating loss of availability due to attack such that the organisation would have to be considered as non-viable. This was rectified by the creation of a more

elaborate attack model described in §4.2.2 which then allowed for the modelling of a corresponding set of defensive processes.

One of the main requirements was to model a system of the size of HP. Therefore we have chosen a system with 100,000 devices. Given that the parameters in the model have not been calibrated we chose a 'reasonable' starting set and explored the number of security operations staff that gave a utilisation rate close to 70%. This gave a baseline model where 55 security operations staff were required.

In order to acquire reasonable statistics the simulation run of 365 days was repeated 100 times. Using the model as shown in Appendix A as a baseline we obtain an average predicted availability of 98.97%. The corresponding average utilisation of the security operations staff was 75.18%.

Data from a typical run with N_{panics}=0 showing daily availability over a period of 365 days is shown in Figure 3.

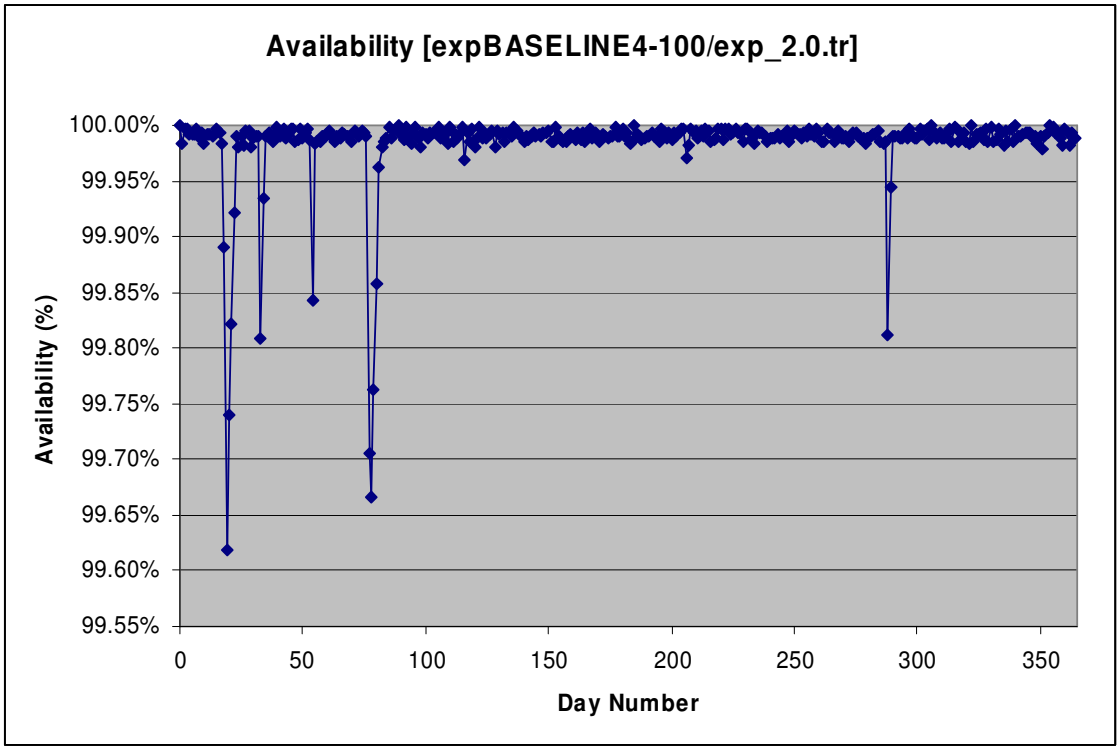

Figure 3. Daily availability versus day number from a typical simulation run. Superimposed on the small impact on availability due to the rate of change management requests arising from misalignment are more significant dips arising from vulnerabilities.

With all other parameters held constant the effect of reducing the number of operations staff by 10% and doubling the rate at which vulnerabilities are discovered have been investigated and summarised in

Table 1.

