Assessing Risks of Policies to Patch Software Vulnerabilities

Jaziar Radianti (jaziar.radianti@hia.no)
Finn Olav Sveen (finn.o.sveen@hia.no)
Jose. J. Gonzalez (jose.j.gonzalez@hia.no)

Research Cell "Security and Quality in Organizations",
Faculty of Engineering and Science, Agder University College, Serviceboks 509
NO-4898 Grimstad, Norway

Abstract

The number of security vulnerabilities, breaches and digital disaster increases over time. One important source of weaknesses in computer networks is the ubiquitous flaws ('bugs') in the software, which are exploitable by malicious agents. Consequently, "patching" the software to correct known bugs is becoming more important, especially for network-based systems. However, decision makers often view this issue differently, due to the presumption that security measures are time consuming and an interruption to the primary business activities. In addition, it is considered too costly to invest in the prevention of something that might not happen. Patching often requires extensive testing and that computer networks be taken down. This work is a preliminary effort to develop a system dynamics model for showing the trade-offs and the risks of different patching policies.

Key Words: Patch, Risk Assessment, Information Security, System Dynamics, Vulnerability

I. Introduction

Corporations today face many security challenges because they have to rely more and more upon computer networks. The combination of the data available on the network and the difficulties in protecting it make the computer network systems vulnerable to a variety of attacks, threats and adversaries.[1] Attacks in the digital world might have the same goals and share many characteristics as those in the physical world. But there are some important differences: Attacks on digital assets are more widespread and it is harder to track, capture and convict the perpetrators (Schneier 2000). As a result of the growing number of attacks, downtime is up. Statistics show that the average downtime caused by a virus attack is around four to eight hours. Many companies often experience between one and three days of downtime, and the duration of such extreme events is rising (Hulme 2004).

Cyber attacks, including those with no apparent political motivation, are becoming more and more sophisticated. Whether motivated by financial gain or simply the challenge of breaking through defences, attackers have been gradually improving their methods for years. Security measures can no longer be considered an excessive response; they are a necessary effort to maintain critical information assets.

Although people are aware of the danger of attacks on the computer networks, many only consider the problem important after they personally experience the cost of damage caused by computer security incidents. Many staff and decision makers in an organization or company have different attitudes and perspectives toward the real threat to their network based systems,[2] often viewing them as a 'disturbance' to the primary work processes. Additionally, such measures are often neglected because the organization pays most attention to their primary activities (to produce something, to gain higher profit), and considers attacks on the security system[3] of secondary importance.

Schneier points out that the number of security vulnerabilities, breaches and digital disaster increases over time. It is important to learn how to manage and reduce the vulnerability, whether through protection, detection and reaction (Schneier 2000).

One important source of weaknesses in the computer network is the ubiquitous flaws ('bugs') in the software. Many of those bugs are "exploitable" by malicious agents. Studies at Carnegie Mellon University (CMU) find that there are from five to fifteen 'bugs' per one thousand lines of code in released software (Schneier 2000); another study reports up to one

[1] Recent security trends indicate that malicious code, vulnerabilities, spam and spyware as well as identity theft are increasing, while at the same time the time to exploit vulnerability is decreasing (Qualys 2006).
[2] Easttom (2006) identified three attitudes toward computer security: The first group assumes there is no real threat; there is little real danger to computer systems and they only take minimal security precautions to ensure the safety of their systems—they have a *reactive* approach to security. The second attitude is one that tends to overestimate the dangers; people in this group believe that talented hackers exist in great numbers and all pose imminent threats to the computer system. The third and more balanced view, and a better way to assess the threat level, is to weigh the attractiveness of the system to potential intruders against the security measure in place.
[3] *Malware* such as virus attacks, Trojan horses and spyware; *intrusions*−effort to gain unauthorized access to the computer system, and *Denial of Service* attacks—to flood the targeted system with so many false connection requests that the system can't respond to legitimate requests.

bug in every ten lines of code (Lynn 2002). [4] The most common method of correcting a software defect is by installing an appropriate software patch. This issue is becoming more important since the time between the discovery of an operating system or application's vulnerability and the emergence of an exploit is being reduced dramatically over the last years and is still getting shorter, sometimes to only a matter of hours (Voldal 2003; Arbaugh W.A., Fithen, and Hugh. 2000). Additional defence solutions such as firewalls, antivirus software and intrusion detection systems are necessary layers of security, but they are incapable of proactively detecting network vulnerabilities and cannot reliably prevent attacks. Today's attacks bypass these layers of protection and directly target network weaknesses. This situation not only make the patching issues more important, but also require that critical patch releases become more frequent. Many organizations are struggling to keep current with the constant release of new patches and updates.

This is only the beginning, however, since there are indications of a growing number of vulnerabilities in software and operating systems that are only known to 'black-hat hackers' and for which there is no patch yet available. Such vulnerabilities might be attacked, with potentially devastating effects, using new kinds of viruses and worms. The problem is even more complex, since installing a patch may not be a simple fix in corporate environments, owing to the broad range of applications the patch must be compatible with.

From the computer security point of view, shorter time to exploit new vulnerabilities means an increased threat to the networked system. The characteristics of threats to computer systems have entered the so-called "third-generation" – automated attacks leveraging known and unknown vulnerabilities, collaboration of social engineering[5] and automated attacks, multiple attack vectors and active payloads.

The shift in the nature of attacks is another reason why patching is becoming more important for security. CMU estimated that 95 percent of all network intrusions could be avoided by keeping the computer systems up to date with patches.[6] But applying patches is often seen as a hard and time-consuming task, requiring computer networks to be taken down. Although security is critical, patching is sometimes skipped to keep the business running.

It is recognized that security measures might introduce some risks, but to leave the computer network without patching is the same as leaving your front door and back door unlocked and all your windows open.

II. Literature Review

Securing a system always means considering vulnerabilities, threats, countermeasures and acceptable risk. There has been much written on vulnerability issues in the literature, especially in the fields related to safety and computer security. Various definitions have been proposed to discover vulnerabilities, how they occur, how they develop and how they are managed.

[4] Lynn quotes a study by Humphrey mentioning three defects in the software production: 1) an omitted line of code, 2) two characters interchanged in a name and 3) an incorrect initialization.
[5] A technique for breaching a system's security by exploiting human nature rather than technology—or a non-technical way of intruding on a system.
[6] See http://www.cert.org/homeusers/apply_patches.html. Schneier even mentioned a more optimistic estimate that over 99 percent of all Internet attacks could be prevented if the system administrators would use the most current versions of their system software (see Schneier, 2000, p. 210).

Some literatures highlight the phases of window exposure where people can exploit the vulnerability (Arbaugh W.A., Fithen, and Hugh. 2000; Schneier 2000). A number of studies reveal interest in developing an approach to classify security vulnerability[7] (Seacord and Householder 2005) or creating taxonomy related to computer program security flaws[8] (Landwehr et al. 1994).

Other studies concentrate more on vulnerability management strategy (Bentley 2005; Martin 2001) and vulnerability risk mitigation (Lynn 2002). One study concerning the best time for patching looked at the cost of breach recovery vs. the cost of bad patch recovery (Beattie et al. 2002). The study concluded that the best time for patching is between 10 and 30 days after the patch's release. Wiik et al. (2004) modelled the life cycle of software vulnerabilities using System Dynamics.

