Security Analytics: Analysis of Security Policies for Vulnerability Management

Yolanta Beres, Jonathan Griffin, Simon Shiu
HP Laboratories
HPL-2008-121

Keyword(s):
security, analytics, risk management, threat, vulnerability, patch, policy, intrusion prevention, modelling, simulation

Abstract:
In this paper we present a novel approach of using mathematical models and stochastic simulations to guide and inform security investment and policy change decisions. In particular, we investigate vulnerability management policies, and explore how effective standard patch management and emergency escalation based policies are, and how they can be combined with earlier, pre-patch mitigation measures to reduce the potential exposure window.

To achieve that we have examined the current practices across several large organizations, and based on this we construct the model of external events and of internal decision points and security processes that the vulnerability management consist of. We show, based on the experimental simulations, how changes in various internal parameters of the model, such as the patching timeline and the effectiveness of early mitigation measures affect the overall exposure window in terms of the time it takes to reduce the potential risk. This enables further analysis of the trade off between investing in improving patching processes, versus adding more mitigation mechanisms that can be put into effect earlier.

We believe that this type of mathematical modelling and simulation-based approach provides a novel and useful way of considering security investment decisions, which is quite distinct from traditional risk analysis.

External Posting Date: September 30, 2008 [Fulltext] - Approved for External Publication

Internal Posting Date: September 30, 2008 [Fulltext]

To be published in Annual Computer Security Applications Conference, ACSAC 2008

Security Analytics: Analysis of Security Policies for Vulnerability Management

Yolanta Beres, Jonathan Griffin, Simon Shiu

Secure Systems Lab, HP Labs, Bristol, UK

yolanta.beres, jonathan.griffin, simon.shiu@hp.com

Abstract

In this paper we present a novel approach of using mathematical models and stochastic simulations to guide and inform security investment and policy change decisions. In particular, we investigate vulnerability management policies, and explore how effective standard patch management and emergency escalation based policies are, and how they can be combined with earlier, pre-patch mitigation measures to reduce the potential exposure window.

To achieve that we have examined the current practices across several large organizations, and based on this we construct the model of external events and of internal decision points and security processes that the vulnerability management consist of. We show, based on the experimental simulations, how changes in various internal parameters of the model, such as the patching timeline and the effectiveness of early mitigation measures affect the overall exposure window in terms of the time it takes to reduce the potential risk. This enables further analysis of the trade off between investing in improving patching processes, versus adding more mitigation mechanisms that can be put into effect earlier.

We believe that this type of mathematical modelling and simulation-based approach provides a novel and useful way of considering security investment decisions, which is quite distinct from traditional risk analysis.

1. Introduction

The security operations team in a typical organization has a number of security controls (policies) such as patching, antivirus, client side firewalls, and so on, that work together in minimizing the exposure of organization's systems to security threats. But it is notoriously difficult to evaluate how well security processes are protecting an organization, and even harder to estimate in advance, the impact of a change in security mechanism investment choice or a change in policy. Examination of historical data would give only partial answers; e.g. it is certainly possible to track how long the deployment of a particular patch takes to be installed across a set of systems, but without the context of an external threat environment, the historical data (such as it is) cannot help determine if the systems were left exposed for too long – thus yielding unacceptable risk.

Security decisions often involve trade-offs: a security policy choice that optimizes time spent by the security team might create burdens (cost) for IT operations or the business; and a decision to spend money defending one risk means reduced resources for another. One of the main tasks faced by the security operations team is vulnerability and patch management. With thousands of systems running popular business operating systems such as Windows in a medium to large size organization, all of these systems may potentially require patches to be deployed. However, deploying patches across all of these systems in a timely manner is not simple. In addition to the time spent for the patch assessment and patch testing, the security operations team often faces restrictions for deploying the patches placed by business in terms of allowed system downtime and minimal business disruption.

Overall, the security operations team needs to consider if the current practice in patch deployment is exposing an organization to an unacceptable risk of potential exploitation, explore the improvements that can be made in the process, or evaluate effectiveness of other mitigations in addition to patching. Of course they also need to weigh the costs of these improvements and mitigations against the potential costs from vulnerability exploits. From what we have observed, beyond examining historical data, security operations staff have few tools to help them understand trade-offs or test the different vulnerability mitigation strategies before putting them in practice.

In this paper we propose using a simulation-based, predictive modelling approach to help explore the space of outcomes for a range of patching policies and mitigation mechanisms. In particular, we analyze the effects of adjusting the patching policies based on the criticality of vulnerabilities and the likelihood of it being exploitable. We additionally investigate the impact of the deployment of other mitigations that are signature and behavior based, to explore to what extent they help minimize the potential exposure window.

The predictive modeling approach provides some advantages over a purely analytical approach as it offers more opportunities to naturally explore, via experimental results, the dependencies among different decisions in the patch management process and the impact of change in vulnerability and exploit rates. This type of approach requires first constructing a dynamic model of the operational environment, such as the patching process, which are explored empirically using executable models (e.g. discrete-event simulation) to derive probability distributions of potential outcomes. If the behaviour of the model correctly matches the relevant behaviour characteristics of the underlying environment, we are able to draw inferences about the effects from experiments using the model and thus provide support for adopting specific policies.

For constructing models we take a control-systems perspective. In such a model, we typically represent various operational entities, such as patching in terms of processes that respond to events occurring at particular rates and with particular probability. We show how we constructed the model to reflect the different decision points in vulnerability assessment and patch deployment processes of a typical large organization. Typically the speed of such processes depends upon the external threat environment, and so we capture in the model events such as vulnerability exploit arrival times, patch release times by vendors, and the arrival times of the `malware' that exploits patchable vulnerabilities.

An important goal of producing these models is to enable the security operations teams to predict the outcome of investment decisions or changes in policy in advance of putting them into effect. We show, based on the experimental simulations, how changes in various internal parameters of the model, such as the rate of exploits or patching timeline, affect the overall exposure window in terms of the number of systems exposed and the time it takes to remediate them.

Our paper is organized as follows. Section 2 reviews related work in this area. Section 3 describes our modeling approach and the advantages of using executable models in the form of discrete event simulations. In section 4, we present how we have constructed the model of the security operations process, capturing the specific tasks undertaken by security operations and the speed at which these tasks are progressed. An important part of this model are the external environment elements that trigger specific tasks, such as the rate of vulnerability related exploits, rate and speed of related malware and the speed for patch and signature development from vendors. Thus section 4 also describes how we selected the parameters from the external environment for use in our model. In section 5 we describe our design of a set of simulation experiments using the constructed model together with detailed analysis of the results obtained. The first half of our analysis aims to determine if the standard patching policy used by many security operations centers are effective at reducing vulnerability exposure to an acceptable level. The second half evaluates the impact of including additional mitigations which can be put into effect earlier, and also the impact of faster patch deployment. Our paper finishes with a discussion of the implications of our analysis for future work and presents some final conclusions.

2. Related Work

In recent years there has been a lot of research interest in the topic of security economics, exploring the trade-offs between costs of security and loss from exposure. Anderson [Anderson 01] argued that security problems should be viewed from an economic view point in addition to the technical challenges of Information Security. Gordon and Loeb [Gordon 02, 06] examined how trade-offs between cost and potential loss should be balanced when evaluating the optimal investment for a single security policy. This economic viewpoint to security is critical for our work, as we aim to analyze both cost and exposure risk trade-offs of different security policies.

Another stream of research relevant to our work here is the analysis of the vulnerability life-cycle, and their past and future trends. The earlier work by McHugh and Arbaugh [McHugh 00] introduces a life-cycle model for system vulnerabilities. Looking at CERT data on attacks related to specific vulnerabilities, McHugh and Arbaugh noted that many systems were still being attacked months or even years after the patches became available for the vulnerabilities exploited in those attacks. Since then, security operations teams in many organizations streamlined their patch management processes so that fewer numbers of systems remain vulnerable for longer. However, the rate at which new vulnerabilities are discovered and exploited is constantly increasing, thus requiring constant reassessment of existing patching practices. The work by Frei et al [Frei 06] is important here for helping understand past and current trends of how vulnerabilities have been exploited and how mitigations have been handled by software vendors. This work looked at the generic set of vulnerabilities as disclosed by CERT, Security Focus, and others. In building our model of the external threat environment, as described in section 4.1, we have used some of distribution functions derived by the authors of this paper, such as for exploit availability rate. The set of vulnerabilities that large organisations care about usually represents only a subset of all the known vulnerabilities that could be relevant. Thus, further analysis is necessary to assess the sensitivity of the distributions functions across the various sets.