Case	N_{ops}	Vulnerability Rate	Availability (Average)	Availability (Min)	Availability (Max)	Utilisation (Average)	Utilisation (Min)	Utilisation (Max)	$N_{panics}=0$	$N_{panics}=1$	$N_{panics}=2$	$N_{panics}\geq 3$
1	55	negexp(365/12)	98.97%	69.34%	99.99%	75.18%	67.13%	92.90%	62%	19%	7%	12%
	N_{ops}	Vulnerability Rate	Availability (Average)	Availability (Min)	Availability (Max)	Utilisation (Average)	Utilisation (Min)	Utilisation (Max)	$N_{panics}=0$	$N_{panics}=1$	$N_{panics}=2$	$N_{panics}\geq 3$
2	50	negexp(365/12)	-0.75%	-1.75%	0.00%	+6.46%	+8.49%	+2.22%	36%	29%	15%	20%
3	55	negexp(365/24)	-2.04%	-5.80%	0.00%	+6.74%	+6.44%	+2.75%	23%	33%	11%	33%
4	50	negexp(365/24)	-2.10%	-2.21%	0.00%	+11.66%	+12.19%	+2.27%	5%	23%	21%	51%

Table 1. Summary of data for 4 scenarios changing the number of operations staff and the rate of discovery of vulnerabilities. The data in the first row are presented as a baseline case. Data for availability and utilisation for cases 2, 3 and 4 are presented as deltas.

Exploration of other questions such as change in patch coverage rates and impact of use of HP's Active Countermeasures, will be covered in a subsequent paper.

In terms of meeting the requirement of achieving cost effective modelling each 365 day simulation takes approximately 120 seconds to run on a 1.6GHz Pentium notebook machine.

6. Discussion and conclusions

In this paper, we have described a framework within which we can develop meaningful security SLAs; that is, we have outlined a mathematically well-founded modelling system, partially realized within the Demos2k tool, which allows us to predict the availability of systems that are vulnerable to downtime arising from security vulnerabilities and business misalignments. The model presented here has fulfilled the basic requirement of providing a predictive capability for the candidate SLA for consideration by the HP IT CSO for inclusion in the monthly dashboard. We have used the (lack of) availability directly arising from vulnerabilities and misalignment and provided a prediction of the impact on availability of the system when the number of security operations staff resources is reduced and when the threat environment is raised thus demonstrating the capability to answer some of the basic questions posed in §1.3.

Results obtained from the model show that based on our current understanding of the dynamics of security operations that we have direct link between the utilisation rate for security operations staff and reduction in overall resources available and increase in threat environment. We have also encountered runaway conditions due to attack which illustrate the need for better models of attack mechanisms and defensive processes. The rate of encountering these situations is also linked to reductions in availability of operations staff and increased threat. Empirical evidence suggests that the high number of *panics* predicted by this model is unrealistic and that a better model would divide panics into $type_1$ and $type_2$; where $type_2$ would correspond to the existing model, but $type_1$ would describe an emergency mitigation process employing all available resources to give a rate of $type_1$ panics ~3 per year and type $_2$ of ~0.3 per year. $Type_1$ panics would not involve the loss of availability of systems due to vulnerability, unlike $type_2$, but would prevent the normal stream of change management processes taking place and thus contribute somewhat to a reduced level of availability. This is an active area of further work.

The results indicate that the gross dynamics of the model are what would be expected. The goal of further work would be to better understand the attack and defensive process dynamics and to provide some calibration on any predicted change in availability rather than chase after any precision in exact levels of availability. This will be the initial task in future work. We can then go on to further explore a number of questions including patch deployment processes and the impact of technologies such as HP Active Countermeaures.

We have also identified a number of new measurements that should be included in the cost of security project undertaken by the BISMs and a set of potential metrics that could be used for improving the utility of predictive modelling. The model has also fulfilled another basic requirement in that it is cost effective to run. We have also outlined our ongoing research to support modelling techniques that will support the structurally rich space of systems security questions.

Acknowledgements

Many people have contributed their valuable time and ideas to this paper. In the office of the CSO we would like to thank Sam Horowitz, Mark Wickham, Andy Radle and George McKee and in HP Labs we would like to thank Mike Wonham (HPL BISM), Rich Smith (threat modelling), Jonathan Griffin (Active Countermeasures), Chris Tofts (Demos2k) and Richard Taylor (Open Analytics). We would also like to thank Philippe Lamy and Doug Young for valuable discussions on business development.