Vulnerability is a state of *a missing or ineffectively administered safeguard or control* that allows a threat to occur with a greater impact or frequency (Peltier, 2005, p.14), a *flaw or defect* in a technology or its deployment that produces an exploitable weakness in a system (Arbaugh, Fithen and Hugh, 2000, p. 54) and makes it susceptible to attack (Wahlström 2005). In essence, vulnerability is the situation that someone or something may break into the system or misuse it, or lead to intrusions into the computer system, thus making it a possible security threat.

However, for the purpose of this study, a more specific understanding of software vulnerability is needed. Seacord and Householder (2005) have specifically defined some terms: A security flaw[9] is a defect in a software applications or component that when combined with the necessary conditions, can lead to software vulnerability. A vulnerability is a set of conditions that allows violation of an explicit or implicit security policy. These vulnerabilities may result from mistakes when writing software, whether a math error, incomplete logic or incorrect use of a function or command (Martin 2001). Objects and relationships for software vulnerability are shown in the following figure:

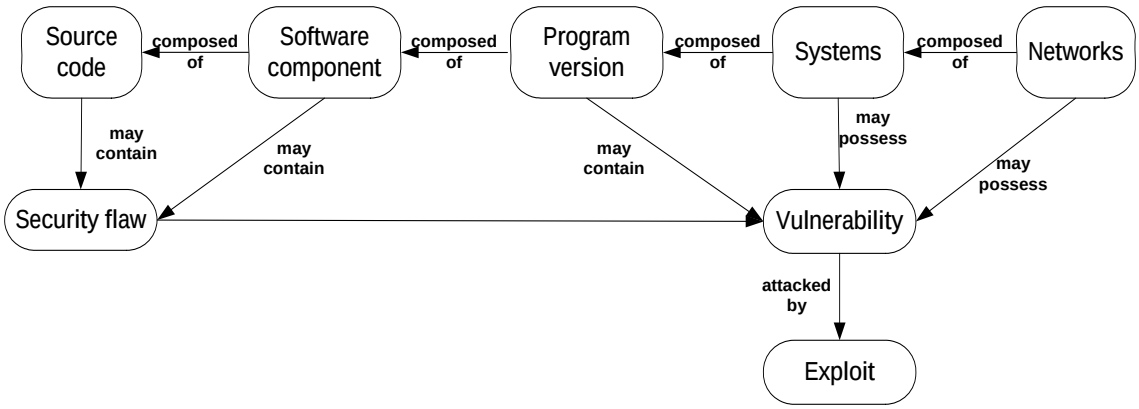

Figure. 1
Objects and Relationships of Vulnerability
Sources: modified from Seacord & Householder, 2005

[7] Classification scheme uses "attribute-value" pairs.
[8] They divide security flaw taxonomy *by genesis* (intentional and inadvertent), *by time of introduction* (during development, during maintenance or during operation) and *by location* (in software or in hardware).
[9] Landwehr et.al. 1994 simply say flaw in a program is a "bug". And they use terminology "flaw" instead of other tems such as *error* (human action that produces an incorrect result), *fault* (an incorrect step, process, or data definition in a computer program, and *failure* (the inability of a system or component to perform its required functions within a specified performance requirements).

The quickest way to counter a software vulnerability is patching (as opposed to installing a newer version of the software in question with the vulnerability removed). A patch is a piece of code added to the software in order to fix a vulnerability or a bug, especially as a temporary correction between two releases. Patches may also be used to enable new hardware, supply new functionality or deliver utilities (Packard 2006).

Vulnerability is closely related to the concept of risk. Risk has been defined as something that creates hazard (Peltier, 2005, p.41), a source of danger, a possibility of incurring loss (Wahlström 2005). Therefore, some methods of risk analysis must be applied, through vulnerability and risk assessment and its role to identify and assess factors that may jeopardize the success of a company's goal.

In the case of patching policies, Voldal states that an organization should look at the potential threat along with the vulnerability to determine the risk of having an unpatched system. The risk assessment of whether to apply a patch or not should include the risk of not patching the reported vulnerability, the risk of extended downtime, impaired functionality and lost data (Voldal 2003).

Voldal divided the environments of the servers in an enterprise into three parts:

- Mission critical—an environment in which even one hour of downtime will have a significant impact on the business service and whose availability is required at almost any price.
- Business critical—an environment in which business services require continuous availability, but breaks in service for short periods of time are not catastrophic.
- Business operational—an environment in which service breaks are not catastrophic.

Factors to consider when determining the mission critical status of the system include: the system's role in the enterprise mission; impact of system down time on the mission; and time and effort required for disaster recovery.

Risk cannot be completely eliminated through computer security policy and measures. If the systems are functioning well today, it does not mean that they will continue to do so indefinitely in the future. Conditions and requirements may change over time. There are several types of risks involved with software. Some risks are inherent in the product, others are related to the way in which the software is used. Some of the risks involved with patches and the patching process are listed below. In many cases, risks could be an option between doing nothing (not patching), and doing something (patching). Technically, risks related to patching are:

- ***Unplanned Downtime***
  The biggest risk to systems from insufficient software quality is downtime. If a critical system or application is unavailable, the negative consequences can include substantial loss of revenue or even loss of life. For mission critical systems and applications, all efforts should be focused on maximization of availability.
- ***Poor Performance***
  Even if a system is working, it may run the risk of delivering unacceptable performance. If the system response is slow, end-users will only wait a limited amount of time before giving up. Some may try again later. Others will find an alternative. In

the case of e-commerce, this can mean the difference between a company's survival and its extinction.

- ***Impaired Function***
  Even though a system may be available and performing its primary mission well, it may still have problems. These can range from minor disturbance to severe usability issues.

To recognize a real threat, people most often perform *risk assessment*, a computation of risk to determine what threats exist to the project mission, and a *vulnerability assessment*, a systematic examination of a critical infrastructure, the interconnected systems on which it relies, and its information or product, to determine the adequacy of security measures that identify security deficiencies (Peltier 2005).

III. The Case: Patching or not patching

Security patches and virus definition updates are not a major priority for most companies, until they experience a virus attack of significant severity. When a virus or other methods of attack exploit known vulnerabilities, many people wonder why the companies had not patched their systems in the first place.

Many companies worry about the amount of money they have to spend on patching. Installing the patch means paying money to prevent something that might only potentially happen. Companies do not see such issues as critical until the risk actually materializes (Hurley 2006). In addition, applying patches introduces changes into the operating environment. When the change is necessary to prevent or correct a problem or to enable needed functionality, patches become extremely important tools. However, patches also have the potential to cause incompatibility problems.

Patching in complex organizations is difficult because most companies have policies in place requiring quality assurance and testing of any patches. Accordingly, installing a patch requires an ample amount of work. Moreover, big organizations need different patching solutions for systems they have acquired from different vendors, which again add to the complexity of the job. IT organizations must develop a process to ensure the availability of resources, install required security patches and not disrupt existing systems in the process.

For our modelling purposes, consider a company that tries to enhance their security policy by applying patches to all their computer systems.[10]

The company's primary defence of their networks is a perimeter that separates their networks from the outside world and controls all traffic through the central firewall—a barrier between their network and the outside world, a tool for filtering traffic entering the network. However the company is conscious that the perimeter approach[11] not only overlooks the possibility of internal threats, but also provides an attractive target for any intruder or malicious code (viruses, worms) that can breach the external perimeter.