There has also been some work examining the trade-offs of different patching policies. Beattie, et al. [Beattie 02] explores the factors affecting the best time to apply patches so that organisations minimize disruptions caused by defective patches. Their results indicate that patching during the period of 10 to 30 days after first patch release date is the optimal period for minimizing the disruption caused by defective patches. In our work, we take this into account when modelling internal patching processes, allowing a varying period with a mean of 10 days as a patch testing time before the patches are deployed. However, rather than looking just at a single patch and adjusting a single point in time to start patch deployment, we model the overall patch management process that also takes into account the time taken to apply the patches across all the vulnerable systems in an organisation; this can be considerable for a large organisation.

The work by Zhang, Tan and Dey [Zhang 08] provides an analytical framework for cost-benefit analysis of different patching policies. They assume that patching lead time (the time taken to apply a patch across the systems) is negligible or very small comparing to the overall patching lifecycle, which we argue is not the case in large organisations with thousands of systems requiring the same patch. The authors also assume that the costs associated with vulnerability exploitations can be estimated with relative ease by an organisation, which in reality is very difficult to determine with any accuracy. We believe that our proposed simulation based approach allows an organisation to more naturally and flexibly explore the pragmatic outcomes from different patching policies than a purely-analytical framework would allow.

The simulation-based stochastic modeling approach has had little application in the computer security world, but it has been extensively used in the aerospace, automobile, oil, gas, steel, and telecommunications industries. Recently we have applied this approach to also examine security policies for USB memory sticks [Pym 08], and in earlier work we have examined resource contention issues such as staff utilization for network security operations tasks [Pym 07].

3. Probabilistic modelling approach

The use of mathematical models in engineering has a long and distinguished record of success. From earthworks to suspension bridges, from bicycles to spacecraft, mathematical models are used to predict behavior and give confidence that necessary properties of the constructions — such as capacity, resilience, and cost — obtain. Such applications of applied mathematics in engineering are useful, and usable, by virtue of the scientifically rigorous modeling methodology, where observations about the external environment and the parameters that the system depends upon are interpreted and a range of properties of the mathematical model are deduced. In the worlds of traditional engineering, ranging over mechanical, civil, environmental and electrical/electronic engineering, the mathematical methods used are mainly concerned with continuous phenomena and typically use techniques from calculus such as differential equations. For modeling the performance of security operations the appropriate mathematical methods are more discrete, being drawn from algebra, logic, theoretical computer science, and probability theory. In order to apply these methods, we require a conceptual analysis of the relevant aspects of the systems of interest. In this section we will describe the main system elements that have to be captured in our models and the methodology we use to construct the models.

3.1 The modelling methodology

The diagram below shows the steps typically followed when building a formal model in our framework. We seek to model the system of interest in order to answer some specific questions. The modelling process then continues with observations of the key stages and decision points of the processes involved. In this case, we are concerned with security operations processes, in particular looking at patch management and the key parameters of mitigation mechanisms, for example anti-virus. These observations induce the definition of a mathematical process model, of which properties may be deduced. A critical aspect to this is the choice of level of abstraction: it is important to represent in a model just those features of the phenomena pertinent to the properties and questions of interest. The mathematical consequences of the definition of the model are then interpreted as to how well they help explain reality and, experimentally, the extent of their validity is observed. The cycle is repeated until a model is judged to sufficiently capture enough of the decision making situation.

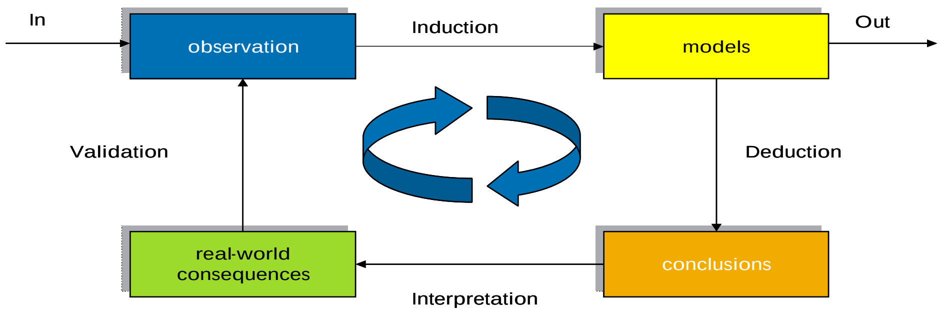

Figure 1. Modelling lifecycle.

Pictorially the way that our models are built and used is represented using activity diagrams. We found that in real life situations, it is important to have such a high level representation of the model as it helps discuss the models with the specialists who run the actual system. Feedback from the specialist is essential as not many systems work exactly as planned and documented, and in order that the simulations based on the model are useful, the model itself has to describe the system as it is actually used – which may not be as it was intended to be used when it was designed and originally implemented.

The activity diagrams are then converted into executable code, which can be used in Monte-Carlo style discrete event simulations. In this stage either extensions to general purpose programming language can be used to capture the model or specialised simulation tools.

In our case we have used the specialised simulation oriented language Demos2k [Dem, Bir79], which implements the mathematical framework based on the mathematical foundations of a synchronous calculus of resources and processes, together with an associated modal logic [Pym06]. Because of these strong mathematical foundations and sound semantics, we have assurance that simulations based on the models developed in Demos2k language are robust and reliable – thus, meaningful observations can be taken. The code is executed via repeated experimental simulations in the specially developed experimental environment [Monahan08], where statistically significant information is gathered.

3.2 Basic concepts

The mathematical framework behind the Demos2K programming language revolves around four key concepts: *resources*, capturing the essentially static components of the system; *processes*, capturing the dynamic components of the system; *location*, capturing the spatial distribution and connectivity of the system; and *environment* within which a system functions. A full description of this mathematical framework can be found in [Pym 06]. For the purpose of this paper we will briefly introduce only the two key concepts used extensively to capture operational aspects of the patch management process described in the next section:

- *Process*, which is considered to be the dynamic parts of systems, business entities or operations, and which is designed to achieve a specific purpose and/or to manipulate resources. For instance, the basic functions of operating systems or the patch management process that consists of patch assessment, testing, and deployment. In this framework the process is modelled based on a well-developed mathematical theory of processes [Milner 83, Milner 89, Pym 06].

- *Environment*. Systems, be they IT, social, business, or physical, are intended to perform functions within an environment of *events*. Such events affect systems behaviour and its state. Since there is no realistic hope of our being able to model all of pertinent environmental events exactly, it is appropriate to use probabilistic methods for modelling such events. That is, we assume that given classes of events occur — that is, are incident upon the system of interest — with given probability distributions. For example, we might assume that exploits are discovered according to a negative exponential distribution with a rate parameter λ. Given such an assumption, it is then natural to understand how the different processes are triggered by various external events, such as patch distribution by the vendor or the arrival of malware.

The model of the specific system usually consists of multiple processes which either consume resources, or take certain time to finish. As a result of one process finishing, another might be triggered. Alternatively, they might start concurrently depending on the structure and complexity of the system to be modelled. Some of the processes could be triggered by events from the environment.

Because of the strong semantic properties of these concepts in the Demos2k programming language, we have strong guarantees that during the execution of the model, the processes are executed as intended by the modeller. Demos2k efficiently handles concurrency, queuing, and prioritization among processes.