References

Abadi, M., Burrows, M., Lampson, B., Plotkin, G., A calculus for access control in distributed systems. *ACM Trans. Prog. Lang. Sys.* 15(4):706 – 734, 1993

Birtwistle, G., Tofts, C., The Operational Semantics of Demos: part I, *Transactions of the Simulation Society* 10(4):299 – 333, 1993

Birtwistle, G., Tofts, C., The Operational Semantics of Demos: part II, *Transactions of the Simulation Society*, 11(4):303 – 337, 1995

Christodolou A., Taylor R., Tofts C., 2000. Demos 2000. http://www.demos2k.org

Gordon, L., Loeb, M., 2006. *Managing Cybersecurity Resources: A Financial Perspective*. New York. McGraw Hill Higher Education

Milner, R., 1989. *Communication and Concurrency*. Hemel-Hempstead. Prentice Hall International

Monahan, B., Pym, D.,2006. A Structural and Stochastic Modelling Philosophy for Systems Integrity. http://library.hp.com/techpubs/2006/HPL-2006-35.html

Monahan, B., Yearworth, M., 2005. Meaningful Security SLAs

Pym, D. and C. Tofts, 2004. A Calculus and Logic of Resources and Processes. http://library.hp.com/techpubs/2004/HPL-2004-170R1.html

Taylor, R., Tofts, C., 2003. Business as a Control System: the essence of an intelligent enterprise. http://library.hp.com/techpubs/2003/HPL-2003-247.html

Taylor, R., Tofts, C., Yearworth, M., 2004. Open Analytics.

Tofts, C., Taylor, R., 2003. A Reliability Calculator You Can Rely On.