[10] This work is an anonymous and simplified version of a real-life case of an enterprise belonging to a critical infrastructure. We have left out any reference that might identify the client.

[11] In the network security paradigms there are two approaches that influence the security decision. In the perimeter security approach, the bulk of security efforts are focused on the perimeter of the network. This focus might include firewalls, proxy servers, and password policies. Another one is the layered security approach, one in which not only is the perimeter secured, but also individual systems within the network. All servers, workstations, routers and hubs within the network are secure (Easttom 2006).

It presumes that segregating infrastructure into domains could increase the resilience toward an attack by confining an attack or infection to a section of the network, enabling business to continue in other domains. Worm attacks generate so much spurious traffic when they replicate that the network becomes unusable. Segmenting the network not only limits the spread of the attack, but also confines the denial-of-service- effect.

The company feels certain that the implementation of this concept will improve security and resilience of their infrastructures despite the lack of or inadequate patching, worm attacks, hacking, intrusion, operational faults and unauthorised access.

However, the company is aware of the issue of vulnerabilities and that a virus attack is still possible, even though it has segregated its infrastructure. For that reason a patching policy is developed which attempts to balance benefits and inconveniencies of patching vulnerabilities.

The purpose of this paper is to provide a model using system dynamics that can help decision makers in organizations to increase the knowledge and understand the trade-offs of patching policies. Policy analysis can help decision makers assess the risks for various patching strategies.

IV. The Model Structure

Based on the previous background and case, we built a generic model to assess the risks of different policies to patch software vulnerabilities. The model describes a generic case for an anonymous organization.

The overall model is shown in Figure 3. The following assumptions have been made:

1. We assume that only publicly known vulnerabilities affect the company. That is, within the scope of this paper we disregard the issue of vulnerabilities known only by a handful of hackers and not by vendors and security companies.
2. Patches are released as new vulnerabilities are discovered. We assume that this happens very quickly so that we do not include in the model the discovery of vulnerabilities.
3. The model does not explicitly represent the attackers. They are instead implicit in the form of the probability that unpatched vulnerabilities be exploited.
4. Patches are applied for mission critical systems (*see* the definition in literature review section)

The basic model is as follows:

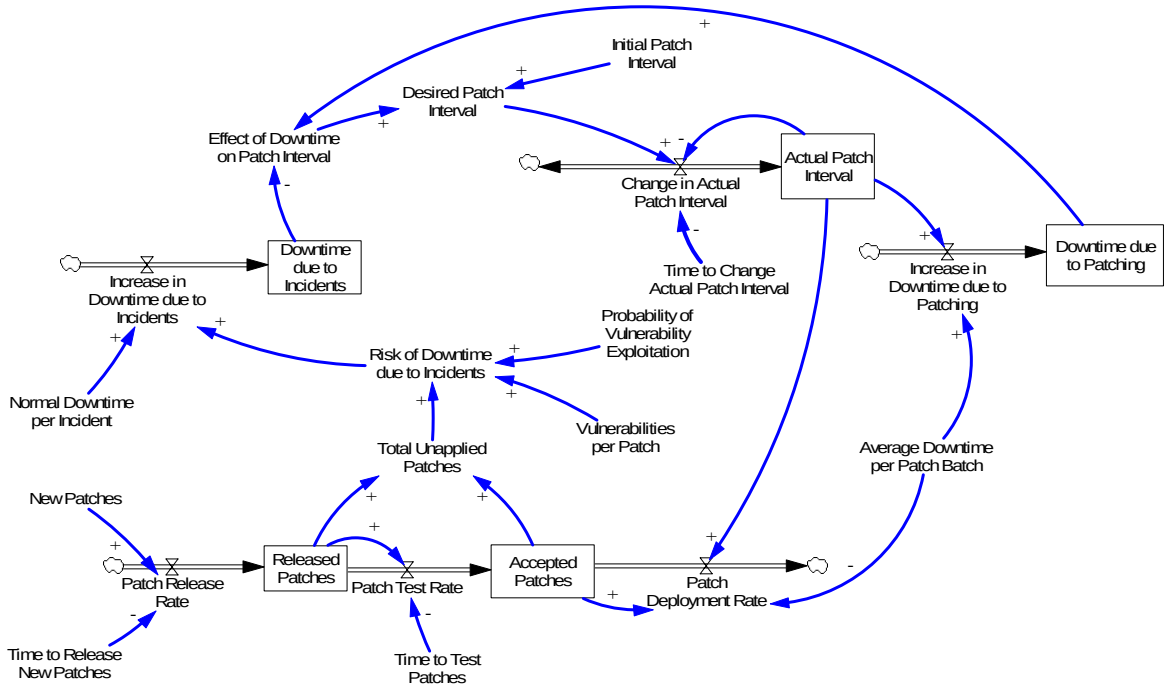

Figure 2
Main Model

A patch is assumed to be released whenever a new vulnerability is found. Therefore, the flow *Patch Release Rate* also represents new vulnerabilities found. One patch is made to fix one unit of vulnerability. Thus far, this model does not consider the possibility that a new patch itself may create new vulnerabilities or a situation where a patch could be rejected and the vulnerable condition persist.

The best practice for implementing patching requires an iterative test process that takes days and sometimes even weeks to complete. Automatically triggered security patches are not desirable, nor recommended (Voldal 2003). In this model, two stocks *Released Patches* and *Accepted Patches* portray the testing process. The stock of *Released Patches* increases through the inflow from *Patches Release Rate*, and decreases through the outflow *Patch Test Rate*. *Patch Test Rate* is formulated simply by dividing the stock value by *Time to Test Patches*.

The stock *Accepted Patches* empties when the patches are deployed. Since the accepted patches will be deployed based on schedule, *Accepted Patches* becomes zero according to such schedule. The patch schedule is represented by the stock *Actual Patch Interval*. The model in Figure 2 assumes that the initial value for *Actual Patch Interval* equals *Initial Patch Interval*, namely, 180 days. The reason for choosing a patch interval of 180 days as an initial value is based on the experience of the company for our case. The company has a schedule for patching twice a year for their mission critical systems in order to avoid too much downtime because of maintenance. It means they deploy *Accepted Patches* twice a year. We model this assumption with a PULSE TRAIN function to represent the outflow (*Patch Deployment Rate*) from the stock of *Accepted Patches*.

> *(Accepted Patches/Average Downtime per Patch Batch)*
> **PULSE TRAIN(Actual Patch Interval, Average Downtime per Patch Batch , Actual Patch Interval, FINAL TIME).*

The height of the pulse equals *Average Downtime per Patch Batch* and has a constant value (1 day). *FINAL TIME* is defined as the end of simulation time. The initial time for patching and the patching frequency depend upon the length of the actual patch interval. The above equation means that one-day downtime per patch batch will occur every time the system is patched.

The vulnerability condition is not modelled explicitly. Implicitly, we assume that the time between patch release and patch deployment is a window of vulnerability: Malicious agents could exploit the situation, since an unpatched computer network open to attacks. *Total Unapplied Patches* as a sum of *Released Patches* and *Accepted Patches* is modelled to capture the total vulnerability the system.