4. Constructing the Vulnerability Management Model

The need for having an explicit patching and vulnerability handling policy, together with a systematic process for handling patches, has been recognised for almost a decade. Both the early ISO standard 17799:2000 [ISO 27] and NIST publications [NIST 02] provide recommendations on principles and methodologies that can be used for accomplishing this. However, the actual implementation of the patch management process differs from one organization to another, often depending upon the resources that could be dedicated to this process, the number of systems that an organization has, and the organization’s risk appetite. To construct a model of the patch management process, we have examined patching lifecycles across several large organizations with production environments typically of more than 100,000 systems, by interviewing security policy decision makers and operational staff, and by also examining internal policy documentation.

In this section we describe the different entities and environment parameters that influence the vulnerability patching process. There are two parts of the model in this context: (1) the external environment parameters, and (2) the internal workflows and decision points in the vulnerability assessment and mitigation/patching process. Before we cover the model itself we will first describe the vulnerability timeline we use, as this affects how we examine and present results from simulations based on the model. The results from the simulations will be examined in section 5, but in designing the model that is presented in this section we had two main questions in mind that we wanted to explore via experimental runs: (1) how good are the standard patching processes at minimizing the vulnerability exposure, in terms of the time it takes and the percentage of systems exposed at certain time points; (2) what impact do different policy decisions make, such as introducing shorter patching timelines, compared to introducing additional mitigation mechanisms, such as host based intrusion prevention.

4.1. The vulnerability timeline

Figure 2 shows a timeline of events in the life cycle of a typical vulnerability. This timeline is not dissimilar to what was described by Schneier as the “window of exposure” [Schneier 00] and also examined by McHugh [McHugh 00] and Frei, et al [Frei 06].

Distinctive points in the timeline correspond to different awareness states and potentially different perceptions of risk by an organization. The four important points in this timeline are discovery, disclosure, exploit, and patch availability. The discovery point is when only the individuals who discovered the vulnerability know about it. The risk of exploit after the discovery is present, but we can assume it is minimal (or more targeted) compared to the risks after the public disclosure. A typical organization would probably be unaware of a particular threat at this stage of the lifecycle.

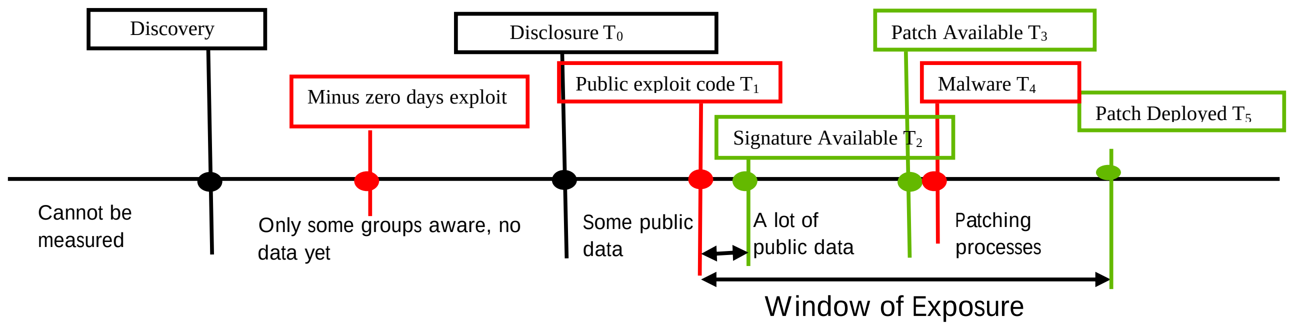

Figure 2. Vulnerability Timeline[1].

At the time of disclosure the vulnerability becomes widely known. Usually, this happens when vulnerability information is published by CERT or security vendors such as ISS, Security Focus, Secunia among others, later resulting in additional security advisory by the vendor. From this point in time the risk increases considerably, as the vulnerability is analysed by hackers with exploit code potentially appearing quickly afterwards. We assume that soon after this point, e.g. a day or two, an organization would note the vulnerability as part of its ongoing threat awareness programme. The organization would then usually conduct a preliminary risk assessment to determine if the vulnerability applies to its environment.

We distinguish between the time when exploit code is available and the time that malware might appear. Code exploiting some features of vulnerability is usually published earlier, often very close to disclosure time, sometimes as a proof that the vulnerability exists. It takes longer to develop exploit code with the potential to cause substantial damage. However, the sequence of the exploit, disclosure, and malware time is not fixed. Both, the exploit and the malware time can be before, at, or after the disclosure time. Similar reasoning applies to the time taken to develop patches; release of a patch could coincide with disclosure time or it may take as much as 30 days or more to develop.

The time of patch availability is the earliest date that the vendor of the software releases a patch providing protection against the exploitation of the vulnerability. We assume that the organization learns of the patch either the same day or with a delay that is negligible. The availability of other security mitigations such as signatures for intrusion prevention systems or anti-virus tools we take as being a separate matter from patch release. In our time line we assume that such mitigations are usually available earlier than patch. That being said, we also recognise that these mitigations are necessarily temporary stopgap measures and might not necessarily cover all disclosed vulnerabilities.

Once the organization receives a patch, the internal patch management process starts and continues until the patch is adequately deployed across the production environment. The organizations often regard that the exposure window closes when the proportion of systems patched is somewhere around 95%, as there will always be some number of systems that are either offline or unmanaged. This does not mean that risk goes to zero, since even one unpatched machine could present substantial exploitation risk, depending how it is positioned in the environment. However, to simplify our assumptions in the model, we regard the time when a patch is adequately deployed is when the window of exposure is closed: T_5-T_0. If an organizations uses other mitigations such as signatures we also regard the time that these are adequately deployed as the time that exposure is closed: T_2-T_0.

4.2. The rate parameters of external events

The analysis of the vulnerability timeline also highlights the events in the external environment that directly affect decisions the organizations takes on when to deploy security mitigations. The availability of an exploit poses a security threat to the organizations; sometimes the organization has to decide on an appropriate action before the patch is available. When the patch is released, the organization can start its patch management process, finally reducing the threat risk when the patch is deployed. The resulting exposure risk depends strongly on the timing and dynamics of those two external events.

[1] The sequencing of events in this timeline is not fixed; the aim is to illustrate the various stages in the vulnerability life cycle.

In our model we need to describe such events stochastically, by specifying the probability distribution functions that these events occur with. Based on the vulnerability data analysis done by Frei, et al [Frei 06] and data published by Symantec [Symantec 08], we chose to generalise on the following distributions:

- For *exploit availability event* after the vulnerability disclosure (time interval T_1-T_0) we assume that exploits appear at a constant rate of 5 days after disclosure , and so we use an exponential distribution [Grimmett 01] with the following probability density function]:

$$f_{exploit\_delay}\,(t_{exploit}) = 5e^{-5t_{exploit}}.$$

- For *patch availability event* (time interval T_3-T_0) similarly we assume that patches are released by vendors at a constant rate of 23 days after disclosure, and so an exponential distribution with λ=23 is used.

$$f_{patch\_delay}\,(t_{patch}) = 23e^{-23t_{patch}}.$$

- For *malware arrival event*[2] (time interval T_4-T_0) we use an exponential distribution as well, but we will vary the rate with λ=10 and λ=25.

$$f_{malware\_delay}\,(t_{malware}) = 25e^{-25t_{malware}}.$$

We acknowledge that our assumptions might not apply across the whole set of disclosed vulnerabilities, and we believe that for highly critical vulnerabilities or for different software vendors, these rates might be different. However, for the time being, we have not found enough data to contradict the rates we choose to use above.

Since we aim to simulate how the internal security processes in an organization react to the various vulnerabilities, we also need to take into account that not all vulnerabilities might be exploited and some might never have patches released. We decided to use the latest Symantec Threat Report [Symantec 08] to select the appropriate proportions of vulnerabilities[3]:

- 72% of vulnerabilities have exploits.
- 3% of vulnerabilities have no patches released.

Each event is captured as a separate entity in our model. We define certain sequencing between the events, so that a patch is available only on or after the vulnerability is disclosed, and, for simplification, we also define that vulnerability exploit and or malware arrive after a vulnerability has been disclosed as that is when the most of the internal mitigation processes also starts. This sequencing of events is also shown in figure 3. The Demos2K code of these three external events can be found in Appendix A.