Appendix A

```
// Security operations model - Mike Yearworth, Brian Monahan & David Pym
// 10th March 2006

// Measurement
cons holdTime                  = 1*365;    // sim. period -- in days

// Sizes
cons systems                   = 100000;   // number of systems
cons Nops                      = 55;       // number of operations staff

// Rates
consFailureRate                = uniform(1/1000, 2/1000);

// Concurrent Work Streams
cons bizAlignTeams             = 40;
cons repairTeams               = 40;
cons testPatchTeams            = 10;

// Damping/spread parameters
cons maxAttackSpread           = 500;
cons maxRepairPerDay           = 400;

// Accounting
cons accountingSamplesPerDay   = 96;

// Emergency recovery action (i.e. panicRecovery)
cons panicLevel                = 1000;    // emergency recovery triggered if
                                          // either repairQ or bizAlignQ exceeds this level
cons suspensionPeriod          = 2;       // number of days to suspend business
                                          // emergency recovery triggered

// IT tasks due to Business (mis)Alignment
cons bizAlignInterval          = negexp(1/200);
cons bizAlignStaff             = puni(1, 3);
cons bizAlignTime              = uniform(1/24, 3/24);

// ACM
cons applyACM                  = 1;
cons devACMpayloadTime         = uniform(2, 5);        // time to develop ACM payload
cons deployACMpayloadTime      = uniform(1/24, 3/24);  // concurrent deployment
cons acmScanTime               = uniform(1/24, 1);     // concurrent scans

// Process Times
cons genExploitCodeTime        = negexp(40);
cons genIPSsigTime             = negexp(10);
cons genPatchTime              = negexp(30);
cons deployAttackTime          = negexp(7);
cons testPatchTime             = normal(3,0.5);
cons repairSysTime             = normal(8/24, 2/24);
cons vulnAssessmentTime        = negexp(5);

// Effectiveness levels (randomised per instance)
cons attackEffectiveness       = uniform(05/100, 25/100);
cons badPatchEffectiveness     = uniform(10/100, 15/100);
cons IPSsigEffectiveness       = uniform(95/100, 99/100);
cons acmEffectiveness          = uniform(95/100, 99/100);
cons workaroundEffectiveness   = uniform(95/100, 99/100);
cons patchEffectiveness        = uniform(95/100, 99/100);

// Deployment coverage (randomised per instance)  (METRICS??)
cons patchDeploymentCoverage   = uniform(95/100, 99/100);
cons sigDeploymentCoverage     = uniform(95/100, 99/100);
cons acmDeploymentCoverage     = uniform(95/100, 99/100);
cons workaroundDeploymentCoverage   = uniform(95/100, 99/100);

// Environmental probabilities
cons exploitable               = 50/100;
cons potency                   = 20/100;
cons IPSdetectable             = 85/100;
cons needsWorkaround           = 85/100;

// Probability of a "bad" patch arising and being rejected when tested
// A "bad" patch is like an attack - it could cause serious damage if ever deployed
// Testing the patches rejects more of the bad patches
```

```
cons probPatchIsBad            =  1/100;
cons probTestRejectsBadPatch   = 95/100;
cons probTestRejectsOKPatch    =  5/100; // probability of an OK patch being
                                         // rejected, when tested
cons probPatchTest             = 80/100; // probability of patch being tested
                                         // prior to deployment

// Checks
cons couldBeExploitable        = binom(1, exploitable);
cons couldBePotent             = binom(1, potency);
cons couldBeDetectable         = binom(1, IPSdetectable);
cons isPatchBad                = binom(1, probPatchIsBad);
cons testPatches               = binom(1, probPatchTest);
cons okPatchRejected           = binom(1, probTestRejectsOKPatch);
cons badPatchRejected          = binom(1, probTestRejectsBadPatch);
cons hasWorkaround             = binom(1, needsWorkaround);

// Attack impact
// -- Initially this is 0
// -- When attack exploit discovered, this becomes attackEffectiveness
// -- When IPS sigs are deployed, impact is reduced ...
// -- As attacks come and IPS sigs deployed, attackImpact fluctuates ..
var attackImpact               = 0.0;

// Counts and Stats
var vulnerableSystems=0;
var downSystems=0;
var liableForAttack=0;

var accounting=0.0;    var avail=1.0;          var cAvail=0.0;
var day=0;             var util=0.0;           var staffAvailable=Nops;