Unapplied patches (represented by *Total Unapplied Patches*) increase *Risk of Downtime due to Incidents*. The latter variable is affected too by *Probability of Vulnerability Exploitation* (assumed to be a constant value = 0.003). The model assumes that the likelihood of malicious agents exploiting a vulnerability is low at the beginning. *Risk of Downtime due to Incidents* is also influenced by *Vulnerabilities per Patch*. The model assumes that one patch is produced for each vulnerability. Therefore the value for this parameter is unity. *Risk of Downtime Due to Incidents* will increase the stock *Downtime due to Incidents* through the flow *Increase in Downtime due to Incidents*. In turn, *Downtime due to Incidents* increases *Effect of Downtime on Patch Interval*. The stock *Downtime due to Patching* decreases *Effect of Downtime on Patch Interval*.

Threats and attacks are not explicitly modelled. They are captured by the concept of *Probability of Vulnerability Exploitation*.

Variables affecting the stock *Downtime due to Patching* are *Actual Patch Interval* and *Average Downtime per Patch Batch*. The flow *Increase in Downtime due to Patching* is formulated by using a PULSE TRAIN equation, so that the actual patch interval becomes an input for the function. The flow *Change in Actual Patch Interval* is formulated as the difference between *Desired Patch Interval* and *Actual Patch Interval*, divided by *Time to Change Actual Patch Interval*. *Desired Patch Interval* is *Initial Patch Interval* times *Effect of Downtime on Patch Interval*.

Whether the patch interval would be shorter or according to the schedule depends upon which one is bigger, *Downtime due to Incidents* or *Downtime due to Patching*. If the value of *Downtime due to Patching* is bigger than *Downtime due to Incidents*, patching will occur at the patching schedule. In the opposite case, patching will occur at shorter time intervals.

V. Base Run

The base run for the Main Model from Figure 2 is a situation in which new vulnerabilities are very rarely discovered, and the default model parameters define the base run; in particular, *New Patches* assumes the value of unity. The company will adjust its patch schedule depending on the occurrence of incidents.

In Figure 3 we show the base run for the three variables *Risk of Downtime due to Incidents*, *Patch Deployment Rate* and *Desired Patch Interval* over a period of two years. There are thirteen pulses in *Patch Deployment Rate*, that is, the accepted software patches are installed

together according to the current deployment schedule. The height of the pulses indicates the number of patches per deployment. Notice that the first deployment occurs when the *Desired Patch Interval* (red line) has decreased from initially 180 to about 72 days (y-axis); since the current time then happens to be the same, that is 72 days (x-axis), the patches are installed at that time. *Desired Patch Interval* continues to fall, and the second deployment occurs already at around the 130th day. *Risk of Downtime due to Incidents* shows a "saw-tooth," pattern since it is affected by *Total Unapplied Patches,* which is the sum of *Released Patches* and *Accepted Patches*. This behaviour is triggered by the PULSE TRAIN equation in *Patches Deployment Rate*. In the base run, the peaks of *Risk of Downtime due to Incidents* occur in the interval between 0.08 and 0.13 per day. Understanding the fluctuations of *Desired Patch Interval* and the relationship with the patch deployment rate and the risk makes its easier to understand the changes in the behaviour of risk and accepted patches in the next section.

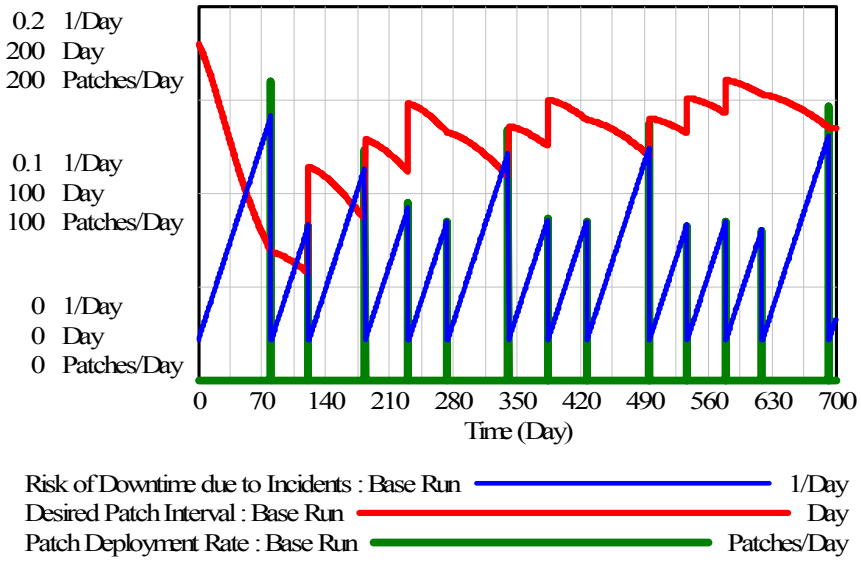

Figure 3
Base-Run Simulations of Risk of Downtime due to Incidents, Desired Patch Interval and Patch Deployment Rate

However, we want to further analyze this base run to determine whether the patch deployment schedules are triggered by *Downtime due to Incidents* or *Downtime due to Patching*. Figure 4 shows three different runs displaying *Increase in Downtime due to Patching*, *Increase in Downtime due to Incidents* and *the Effect of Downtime on Patch Interval*. In the base run we see that the average value of effect of downtime on patch interval is below unity. This implies that the downtime due to incidents is greater than the downtime due to patching.

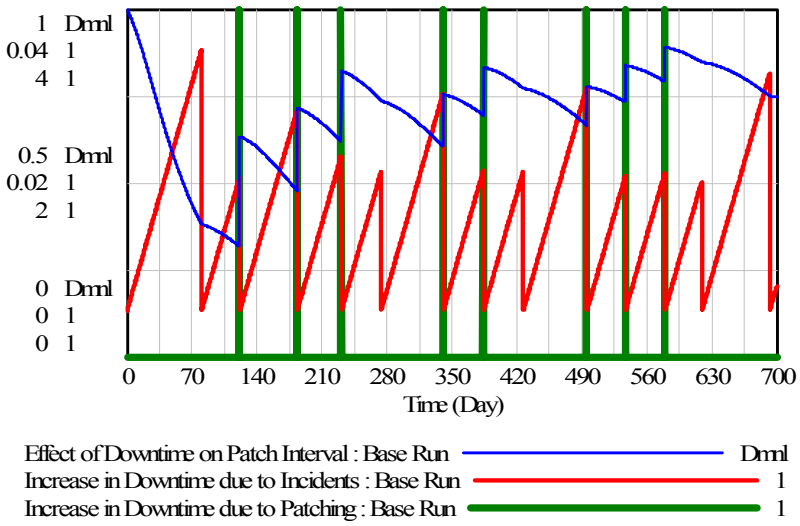

Figure 4
Base-Run Simulation of *Effect of Downtime on Patch Interval*, *Increase in Downtime due to Incidents* and *Increase in Downtime due to Patching*

VI. The Dynamics of the Model

In this section we experiment with various scenarios. As before, the simulation time period is 2 years (700 days). We concentrate on three interesting scenarios. For the understanding of the terminology below, the base run is set to *New Patches* equal to unity, implying a low rate of discovered vulnerabilities per unit of time.

1) Scenario where the vulnerabilities start to increase, represented by the release of new patches to fix the vulnerabilities (scenario *Vulnerabilities Increase*).
2) Scenario where the vulnerabilities are exploited more often at the same time, so that the probability of vulnerability exploitation goes up as well (scenario *Exploitation Increases*).
3) Scenario where the patch interval in a very vulnerable condition is adjusted to be shorter (scenario *Shorter Patch Interval*).