4.3. Internal processes for threat assessment and mitigation

As we mentioned before, in implementing the patch management processes the organizations use guidelines and best practices as suggested by standard bodies, but the exact details usually differ based on internal organizational requirements. By working together with the security operations and threat awareness teams, we have tried to identify the main decision points and patch deployment policies that we believe are credibly representative across many organizations. Figure 3 shows the activity based diagram of the typical vulnerability assessment and patch management process.

[2] We should note that, since we aim to explore the performance of patching processes in our model, we only consider vulnerabilities and exploits where patching is the primary mitigation. For example, we don't take into account malware that exploits other types of vulnerabilities such as configuration errors.

[3] Again these assumptions are made across a generic set of vulnerabilities, and may differ depending on the type of the vulnerability.

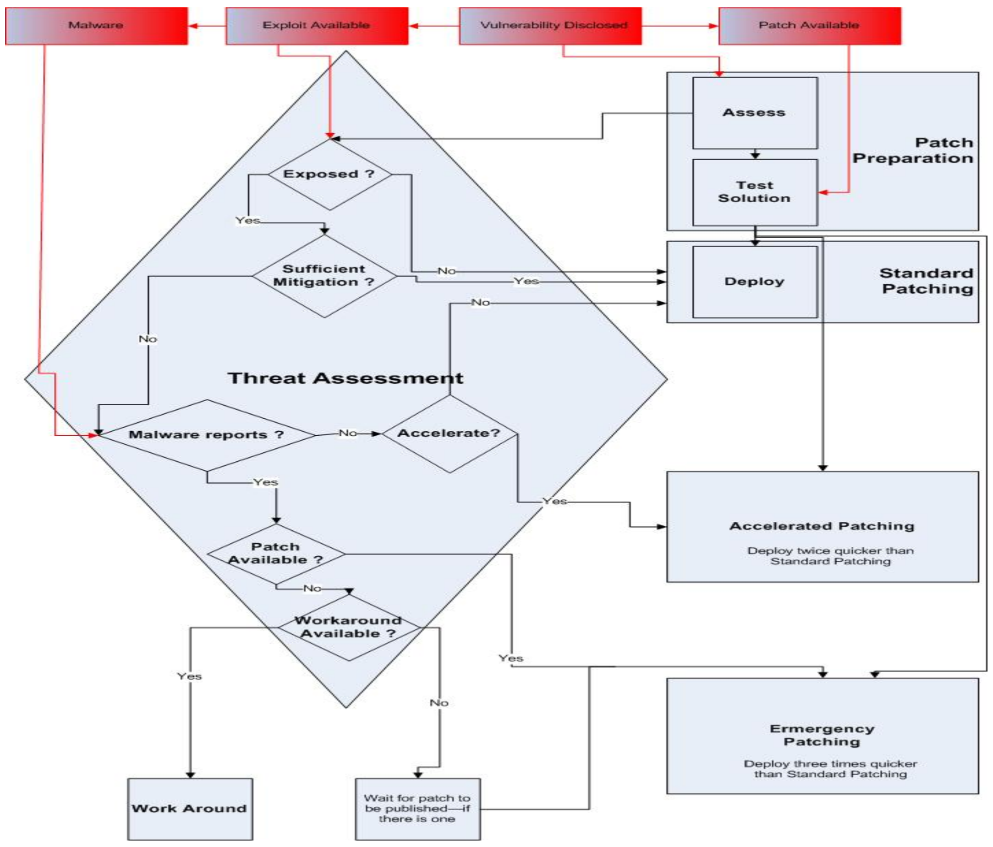

Figure 3. Activity diagram of a typical vulnerability and patch management process.

The left hand side of Figure 3 shows the various decision points in the threat assessment process that is usually triggered by the public disclosure of the vulnerability. On the right hand side of this diagram are the three types of patch deployment policies that an organization might decide to apply depending on the criticality of the vulnerability determined during the threat assessment. Once a patch is published by the software vendor an organization usually does its own patch preparation that consists of assessing and testing the patch within its own environment. This usually takes around 15 to 30 days, and includes an assessment of potential patch failure and might include waiting time for a second round of patch release from the vendor. Sometimes, the patch preparation might be expedited together with the patch deployment. However, due to the high risk of failure when deploying patches that are not properly tested, the security team might be reluctant to expedite the patch testing, and so for the purpose of our model, we decided to assume that patch preparation always takes the same time independent of the decision for patch deployment.

The three types of patch deployment policies differ only in the time it takes for a patch to be deployed across all vulnerable systems in the organization. Depending on the size of an organization and on the prevalence of the vulnerable software the deployment stage up to the time that most systems are patched can take up to 180 days, as the security operations teams have to negotiate with the business for the specific periods of IT systems downtime. Usually, the criticality and likelihood of vulnerability as determined during threat assessment is used in deciding the justifiable level of disruption to the business from deploying a patch compared to exposure risk.

For its threat assessment process an organization might use screening algorithms to suggest which vulnerabilities require discussion. Often it is based on the criticality score that a vulnerability has, applied according to the Common Vulnerability Scoring System (CVSS) version 2 standard [CVSS], but the security team might apply its own algorithm on top of that. The next step is to determine if this vulnerability applies to the organization's environment; if the vulnerable software is actually used in the IT environment. Once the applicability is determined, the threat assessment team reassesses the criticality of vulnerability and identifies if any mitigations exist before patch can be applied. These would include any prior mitigations such as when a newly disclosed vulnerability might have been covered by previous patches, signature-based mitigations such as antivirus, or any other type of mitigation employed before patching.

We observed that a primary trigger for deciding on quicker patch deployment are the reports of public exploits of the vulnerability, especially exploits with a large attack surface or that might result in quick spreading. The organizations might be advised on such reports by security software vendors, or have its own intelligence through tracking hacker community mailing lists. In rare cases, the threat assessment team might decide on emergency patching based on the criticality of vulnerability itself without there being public exploits yet. Based on the impact of a potential exploit and the criticality of the vulnerability, either accelerated or emergency patching policy is then applied instead of a standard, business as usual, patching. In cases, when at the time of public exploit, the vendor has not released a patch the security team might design a workaround, which could be firewall rule changes, shutting down of certain systems, or other measures usually resulting in more drastic disruptions.

4.3.1. Internal Parameters used for Simulations

For our simulations we have captured the activity diagram as six separate processes: vulnerability assessment, threat assessment, mitigation assessment and/or deployment, patch preparation, patch deployment, and workaround deployment. The Demos2k code for each of these processes can be found in Appendix B. Here we will specify the internal parameters that determine the sequencing of these processes and their execution.

The first aim with the model is to estimate how long the exposure window is across a large set of vulnerabilities with the above internal processes applied in an organization with around 100,000 systems requiring patching at the same time, and based on the probability distributions of external environment events as defined in section 4.2. In the first iteration of the model we decided to express the time it takes for each of the five processes to finish as a uniform distribution of time delay in terms of days, which will be represented here as *U(days_min, days_max)* .