var isBizOperational = 1;  // indicator var == 1 when the business is operational

var cRepaired=0;           var cAttackEvents=0;    var cPatched=0;
var cVuln=0;               var cSigs=0;            var cACMrepairs=0;
var cACMscans=0;           var cBadPatch=0;        var cVulnAnnounce=0;
var cWorkarounds=0;        var cCrashEvents=0;     var cPanics=0;
var cAttacked=0;           var cCrashed=0;


// Resources
res(opsStaff,Nops);    // Operations staff
res(lock, 1);          // semaphore for concurrency update control (hygiene)

// Bins
bin(bizAlignQ,      0);
bin(repairQ,        0);
bin(patchQ,         0);
bin(testQ,          0);
bin(patchDeadlineQ, 0);


//////////////////////
// Classes
//////////////////////

////////////////////////////////////
// External processes/activities
////////////////////////////////////

class vulnerable = {
  repeat{
    hold(vulnerabilityDiscovered);

    getR(lock, 1);
      cVuln := cVuln+1;
      vulnerableSystems:=systems;
    putR(lock, 1);

    try [couldBeExploitable == 1] then { entity(DXC, devGenericExploitCode, 0); }
    etry [] then {hold(0);}

    entity(VulnAnnounce, vulnerabilityAnnouncement, 0);
  }
}
```

```
class vulnerabilityAnnouncement = {
  hold(vulnerabilityAnnounceIntv);     // wait until the vulnerability is announced
  entity(ASV, assessVulnerability, 0); // assess vulnerability
  entity(DP, devPatch, 0);             // develop patch

  cVulnAnnounce:=cVulnAnnounce+1;

  try [couldBeDetectable == 1]  then { entity(DS, developIPSsig, 0); }
  etry [] then {hold(0);}
}

class devGenericExploitCode = {
  hold(genExploitCodeTime);
  getR(lock, 1);
    // attackImpact updated in some way to take account of new exploit being found
    // This should depend upon (a) current attackImpact and (b) new attackEffectiveness
    // Crudely, we combine them by taking the average - preserves 0 <= attackImpact <= 1
    //
    attackImpact := (attackImpact + attackEffectiveness)/2;
  putR(lock, 1);

  try [couldBePotent == 1] then { entity(A, attack, 0); }
  etry [] then {hold(0);}

  // IPS signatures might be developed based upon found exploit ...
  try [couldBeDetectable == 1]  then { entity(DS, developIPSsig, 0); }
  etry [] then {hold(0);}

  try [applyACM== 1] then { entity(ACMpayload, developACMpayload, 0); }
  etry [] then {hold(0);}
}

class devPatch= {

  // workround may be issued prior to the release of a patch ...
  try [hasWorkaround == 1] then { entity(workaround, deployWorkaround, 0); }
  etry [] then { hold(0); }

  hold(genPatchTime);
  putB(testQ, 1);  // put patch into test Q
}

class attack = {
  hold(deployAttackTime); // concurrent deployment

  // development of a attack vector allows further opportunity for
  // IPS signatures to be found.
  try [couldBeDetectable == 1]  then { entity(DS, developIPSsig, 0); }
  etry [] then {hold(0);}

  req[isBizOperational == 1]; // can't attack anything if systems aren't up

  getR(lock, 1);
    cAttackEvents := cAttackEvents+1;

    // can only attack vulnerable systems that are still up ...
    try [vulnerableSystems > downSystems] then {
      // attackImpact moderates the number of items that can be attacked.
      // Also, attack doesn't take things down immediately - there is a "spread" effect
      liableForAttack := rnd((vulnerableSystems - downSystems)*attackImpact);
    }
    etry [] then { hold(0); }

  putR(lock, 1);
}

// This is more subtle than the other damping processes -  mainly because recovery
// is happening at the same time, and the vulnerable machines/down systems is changing
// simultaneously ...
class attackDamping = {
  local var attacked   = 0; // the instantaneous number of machines attacked.
  local var attackable = 0; // the instantaneous number of attackable machines.

  repeat{
    hold(1); // once a day

    req[isBizOperational == 1]; // can't crash anything if systems aren't up
```

```
    getR(lock, 1);
      attacked := 0;

      try [vulnerableSystems > downSystems] then {
        attackable := (vulnerableSystems - downSystems);

        // We adjust liableForAttack in the light of the instantaneous values of machines
        // available for attacking.  This is because attacks now take some time to
        // spread/transmit.
        //
        try [liableForAttack > attackable] then { liableForAttack := attackable; }
        etry [] then { hold(0); }