The changes of some parameters for analysis are as follows:

Table 1.
Parameter Changes

Parameter Changes / Scenario	New Patches	Probability of Vulnerability Exploitation	Initial Patch Interval
Base Run	1	0.003	180 Days
1. Vulnerabilities Increase	2	0.003	180 Days
2. Exploitation Increases	2	0.01	180 Days
3. Shorter Patch Interval	2	0.01	120 Days

1. Behaviour of *Accepted Patches*

Remember that the time between patch release and patch deployment is a vulnerable situation. When patches are deployed the risk of attack is assumed to be eliminated. We present the simulations of the three above scenarios compared with the base run in three separated graphs. Referring to Figure 5, in the first scenario the stock *Accepted Patches* rises at the beginning of the simulation, and it reaches a maximum peak, around 65 patches, in the first deployment. But afterwards it falls to approximately 22 patches in the next schedule, before it lifts up again. So the combination of two schedules with low amount of patches deployed and one period with high deployment occurred in this scenario. During the 700 days of the simulation period the patches are deployed sixteen times, three times more often than in the base run.

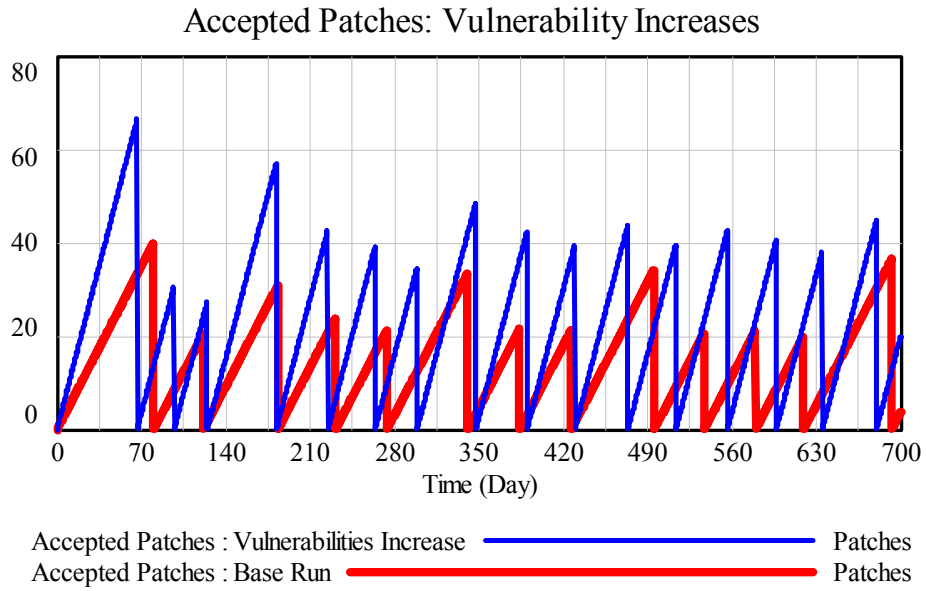

Figure 5
Simulation result for *Accepted Patches* (for the run *Vulnerabilities Increase*; base run shown for reference)

In the second scenario (Figure 6), patches are deployed more often, viz. twenty-three times during the simulation period. The number of patches deployed is high at the beginning but in the second year it drops to approximately fifteen patches. This low pulse pattern is repeated around nine times with lower values, and afterwards *Accepted Patches* goes up again and peaks at around 32 patches in the day 530. It stays high for the last five patch deployments, obtaining a local maximum of 45 patches around the day 600.

In third scenario (Figure 7) the patching time stays short since the beginning; a total of twenty-seven deployments occur during the simulation period. The number of patches per deployment is relatively low compared with the previous two scenarios, and the average peak height is almost the same as for the base run, though in third simulation, vulnerability and probability of vulnerability exploitation stay high.

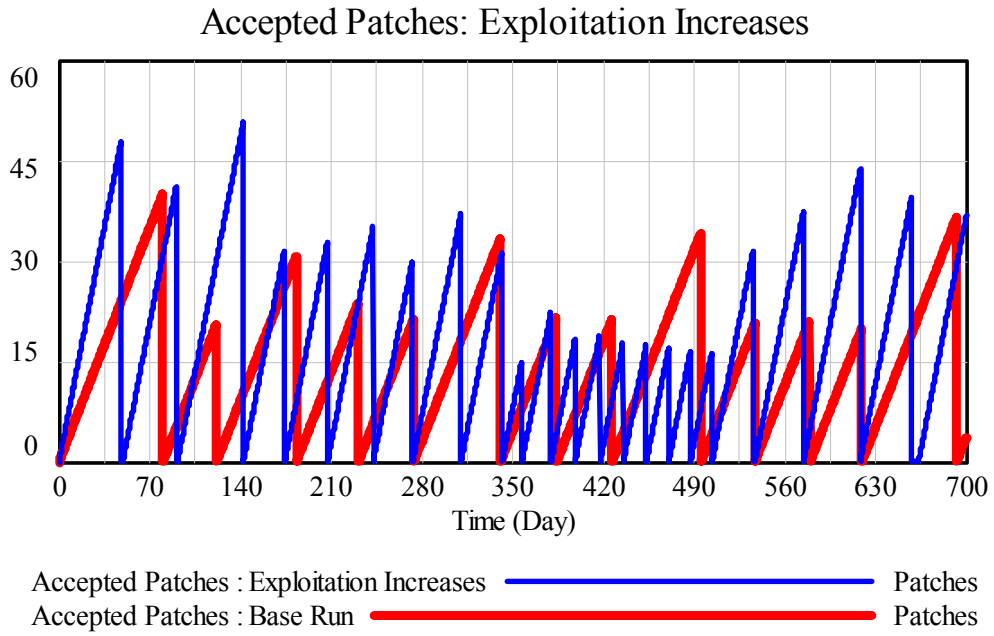

Figure 6
Simulation result for *Accepted Patches* (for the run *Exploitation Increases*; base run shown for reference)

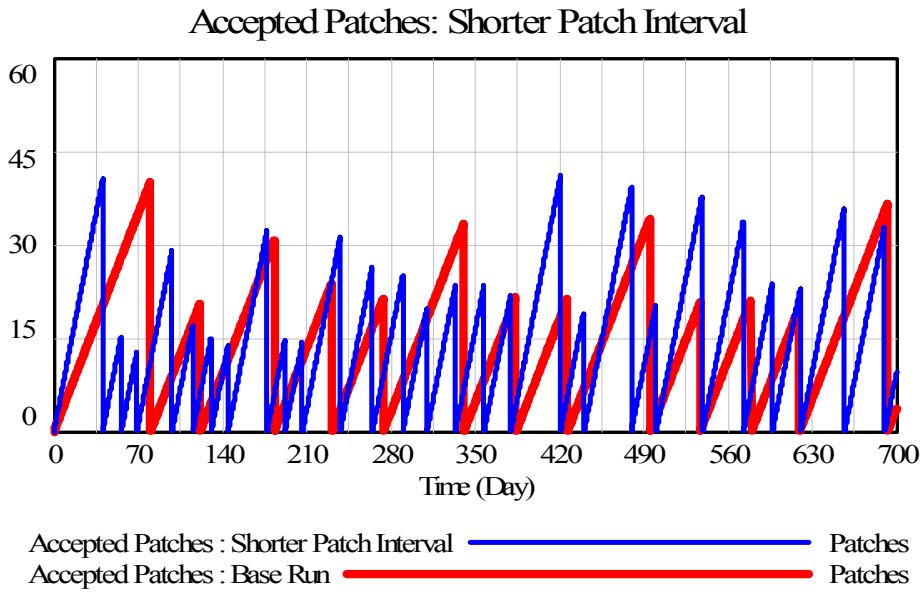

Figure 7
Simulation result for *Accepted Patches* (for the run *Vulnerabilities Increase*; base run shown for reference)

The variable responsible for the different patching intervals (*Actual Patch Interval*) is the *Effect of Downtime on Patch Interval*. Turning to *Accepted Patches*, this variable will fluctuate more often if *Actual Patch Interval* becomes shorter; this may happen if the incident downtime is greater than the patching downtime. A shorter *Actual Patch Interval* will produce

lower peaks in *Accepted Patches* and a longer *Actual Patch Interval* will produce higher peaks.