- V*ulnerability assessment* process is triggered by a vulnerability disclosure event. In our representation this process is characterized by delay, which we selected within minimum and maximum parameters of 0.5 and 1.5 days:

$$f_{v\_assessment\_delay}\,(t_{assess}) = U\,(0.5, 1.5).$$

- *Threat assessment* happens after the vulnerability assessment and is triggered by an exploit availability event. This process is represented as a one day delay. It results in deciding that: (i) the vulnerability does not present high risk to the organization and can be mitigated by standard patching, (ii) existing mitigations cover the threat (standard patching would still be done), (iii) declaring either an accelerated patching or an emergency patching in case of reports of worm-type exploit/malware. By examining the past 2-3 years data of the of threat assessments and of the vulnerability CVSS scores as published in National Vulnerability Database [NVD] we have selected the following proportions for the different decisions that would be taken for patching the vulnerability:
    - 93% of vulnerabilities would be handled with standard patching,
    - 7 % of vulnerabilities would go through threat assessment; out of these:
        - 10% of vulnerabilities would require accelerated patching,
        - for 40% of vulnerabilities an emergency patching would be needed,
        - 5% of vulnerabilities require workarounds,
        - 45% of vulnerabilities could be covered with signature-based mitigations.
- *Patch preparation* begins whenever the vulnerability is disclosed and a patch is available. This process sometimes starts concurrently with threat assessment. It is represented as a time delay ranging from 10 to 20 days:

$$f_{p\_testing\_delay}\,(t_{esting}) = U\,(10, 20).$$

- *Patch deployment* has three timelines that are selected depending on the result from the threat assessment stage. We decided to simulate the environment of large organizations that have in excess of 100,000 systems and selected a fairly large range for how long *normal patching* across these numbers of machines would take: minimum 68 days and maximum 204 days[4]. The *accelerated* and *emergency patching* would be correspondingly twice and four times faster:

  $$f_{patching\_delay}\,(t_{patching}) = U\,(68/rate,\ 204/rate),$$

  where rate = 1, 2, or 4.

  Although these time periods for patch deployment seem quite long, they are actually realistic for organizations with a large number of systems, especially if these are servers running business critical applications. The time delays in such cases are usually due to restrictions set by business on systems downtime periods rather than resource constraints. For critical vulnerabilities, due to increased likelihood of exposure, the immediate patching of vulnerabilities would be raised as a priority, and so businesses would allocate additional periods for system downtime.
- *Signature-based mitigation deployment* might be triggered after threat assessment has determined that such mitigations work for the type of disclosed vulnerability. We assume that such signature based mechanisms would be provided by security software vendors, and the security team in an organization would not need to do their own preparation and testing. The time delay would then include only the time it takes for a vendor to create signatures and for the systems in an organization to get the updates. We selected this delay as being between 3 and 5 days:

  $$f_{signature\_delay}\,(t_{signature}) = U\,(3,\ 5).$$
- *The workaround deployment process* starts in those cases where a patch is not available, but the threat assessment team still declares emergency remediation – this process takes around 2-3 days to complete.

We believe that these internal process parameters chosen together with the decision process and mitigation mechanisms captured in the model are representative of a wide range of large organizations, but some of the specific details such as the time lines for patch deployment would be different. However, one of the strengths of this modelling approach is that parameters such as these can be easily adjusted and variations in processes or decision points can be easily captured in the model.

5. Analysis from experimental simulations

The experimental runs with the described model were performed through simulations of 100,000 vulnerabilities. As described in section 4.2 an individual vulnerability had 0.72 chance of having an exploit code and 0.97 chance of having a patch, with time delay probability distributions sampled for each event as defined in section 4.2. The time delay distribution for malware events was sampled with a rate parameter of 25 days, and based on the amount of past emergency patching due to malware appearing in the wild, the probability of malware was set at 0.84 for vulnerabilities where exploit code appeared beforehand.

Given these threat environment parameters, we first examined how long (in terms of days) an organization would be exposed when its security processes and chosen mitigations were performing as defined by the parameters described in section 4.3.1. The exposure window was examined across the full vulnerability set, and then zooming in on vulnerabilities, which threat assessment determined were of high risk for the organization.

In the next set of experiments we wanted to examine the changes in the length of the exposure window as we introduce more mitigation that would be signature or heuristic based, and so deployed soon after the vulnerability was disclosed. This could be the case if, for example, the security operations team decide to add intrusion prevention mechanisms. We recognize that the effectiveness of such mechanisms may vary, with signatures

[4] The time intervals selected are based on the assumption that a single patch would take around 1-5 minutes for each machine including rebooting the system and potential for failures, and are not based on the actual data from any specific organization.

available only for the proportion of vulnerabilities. To reflect that, we performed multiple simulations by varying the effectiveness of signature-based measures: the range starting from the signatures available for 3% of the vulnerability population and increasing up to 60%.

We also examined what the model predicts will happen to the time taken to achieve full patching when the patch deployment speed is increased 1.5 and 2 times. This could be representative of the cases where the number of vulnerable systems is smaller and so patch deployment is faster.

The experiments were run across vulnerability instances, and we sampled the various time points within the vulnerability timeline that was introduced in section 4.1. The data from simulations was then plotted as histograms with the y-axis representing the proportion of vulnerabilities in 100,000 instances, while the x-axis represents the time taken in days for the vulnerability risk to be reduced (to reach time point T_2 or T_5 within the vulnerability timeline).

5.1. Exposure window analysis

Figure 4 shows the histogram of the time it takes for an organization to minimize the risk of vulnerability from the time when it was first disclosed, where the model was executed with default parameters as described in section 4, and spans across all types of vulnerabilities. As is visible from this diagram, for around 47% of all simulated cases, it takes between 180 and 360 days for risk to be reduced, which mainly includes reaching time point T_5 in the vulnerability timeline. Within our default settings the standard patching was applied across 97% of vulnerabilities, with this type of patching taking 136 days on average, so the results reflect those settings. However, the results also highlight that with patching as the primary mitigation some vulnerabilities might remain unmitigated for almost a year in some large organizations. The patching policies are usually set, so that patch deployment teams have clear timelines. However, they also highlight that various delays such as waiting for patches to be released from the software vendor, internal patch assessment and testing, as well as business related delays may push the exposure window quite a few days beyond the average, leaving systems exposed for longer than might be expected.

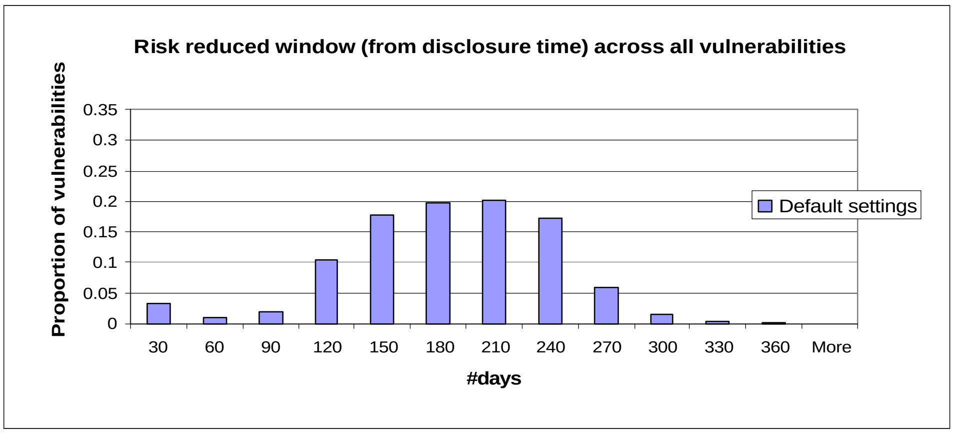

Figure 4. Risk reduced window across all types of vulnerabilities with default settings.

One might argue that this would be mostly vulnerabilities where the threat assessment has determined that the vulnerability does not present a high risk to the organization, and the expectation is that the 7% of vulnerabilities that went through threat assessment are mitigated faster. The spike at 0 to 30 days interval in the graph shows these faster mitigations of the critical vulnerabilities.

When we zoom in on these types of vulnerabilities as shown in purple in the histogram in figure 5, we can clearly see the difference in the time that it takes to reduce the risk. However, we should remember that we assumed in our model that for these types of vulnerabilities, security operations can deploy signature-based mitigation in around 45% of instances (or 3% of the total population). When those mitigations are removed, as represented by the blue histogram, the average exposure window is much longer.