        // min of maxAttackSpread and liableForAttack
        try  [liableForAttack > maxAttackSpread] then { attacked := maxAttackSpread; }
        etry [] then { attacked := liableForAttack; }

        // Note that: when attackable >= 0, then attacked =< liableForAttack =< attackable

        cAttacked:=cAttacked+attacked;

        // Deduct those attacked from liableForAttack
        // and add to downSystems
        liableForAttack := liableForAttack-attacked;
        downSystems := downSystems+attacked;
      }
      etry [] then { liableForAttack := 0; }
    putR(lock, 1);
  }
}

class crash = {
  local var crashed = 0;

  req[cVuln > 0]; // crashes need some vulnerabilities
  repeat{
    hold(crashIntv);

    req[isBizOperational == 1]; // can't crash anything if systems aren't up

    getR(lock, 1);
      cCrashEvents := cCrashEvents+1;

      // can only crash vulnerable systems that are still up ...
      try [vulnerableSystems > downSystems] then {
        crashed := rnd((vulnerableSystems - downSystems)*sysFailureRate);
        cCrashed := cCrashed+crashed;
        downSystems := downSystems + crashed;
      }
      etry [] then { hold(0); }

    putR(lock, 1);
  }
}

class developIPSsig= { hold(genIPSsigTime); entity(USS, deployIPSsig, 0); }


////////////////////////////////////////////////
// Corporate activities
////////////////////////////////////////////////

class bizAlignRequests = {
  repeat{
    hold(bizAlignInterval);

    // business must be operational
    req[isBizOperational == 1];

    putB(bizAlignQ, 1); // add task directly to the bizAlignQ
  }
}


class bizAlignService = {
  local var ITstaffNeeded = 0;
```

```
  repeat{
    getB(bizAlignQ, 1);

    // business must be operational
    req[isBizOperational == 1];

    ITstaffNeeded := bizAlignStaff;

    getR(opsStaff, ITstaffNeeded);  // get staff

    // needs a system to take down ...
    req [systems >= downSystems + 1];
    getR(lock, 1); downSystems := downSystems + 1; putR(lock, 1);

    hold(bizAlignTime);         // do bizAlign task

    // restores system to operational (assuming it hasn't already been repaired :) )
    getR(lock, 1);
      try [downSystems > 0] then { downSystems := downSystems-1; }
      etry [] then {hold(0);}
    putR(lock, 1);

    putR(opsStaff, ITstaffNeeded);  // release staff
  }
}

class assessVulnerability = {
  // business must be operational
  req[isBizOperational == 1];

  getR(opsStaff, 10);
  hold(vulnAssessmentTime);
  putR(opsStaff, 10);

  putB(patchDeadlineQ, 1);  // generate a "patching deadline"

}

class maintenancePatching = {
  repeat{
    hold(maintenancePatchIntv);

    // business must be operational
    req[isBizOperational == 1];

    putVB(patchQ, [0]); // maintenance patches are always OK (patchOK = 0) ...
  }
}

class deployWorkaround = {
  local var saved = 0;

  // business must be operational
  req[isBizOperational == 1];

  getR(lock, 1);
    saved := rnd(systems*workaroundDeploymentCoverage);
    cWorkarounds := cWorkarounds + saved;
    putB(bizAlignQ, 10); // add in some biz alignment knock on effect
    try [vulnerableSystems > 0] then {
      // Assumption: Effect of patching on vulnerable systems is proportional to number of
      // systems patched
      vulnerableSystems:=rnd(vulnerableSystems*(1 - (saved/systems)));
    }
    etry [] then {hold(0);}
  putR(lock, 1);
}

class deployIPSsig = {
  local var sigCoverage = 0;

  // business must be operational
  req[isBizOperational == 1];

  getR(lock, 1);
    sigCoverage := sigDeploymentCoverage;
    cSigs := cSigs+rnd(systems*sigCoverage);
```

```
    putB(bizAlignQ, 10); // add in some biz alignment knock on effect
    attackImpact := attackImpact*(1 - sigCoverage)*(1 - IPSsigEffectiveness);
  putR(lock, 1);
}

// Provides a "damping" effect on adding repairs to the queue.
// Explained by phenominological aspects: social effect, network effects, queuing policy
class repairDamping = {
  local var needsRepair = 0;
  repeat{
    hold(1);

    // business must be operational
    req[isBizOperational == 1];

    getR(lock, 1);
      // set needsRepair to the *excess* of down systems not yet on the repair Q
      needsRepair := (downSystems - AV_repairQ);

      try [needsRepair > 0] then {

        // limit needsRepair to maxRepairPerDay
        try [needsRepair > maxRepairPerDay] then { needsRepair := maxRepairPerDay; }