2. Behaviour of *Desired Patch Interval* and *Actual Patch Interval*

The results for the first scenario (green) are nearly the same as the results for the second scenario (red) in producing the desired patch interval over time. This is particularly apparent between day 140 and the end of the simulation time. However, the behaviours are very different for these two scenarios. The second scenario has more patch deployments because the probability of vulnerability exploitation is assumed to be higher.

In the second scenario (red) *Desired Patch Interval* drops in the beginning, rises and then falls. At the starting time, patching downtime is smaller than incidents downtime. But the structure of downtime changes afterwards and the system rapidly alters the desired patch interval into shorter patch intervals because of the higher incidents downtime. Hence, the adjustment process on *Actual Patch Interval* takes place in accordance with the change in *Desired Patch Interval*. The maximum value for *Desired Patch Interval* is 180 days since the value of *Initial Patch Interval can* not rise above 180 and the equation for *Desired Patch Interval* is *min(Initial Patch Interval * Effect of Downtime on Patch Interval, Initial Patch Interval)*. Because the model assumes, if there are few incidents, then patching will be implemented in accordance with the company's patching policy, i.e. *Initial Patch Interval*, of 180 days defines the schedule.

In the third scenario (blue), the effort to balance the increasing rate of new patches and the probability of vulnerability exploitation through the adjustment in the patch interval, lead to a more stable and shorter patch interval. As a consequence, there is a lower average amount of patches per deployment.

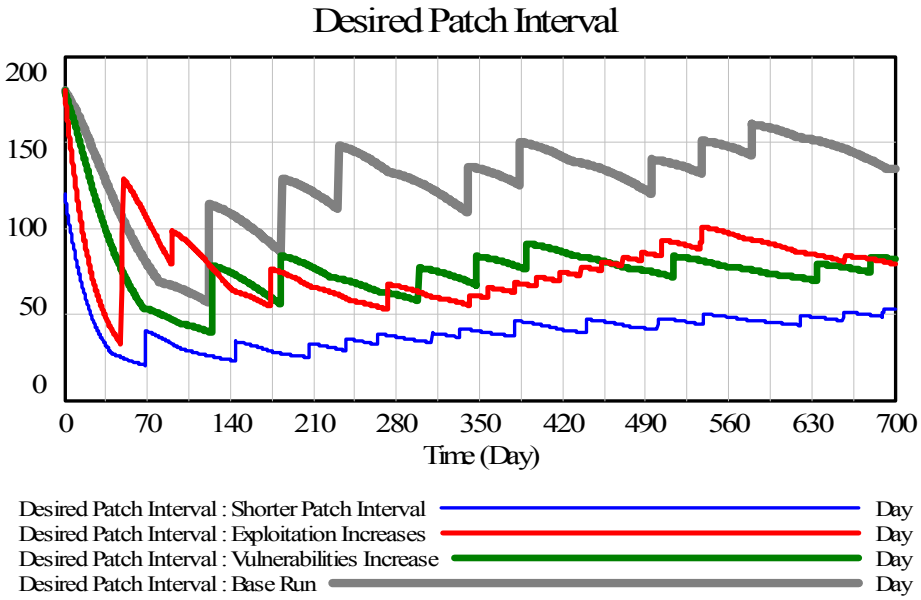

Figure 8
Simulation results for *Desired Patch Interval*

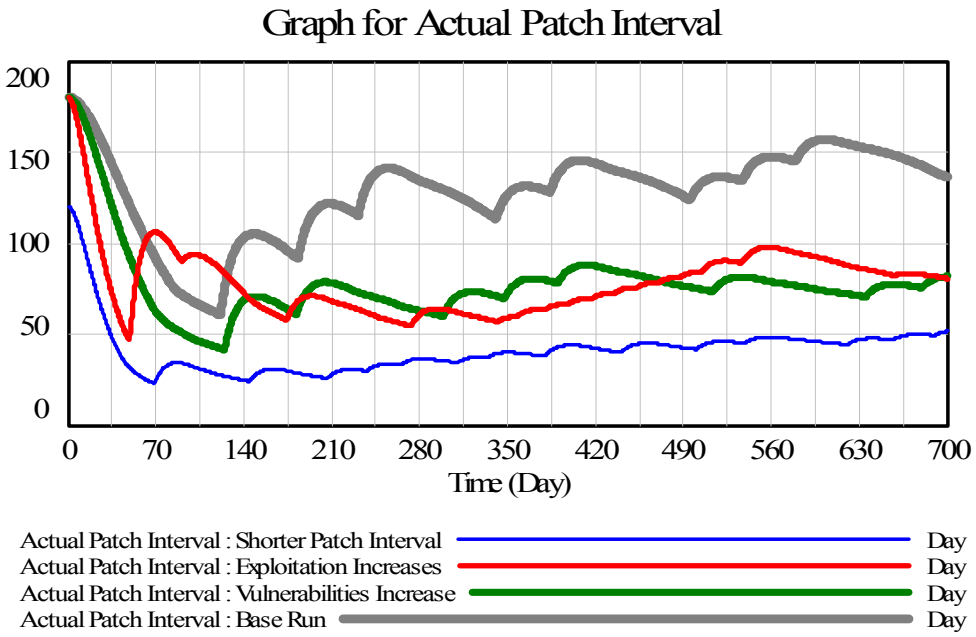

Figure 9
Simulation results for *Actual Patch Interval*

3. Behaviour of *Effect of Downtime due to Patching*

With reference to Figure 10, in the first and second scenario (without policy to adjust patch interval) as well as the third scenario (with policy to adjust patch interval), the average effect moves between the values 0.2 and 1. The incident downtime is greater than the patch downtime. Since *Desired Patch Interval* equals the product of *Effect of Downtime due to Patching* (which is less than unity) with *Initial Patch Interval*, the result is a reduction of the value of *Desired Patch Interval*. It implies that system administrators will patch more often.

The second scenario (red) has an effect of around 0.3 in day 70 and afterwards increases until it reaches the value 0.75 after 60 days. It is the highest peak for the second scenario. This means that the incident downtime (when the effect value is below unity) is higher than the patching downtime. This implies further that the system will be patched more often, and signifies that system administrators are aware of the threat and will shorten *Desired Patch Interval*. The situation will be the opposite when the effect value is above unity. Here the situation changes, since the patching time is higher than the incident downtime.

In the third scenario (blue), patching is performed more often, since the system administrators adjust the initial patch interval value to become shorter. Although in scenario three we assume that the situation is more vulnerable than in scenario one, one has a greater frequent patch interval. In the next section we will see that the risk in this scenario is fairly low over time.