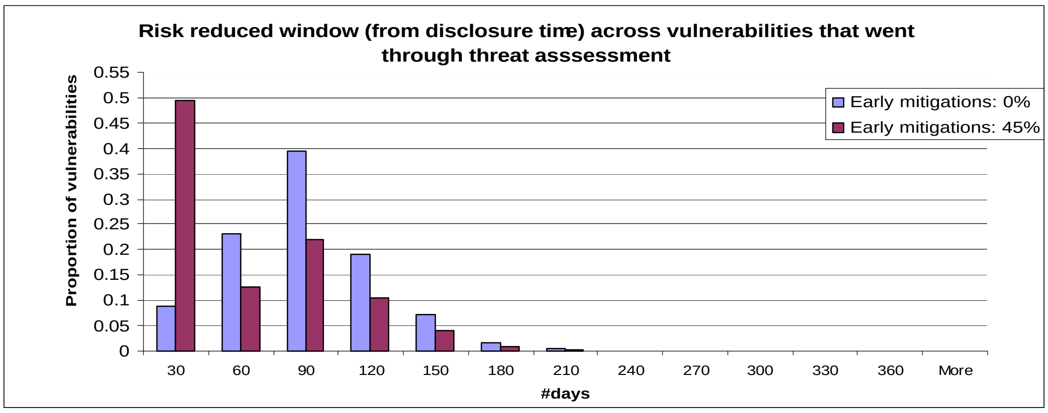

Figure 5. Risk reduced window of vulnerabilities that went through threat assessment.

Even when signature-based mitigations are effective for 45% of the critical vulnerability population, there is still a long tail that includes vulnerabilities taking more than 120 days. Since the chance of there being malware for these types of vulnerabilities is fairly high, the vulnerabilities in the tail might present a major risk. Assuming that malware arrives at a mean rate of 25 days, figure 6 shows that the proportion of vulnerabilities which would be mitigated within 10 days of malware reports, is nearly 52% (purple histogram), with the remaining 48% of vulnerability instances where malware could have major impact.

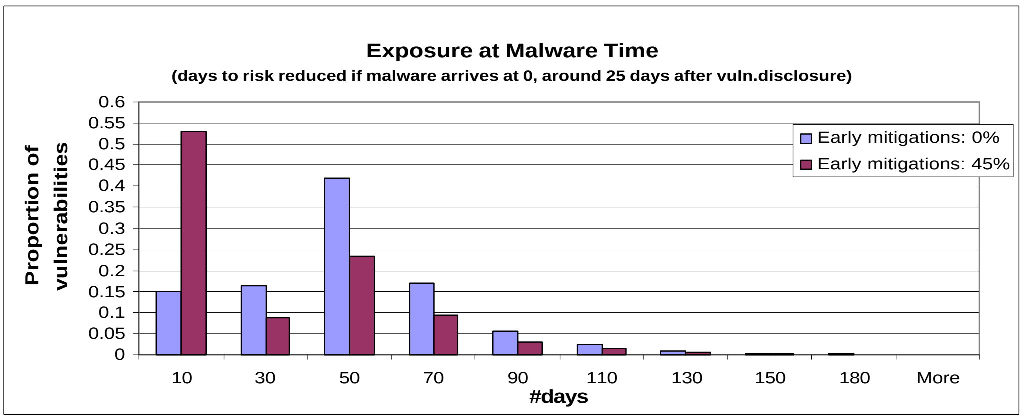

Figure 6. Exposure time when malware arrives.

This basic analysis highlights that in large organizations the vulnerabilities might remain unmitigated for much longer than the expected average, which in our simulations was selected as 136 days, and an intelligent and early threat assessment is crucial at reducing exposure for highly critical vulnerabilities.

5.3. Analysis of trade-offs between different types of mitigations

Through the second set of experimental runs we looked at two options that an organization might use to reduce the exposure window: (1) add more signature based or similar type of security measures that can be put into effect earlier; (2) expedite patch deployment processes.

5.3.1. Signature based mitigations

Figure 7 shows the results from simulations when the early, pre-patching mitigations were increased to cover 22% and then 45% of all vulnerability population compared to the previous results when these types of mitigations where at 3%. We can see that a much large proportion of vulnerabilities is being mitigated earlier, but since the patching timeline stays the same, there is still a large proportion that is taking up to 360 days to mitigate.

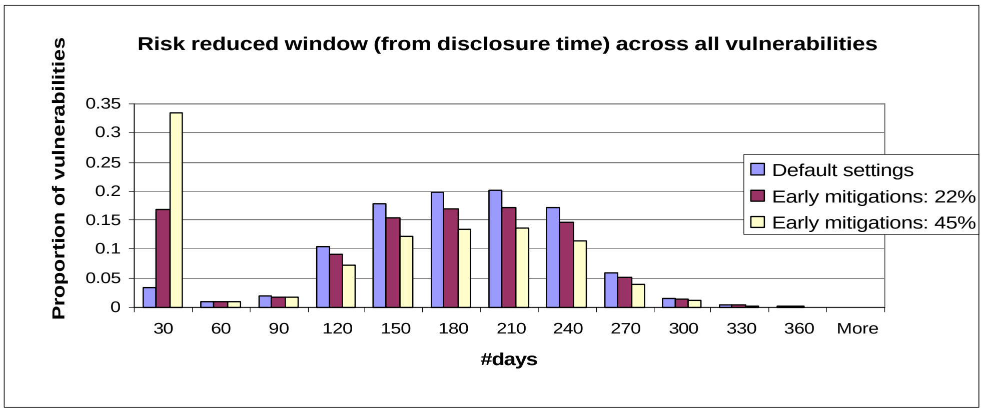

Figure 7. Risk reduced window across all types of vulnerabilities with varying coverage of signature-based mitigations.

For highly critical vulnerabilities the consequence of the increasing effectiveness of pre-patching mitigations from 45% to 60% is somewhat more distinct as seen in figure 8.

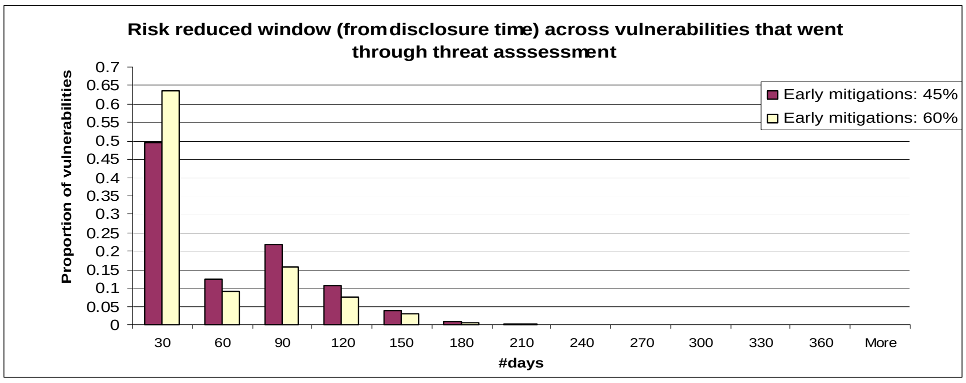

Figure 8. Risk reduced window across vulnerabilities that went through threat assessment, with varying coverage of signature-based mitigations.

This analyses highlights that, depending on how effective signature based mitigation measures are, they can be extremely useful at reducing the exposure window for organizations where patch deployment is slow and constrained. A lot depends, however, on the effectiveness of these extra mechanisms compared to their cost of deployment. None of the current mechanisms are 100% effective, and so organizations would still need to continue with their patching processes.

5.3.2. Quicker patch deployment

Next we examined the difference when patch deployment was done 1.5 and 2 times faster across standard, accelerated, and emergency patching. This could be representative not only of the case when the large organization deploys patches faster, but also of organization with smaller number of systems that takes less time to patch. The results in figure 9 shows a proportional increase in the number of vulnerabilities mitigated earlier, as the patching speed increases.