        etry [] then { hold(0); }

        putB(repairQ, needsRepair);
      }
      etry [] then { hold(0); }
    putR(lock, 1);
  }
}

class repair = {
  // repair is the only process that restores down systems (i.e. cure)
  //
  repeat{
    getB(repairQ, 1);  // take a machine off the repair queue

    // business must be operational
    req[isBizOperational == 1];

    cRepaired := cRepaired+1;

    getR(lock, 1);
      // Now, systems may be down but not vulnerable ...
      // assume that a repair always restores a down machine
      // and clears out vulnerability (whenever poss.)
      try[vulnerableSystems > 0] then {
        vulnerableSystems := vulnerableSystems-1;
      }
      etry [] then {hold(0);}

      // It seems that by the time a machine is due to be fixed,
      // it may no longer be regarded as "down".
      // Note that downSystems is not immediately reduced by placing
      // something in the repair Q - and thus there could be discrepancy.
      try [downSystems > 0] then {
        downSystems := downSystems-1;
      }
      etry [] then { hold(0); }
    putR(lock, 1);

    getR(opsStaff, 1);
      hold(repairSysTime);
    putR(opsStaff, 1);
  }
}

class panicRecovery = {
  local var currentDownSystems = 0;

  getR(lock, 1);
    cPanics:=cPanics+1;

    isBizOperational    := 0;              // switch the business off
    currentDownSystems  := downSystems;    // capture current down Systems
    downSystems         := systems;        // business is now off-line - so everything is
```

```
                                          // notionally down.
  putR(lock, 1);

  // quickly deflate the queues
  entity(eatRepairQ,   deflateQueue(repairQ, #AV_repairQ), 0);      // deflate repair Q
  entity(eatBizAlignQ, deflateQueue(bizAlignQ, #AV_bizAlignQ), 0);  // deflate bizalign Q

  hold(suspensionPeriod);

  getR(lock, 1);
    // Assumption: During Emergency recovery, all the truly down systems are identified
    // and still need to be fixed.  We don't really discriminate between infected and
    // a system being "down".
    downSystems       := currentDownSystems;    // reset to the "truly" down systems
    isBizOperational  := 1;                     // now switch the business back on ...
  putR(lock, 1);
}

class deflateQueue(queue, amount) = {
  local var deflateCount = amount;

  while [deflateCount > 0] {
    hold(1/1440); // 24*60
    getB(queue, 1);
    deflateCount:=deflateCount-1;
  }
}

class developACMpayload = {
  // business must be operational
  req[isBizOperational == 1];

  getR(opsStaff, 6);
  hold(devACMpayloadTime);
  putR(opsStaff, 6);
  entity(ACMScan, acmScanAndRepair, 0);
}

class acmScanAndRepair = {
  // business must be operational
  req[isBizOperational == 1];

  cACMscans := cACMscans+1;

  getR(opsStaff, 6);
  hold(acmScanTime); // concurrent scans
  putR(opsStaff, 6);

  entity(acmRep, acmRepair, 0);
}

class acmRepair = {
  local var ACMrepaired = 0;

  // business must be operational
  req[isBizOperational == 1];

  // remember that ACM only deploys against vulnerable systems
  try [vulnerableSystems > 0] then {
    getR(opsStaff, 6);
    hold(deployACMpayloadTime);  // concurrent deployment
    getR(lock, 1);
      try [vulnerableSystems > downSystems] then {
        ACMrepaired := rnd((vulnerableSystems - downSystems)*acmEffectiveness);
        cACMrepairs := cACMrepairs + ACMrepaired;
        vulnerableSystems:= vulnerableSystems - ACMrepaired;
      }
      etry [] then { hold(0); }
    putR(lock, 1);
    putR(opsStaff, 6);
  }
  etry [] then {hold(0);}
}

class testPatch = {
  local var patchBad = 0;
```

```
  repeat{
    // business must be operational
    req[isBizOperational == 1];

    getB(testQ, 1);
    patchBad := isPatchBad;   // probabilisticlly assign patch status - is it a bad patch?
    try [ testPatches == 1] then {

      // test the patch ...
      getR(opsStaff, 4);
      hold(testPatchTime);
      putR(opsStaff, 4);

      try [patchBad == 1,  badPatchRejected == 1] then { entity (DP, devPatch, 0) ;}
      etry [patchBad == 0, okPatchRejected==1] then { entity (DP, devPatch, 0) ;}
      etry [] then { putVB(patchQ, [patchBad]); }

    }
    etry [] then { putVB(patchQ, [patchBad]); }
  }
}

class deployPatch = {
  local var patchBad = 0;
  repeat{
    // business must be operational