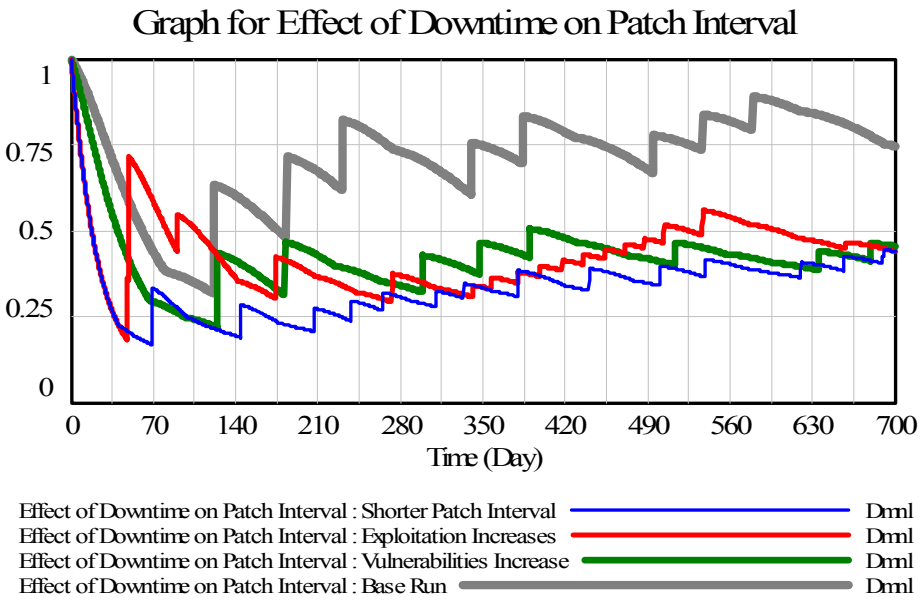

Figure. 10
Effect of Downtime on Patch Interval

4. Behaviour of *Risk of Downtime due to Incidents*

All adjustments of the patch schedule affect patching deployment behaviour and eventually will have an effect on the behaviour of risk of incidents downtime. We also present the simulation results in three different graphs—all are compared with the base run. In the first scenario we see that the average peaks of risk over time are relatively low compared with the other two scenarios (Figure 11). The average peak risk value moves between the lowest value around 0.05 and the highest value approximately 0.25. After each patch deployment, the risk drops to its lowest point in the first scenario.

The highest risk comes from the second scenario (Figure 12) when the increase in vulnerabilities is followed by the increase in the probability of exploitation. More than half of the simulation time the risk peaks more or less between 0.15-0.65, although at the middle of simulation the risk of incident downtime is low, especially in the interval from day 350 to 530. But average risk peaks between values 0.5-0.6.

The main message from our analysis is the notion to patch more often to counteract risks. In the second and third scenarios the probability of vulnerability exploitation is equal, but in the third scenario the shorter patch interval reduces the average risk to peak around 0.28- 0.5. But a majority of peaks of *Risk of Downtime due to Incidents* move approximately between 0.28-0.35 per day (Figure 13).

As a whole, the risks after patching never vanish, because it is assumed that over time new vulnerabilities will be discovered. Patching, therefore, helps to diminish the risks, but it can never totally eliminate them. We consider this assumption to be realistic, that the number of known vulnerabilities always grows, and that security measures are a necessary step to diminish the probability that third parties will (intentionally or unintentionally) exploit the weaknesses in the computer network system.

Ultimately, in this model we assume that the "*saw-tooth*" pattern in the simulation appeared because of the nature of mission critical systems. Such systems cannot be taken off-line too often because this would lead to a too large production loss. Therefore patching has to be done according to schedule. To patch continuously for mission critical systems is undesirable.

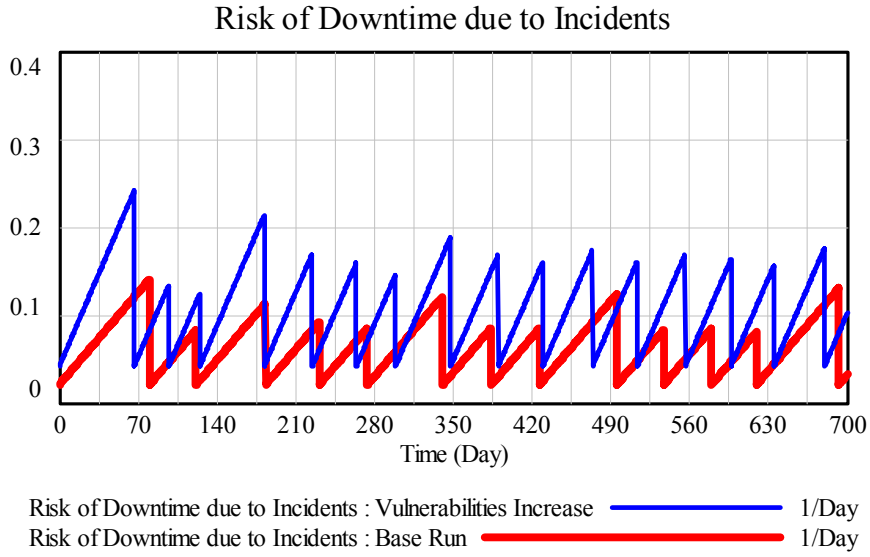

Figure 11
Simulation results for Risk of *Downtime due to Incidents* (for the run *Vulnerabilities Increase*)

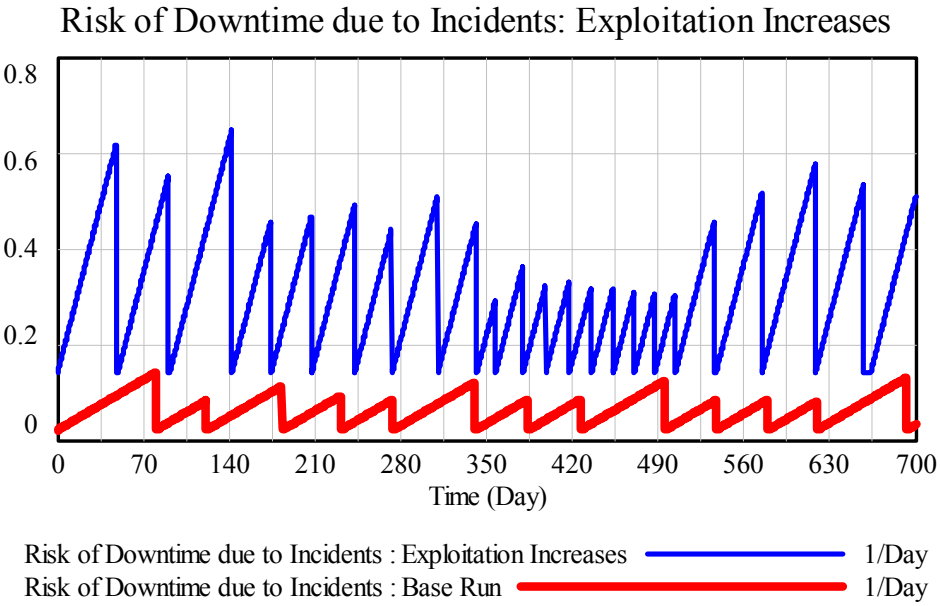

Figure 12
Simulation results for Risk of *Downtime due to Incidents* (for the run *Exploitation Increases*)