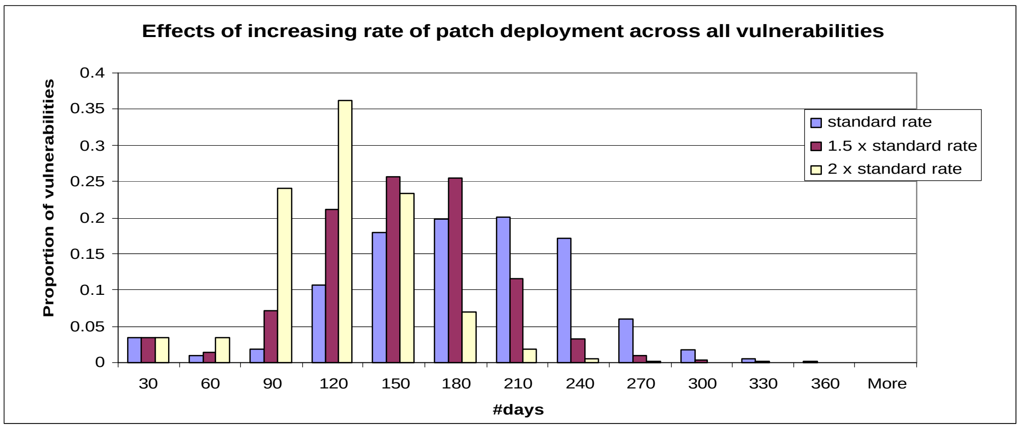

Figure 9. Risk reduced window across all vulnerabilities with increased rate of patch deployment.

As can be seen from the results in this diagram, by increasing the patch deployment speed to twice the default, the tail that we saw earlier with the default patch speed, is no longer present, with most vulnerabilities patched within 3 to 4 months after the disclosure. This observation could be used for deciding on a change in policy—reducing patch deployment deadlines—or alternatively to accepting that there will always be a subset of cases that will take a long time to reach the target level of protection, and so making sure that robust emergency processes are in place that enable the organization to respond very quickly if a major incident occurs.

6. Discussion

While we believe the results from the simulations based on the model described here provide interesting insights into how long it takes to close the vulnerability exposure window and the different impact that various mitigation measures have, the overall modelling approach can equally well be applied to other areas of security, to examine other key security processes or general IT operations processes as they affect the organization's risk exposure.

The modelling approach described in this paper is especially good for exploring possibilities and answering what-if questions, which can be particularly helpful in security decision-making that currently lacks a rich set of tools. In the example we have studied, we can use the outputs from the model to compare potentially competing security investments when there is a limited budget, for example, installing extra client protection such as HIDS/HIPS versus investing in reducing patch deployment times. In large organizations, the security operations team does not always get to control patching, but has to negotiate policy and deadlines with other departments, and through our work we aim to provide them with the tools they can use to show, in concrete terms, the improvements that will result from well-chosen security policies and investments in security technologies.

In the future we would like to take this work into at least two further directions.

Cost trade-offs

Cost trade-offs of the different mitigation measures, such as the cost of implementing signature-based measures compared to the faster patch deployment have not been covered in this paper. However, we plan to extend the model so that it can be used to weigh a smaller exposure window and decreased emergency patching cost against the operational overhead and business risk. The most important part when calculating trade-offs is the effectiveness of the security measures at mitigating the various threats, and the analysis in this paper contributes to that. It would be interesting to complete the picture by incorporating cost information into the model for all the different processes, so that as well as exploring effectiveness we could also investigate expected overall and per-incident costs under various conditions.

External threat environment

The second direction for future work is in improving our model of the external threat environment. For our simulations we used parameters that apply across a generic set of vulnerabilities. These may not be representative of the software portfolio of a typical large commercial organization, and of the subset of potentially critical vulnerabilities in particular, in terms of rates and likelihoods of exploit code, malware and patch releases. Causal influences should also be taken into account, e.g., when prototype exploit code is published for a vulnerability, the software vendor is encouraged to hurry up the patch release; also when a vendor publishes a major patch release this may be followed by a flurry of new exploit development.

Furthermore, security investment is about protecting against future threats. The threat environment is continually evolving, and it is important to have security policies that are robust to different environmental conditions (past performance is no guarantee of future performance.) So one way we intend to apply this technique is to evaluate the effectiveness of an organization's security policies and processes in the face of a variety of possible changes to the threat environment, by varying parameters in the model to correspond to those changes.

7. Conclusion

In this paper we have presented a novel approach to examining the effectiveness of security operations processes and protection mechanisms, which consists of designing a model of these processes and running stochastic simulations, based on external environment events. We focussed on examining the "risk exposure window" as a measure of the effectiveness of these processes, which we defined as the time from public vulnerability disclosure to when an organization believes the risk is mitigated. In designing the model we have examined the decision making process in the security operations teams across several large organizations together with the different mitigation and patching measures that might be selected. We also identified external threat environment events that influence the type of mitigations that are deployed and the time when they are deployed.

An important goal of this work is to enable the decision makers in IT security to predict the outcome of investment decisions or changes in policy in advance of putting them into effect. The results from experimental simulations presented in this paper show the impact of increasing effectiveness of early mitigations and of speeding up patch deployment at reducing the risk exposure window. This trade-off analysis can be directly used by the security operations teams in deciding whether mitigations such as host based intrusion detection would be sufficient in reducing the risk.

References

[Anderson 01] R. Anderson. Why information security is hard: An economic perspective. In *Proc. 17th Annual Computer Security Applications Conference*, 2001.

[Beattie 02] S Beattie, S Arnold, C Cowan, P Wagle, C Wright, A Shostack. Timing the Application of Security Patches for Optimal Uptime. In *Proc of LISA'02: 16th System Administration Conference*, 2002.

[Bir 79] G. Birtwistle. Demos – discrete event modelling on Simula. Macmillian, 1979.

[CVSS] Peter Mell, Sasha Romanosky, A Complete Guide to the Common Vulnerability Scoring System Version 2.0. http://www.first.org/cvss/cvss-guide.pdf, June 2007.

[Dem] Demos2k. http://www.demos2k.org/

[Frei 06] Stefan Frei, Martin May, Ulrich Fiedler, Bernhard Plattner. Large-Scale Vulnerability Analysis. In *Proc. of SIGCOMM'06 Workshops*, September 2006

[Gordon 02] Lawrence A. Gordon, Martin P. Loeb. The Economics of Information Security Investment. *ACM Transactions on Information and Systems Security*, Vol. 5, No. 4, November 2002

[Gordon 06] L.A. Gordon and M.P. Loeb. Managing Cybersecurity Resources: A Cost-Benefit Analysis. McGraw Hill, 2006.

[Grimmett 01] G Grimmett and D Stirzaker. Probability and Random Processes, 3rd ed., Oxford UP, 2001

[ISO 27] Code of Practice for Information Management, ISO/IEC 17799:2000, Int'l Organization of Standartization, 2000.

[McHugh 00] William A. Arbaugh, William L. Fithem, John McHugh. Windows of Vulnerability: A Case Study Analysis. *IEEE Computer*, 2000.

[Milner 83] R. Milner. Calculi for synchrony and asynchrony. Theoretical Computer Science, 25(3):267–310, 1983.

[Milner 89] R. Milner. Communication and Concurrency. Prentice-Hall, 1989.

[Monahan08] Brian Monahan, DXM - The Demos eXperiments Manager, HP Labs Technical Report, 2008.

[NIST 02] Peter Mell and Miles C. Tracy. Procedures for Handling Security Patches. National Institute of Standards and Technology, *NIST Special Publication 800-40*, August 2002.

[Pym 08] A Beautement, R Coles, Jonathan Griffin, C Ioannidis, B Monahan, D Pym, A Sasse, M Wonham. Modeling the Human and Technological Costs and Benefits of USB Memory Stick Security. *Workshop on the Economics of Information Security, WISE 2008*.