    req[isBizOperational == 1];

    // Consume any patching deadline - when present (i.e non-blocking) ...
    // Currently model doesn't track which patches are associated with particular
    // vulnerability
    // Also we can't properly model deadlines - so can't see effect of missing them
    try [getB(patchDeadlineQ, 1)] then { hold(0); }
    etry [] then { hold(0); }

    getVB(patchQ, [patchBad], true);

    try [patchBad == 1] then {
      entity(P, badPatch, 0);
    }
    etry [] then {
      entity(P, patch, 0);
    }
  }
}

// A bad patch (if deployed) resembles an attack ...
// except that it takes down arbitrary number of your (currently working) machines
// vulnerable or not ...
class badPatch = {
  local var newDown = 0;

  // business must be operational
  req[isBizOperational == 1];

  getR(lock, 1);
    try [systems > downSystems] then {
      cBadPatch := cBadPatch+1;
      newDown := rnd((systems - downSystems)*badPatchEffectiveness);
      downSystems := downSystems + newDown;
    }
    etry [] then {hold(0);}
  putR(lock, 1);
}

class patch = {

  local var patched = 0;

  // business must be operational
  req[isBizOperational == 1];

  getR(lock, 1);
    try [systems > downSystems] then {
      // can only patch systems that are still up
      patched := rnd((systems - downSystems)*patchDeploymentCoverage*patchEffectiveness);
      cPatched:=cPatched+patched; // total number patched - vulnerable or not
```

```
      // Assumption: Effect of patching on vulnerable systems is proportional to number of
      // systems patched
      // i.e. reduced by a factor of patched/systems
      vulnerableSystems:=rnd(vulnerableSystems*(1 - (patched/systems))); }
    etry [] then {hold(0);}
  putR(lock, 1);
}

////////////////////////////
// Observers
////////////////////////////

class manpowerAccounting = {
  local var weighting = (Nops*accountingSamplesPerDay);
  repeat{
    hold(1/accountingSamplesPerDay);
    getR(lock, 1);
      try [isBizOperational == 1] then {
        staffAvailable := AV_opsStaff;

        // check if emergency recovery is required
        try [(AV_repairQ > panicLevel || AV_bizAlignQ > panicLevel)] then {
           entity(panic, panicRecovery, 0);
        }
        etry [] then { hold(0); }
      }
      etry [] then {
        staffAvailable := 0; // all staff are committed to emergency recovery
      }

      // account for the instantaneous number of ops staff busy
      accounting := accounting+(Nops-staffAvailable)/weighting;
    putR(lock, 1);
  }
}

class measure = {
  entity(M,measure,1);
  getR(lock, 1);

    try [downSystems < 0 ] then { close; }
    etry [] then {hold(0);}

    avail:=(systems-downSystems)/systems;
    cAvail:=cAvail+(avail/holdTime);
    util:=util+(accounting/holdTime);

    trace("isBizOperational=%v", isBizOperational);

    trace("day=%v vulnerableSystems=%v downSystems=%v", day, vulnerableSystems, downSystems);
    trace("availability=%v cAvail=%v util=%v", avail, cAvail, util);
    trace("staffAvailable=%v", staffAvailable);
    trace("liableForAttack=%v", liableForAttack);
    trace("cVuln=%v cVulnAnnounce=%v", cVuln, cVulnAnnounce);
    trace("cCrashEvents=%v cAttackEvents=%v cRepaired=%v cPatched=%v",
           cCrashEvents, cAttackEvents, cRepaired, cPatched);
    trace("cCrashed=%v cAttacked=%v", cCrashed, cAttacked);
    trace("cWorkarounds=%v", cWorkarounds);
    trace("cACMscans=%v cACMrepairs=%v cSigs=%v", cACMscans, cACMrepairs, cSigs);
    trace("cPanics=%v", cPanics);
    trace("cBadPatch=%v", cBadPatch);
    trace("attackImpact=%v", attackImpact);


    accounting := 0; // reset accounting variable
    day:=day+1;
  putR(lock, 1);
}

// Entities

// Observer
entity(M, measure, 0);
entity(A, manpowerAccounting, 0);

// External source processes
```

```
entity(V, vulnerable, 0);
entity(C, crash, 0);

// External autonomous activities
entity(attDamp, attackDamping, 0); // crudely models the effect of "spread"

// Internal source processes
entity(baeq, bizAlignRequests, 0);
entity(BDP, maintenancePatching, 0);

// Internal autonomous activities
entity(repDamp, repairDamping, 0);
entity(DPP, deployPatch, 0);

do repairTeams    {entity(REP, repair, 0);}
do testPatchTeams {entity(TP, testPatch, 0);}
do bizAlignTeams  {entity(sbAREQ, bizAlignService, 0);}

// Run the simulation ...
hold(holdTime);
close;
```