[Pym 07] Jonathan Griffin, Brian Monahan, David Pym, Mike Wonham, Mike Yearworth. Assessing the Value of Investments in Network Security Operations: A Systems Analytics Approach. *Workshop on the Economics of Information Security, WISE 2007*. http://www.hpl.hp.com/techreports/2007/HPL-2007-89.pdf

[Pym 06] D Pym, B Monahan. A Structural and Stochastic Modelling Philosophy for Systems Integrity. HP Labs Technical Report Series, HPL-2006-35, Feb 2006. http://library.hp.com/techpubs/2006/HPL-2006-35.pdf

[Schneier 02] B Schneier, Managed Security Monitoring: Closing the Window of Exposure. *Counterpane Internet Security*. http://www.counterpane.com/window.pdf

[Symantec 07] Symantec Global Internet Security Threat Report: Trends for July–December 07. Volume XII, April 2008. http://www.symantec.com/business/theme.jsp?themeid=threatreport

[Zhang 07] Guoying Zhang, Yong Tan, Debabrata Dey. Optimal Policies for Security Patch Management. Under review. http://faculty.washington.edu/ytan/UnderReview/Patch.pdf

Appendix A.

Demos2k of the external events.

```
class disclosureGenerator = {
    do maxInstances {
        sync (initialize);
        instance := instance + 1;
        disclosureTime := DEMOS_TIME;
        disclosed := 1;
        // kick off events in external environment
        entity (patch, patchPublicationEvent, 0);
        entity (exploit, exploitEventEtc, 0);
        // wait until the adventure's over
        req [(patchTargetReached == 1 || noPatch == 1) && exploitEtcTally == 0];
        hold (1000);  // big enough to let any hold's finish - sadly hold (100)
isn't enough in rare cases, and I'm not sure why
        sync (traceHistory);
        demos_sample_tick := demos_sample_tick + 1;
    }
    done := 1;}

class patchPublicationEvent = {
    // either publish a patch, or give up after a year
    try [testNoPatch == 0] then {
        hold (disclosureToPatchTime);
        patchPublishedTime := DEMOS_TIME;
        patchPublished := 1;
    }
    etry [] then {
        noPatch := 1;
    }}

class exploitEventEtc = {
    try [testExploit == 1] then {
        hold (disclosureToExploitTime);
        exploitTime := DEMOS_TIME;
        exploited := 1;
        entity (measureAfterExploit, measureAfterExploit, 0);
    try [testMalware == 1] then {
            hold (exploitToMalwareTime);
            malwareTime := DEMOS_TIME;
            malware := 1;
            entity (measureAfterMalware, measureAfterMalware, 0);
        }
        etry [] then {
        }
    }
    etry [] then {
    }
    hold (0);  // Give other entities a chance to get going before reducing
exploitEtcTally to 0
    exploitEtcTally := exploitEtcTally - 1;
}
```

Appendix B.

Demos2k code of internal security processes.

```
(*  Threat assessment decision processes *)
class ThreatAssessmentProcess = {
    local var p = 0;
    local var early = testEarlyErmergency;
    local var doErmergency = testEmergency;
    req [assessed == 1];  // Process can't start until assessment is completed
    try [exploited == 1] then { //threat assessment don't start until there's some
exploit code
        try [exposed == 1] then {
            try [testMitigation == 1] then {
                entity (mitigation, mitigationProcess, 0);  // No ermergency
            }
            etry [] then {
                  try [malware == 0 && testAccelerate == 1] then {
                  patchAcceleratedTime := DEMOS_TIME;
                    patchAccelerated := 1;
                    patchSpeed := patchAccelSpeed;
                }
                etry [] then {}
                try [early == 1 || malware == 1] then {
                    hold (assessmentDelay);
                    try [patchTargetReached == 0] then {
                        try [patchPublished == 1] then {
                            try [doEmergency == 1] then {
                                EmergencyTime := DEMOS_TIME;
                                Ermergency := 1;
                                patchSpeed := patchErmergency24x7;
                            }
                            etry [] then {}  // if we don't declare an emergency,
just leave patching to complete
                        }
                        etry [testWorkaround == 1] then {
                            entity (workaround, workaroundProcess, 0);
                        }
                        etry [] then {
                            noSolution := 1;  // :-)
                            // But if the patch gets published later, I guess we
may declare a Emergency then
                            try [noPatch == 0 && doEmergency == 1] then {
                                try [patchPublished == 1] then {  // blocking test:
wait for the patch to be published
                                    EmergencyTime := DEMOS_TIME;
                                    Emergency := 1;
                                    patchSpeed := patchEmergency24x7;
                                }
                            }
                            etry [] then {}
                        }
                    }
                    etry [] then {} //if patching complete, then nothing more to do
                }
                etry [exploitEtcTally == 0] then {// no Emergency
                }
            }
        }
        etry [] then {  // no Emergency, but there might still be a mitigation
            try [testMitigationNonExposed == 1] then {
                entity (mitigation, mitigationProcess, 0);
            }
            etry [] then {}
        }
    }
    etry [exploitEtcTally == 0] then //no Emergency
    }
}

class mitigationProcess = {
    hold (mitigationDelay);
    // Different from workarounds - mitigation is deployed even if the patch is
published in the meanwhile
    sync (recordRiskReduced);
    mitigationDeployedTime := DEMOS_TIME;
    mitigationDeployed := 1;
}

class workaroundProcess = {
    hold (workaroundDelay);
    // We used to assume that the workaround got aborted if the patch was published
while it was being implemented, but now we don't
    sync (recordRiskReduced);
    workaroundDeployedTime := DEMOS_TIME;
    workaroundDeployed := 1;
}

class patchTestProcess = {
    local var baseDelay = patchTestBase;  // Sample relative interval once and use
it in any calculations of how long patch testing takes
    try [assessed == 1 && patchPublished == 1] then {
        local var patchTestStart = DEMOS_TIME;
        local var normalDelay = patchTestNormalDelayMin + patchTestNormalDelayMax *
baseDelay;
        entity (doPatchTest, doPatchTest (#normalDelay), 0);
        try [GEM == 1] then {
            local var GEMDelay = patchTestEmergencyDelayMin +
patchTestEmergencyDelayMax * baseDelay;
            local var remainingDelay = EmergencyDelay * (1 - (DEMOS_TIME -
patchTestStart) / normalDelay);
            entity (doPatchTest, doPatchTest (#remainingDelay), 0);
        }
        etry [patchTested == 1] then {  // patch test completed before or without
an Emergency declaration
        }
    }
    etry [noPatch == 1] then {
        // nothing to do
    }
}
```

```
class doPatchTest (delay) = {
    hold (delay);
    try [patchTested != 1] then {  // If testing not already finished via some
other path
        patchTestedTime := DEMOS_TIME;
        patchTested := 1;
    }
    etry [] then {}
}


class patchDeploymentProcess = {
    local var variation = patchDeploymentVariation;
    local var startTime = 0;
    local var m = 0;
    local var c = 0;
    local var y = 0;
    local var patchSpeedValue = patchNormal;
    try [patchTested == 1] then {
        entity (progressCurve, progressCurve (nPointsNormal, timesNormal,
valuesNormal, #variation, #0, #patchSpeedValue), 0);
        while [startTime >= 0] {
            try [patchSpeed != patchSpeedValue] then {  // catch change in
patchSpeed (Emergency status)
                patchSpeedValue := patchSpeed;
                y := m * (DEMOS_TIME - startTime) + c;
                try [patchSpeedValue == patchAccelSpeed] then {
                    entity (progressCurve, progressCurve (nPointsAccelerated,
timesAccelerated, valuesAccelerated, #variation, #y, #patchSpeedValue), 0);
                }
                etry [patchSpeedValue == patchEmergency24x7] then {
                    entity (progressCurve, progressCurve (nPointsEmergency,
timesGEM, valuesGEM, #variation, #y, #patchSpeedValue), 0);
                }
                etry [] then {
                    trace ("Error: unexpected patchSpeed %v", patchSpeedValue);
                    errorUnexpectedPatchSpeed := 1;
                    demos_sample_tick := demos_sample_tick + 1;
                    close;
                }
            }
            etry [getSV (deployment, [], true)] then {
                y := m * (DEMOS_TIME - startTime) + c;
                putSV (deployment, [y]);
            }
            etry [getVB (progress, [m, c, startTime], true)] then {
            }
        }
        sync (recordRiskReduced);
        patchTargetReachedTime := DEMOS_TIME;
        patchTargetReached := 1;
    }
    etry [noPatch == 1] then {
        // nothing to do
    }
}
```