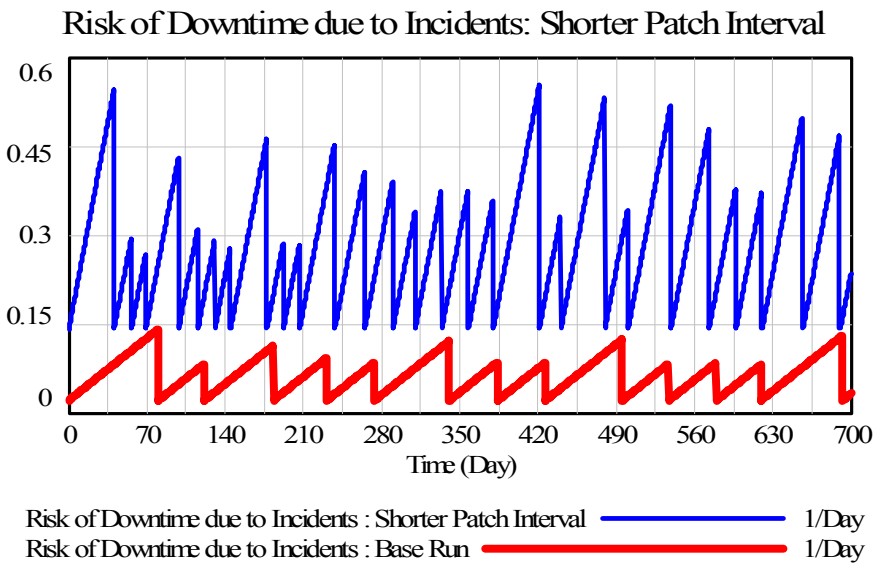

Figure 13
Simulation Results of Risk of *Downtime due to Incidents* (for the run *Shorter Patch Interval*)

5. Behavior of Downtime Due to Incidents vs. Downtime Due to Patch

As we see in Figure 2, *Downtime due to Patching* and *Downtime due to Incidents* are formulated as stocks. Therefore the behaviors of these two variables are actually an accumulation of the inflow of *Increase in Downtime due to Patching* and *Increase Downtime due to Incidents*.

The base run for this model assumes that the system has a low threat. The simulation shows that the amount of downtime due to patching is approximately nine days (Figure 14) and downtime due to incidents is around twelve days (Figure 15). When we try to increase the vulnerability for the first scenario (green), we can observe that in the two years of the simulation time period the downtime due to patching does not change too much, but downtime due to incidents starts rising. Moreover, if the exploitation increases as in the second scenario (red) or the threat to the system is assumed to be high, we would observe that the amount of downtime due to incidents increases significantly to approximately fifty days, compared with the base run and the first scenario. Although this situation makes the downtime due to patching also increase (around twenty-two days), the change is not too high as the unplanned downtime.

The third scenario (blue) shows that regulating patch schedule reduces the downtime due to patching, since it is shown that at the end of simulation time, although so far it reduce only around two days within two years, compared with the third scenario. But this scenario also reduces the number of downtime due to incidents. We also observes that the pattern of *Shorter Patch Interval* in the second scenario (Figure 14) shows a reactive way to adjust patch interval any time incidents happens, especially between days 340 and 530. (While in the third scenario, we see a more regular pattern of downtime due to patching.) This pattern actually will affect *Accepted Patches* and in turn also influence *Risk of Downtime Due To Incidents*, for having lower peaks.

However, as long as the probability of exploitation exists, we never really reduce the downtime due to incidents to zero. But we are able to reduce the risk of having too much downtime due to incidents by having a regular patching schedule.

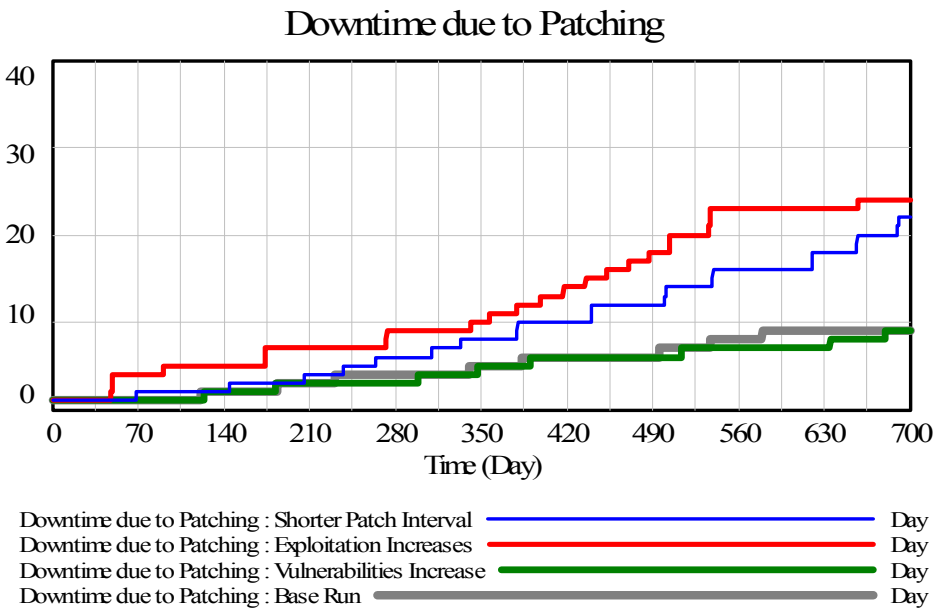

Figure 14
Simulation results for *Downtime due to Patching*

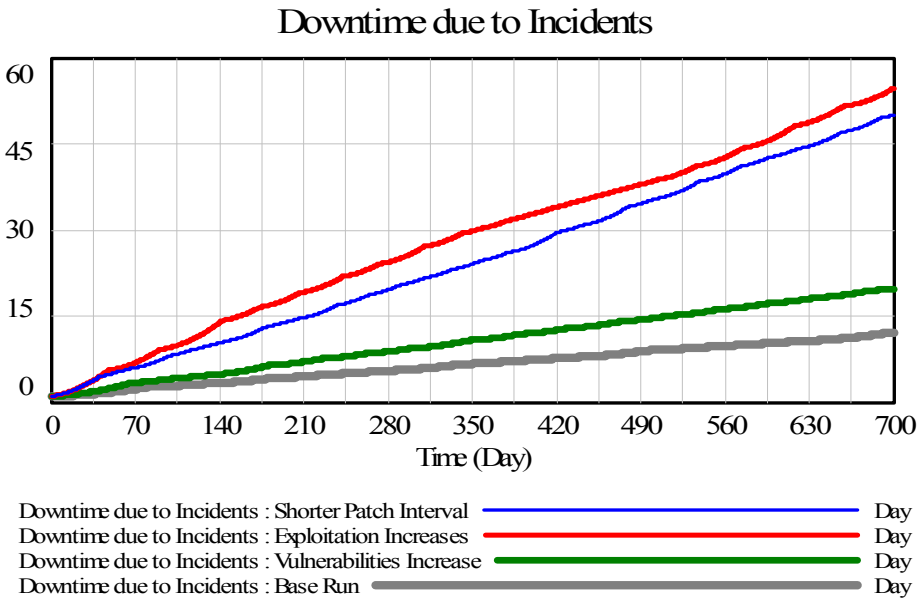

Figure 15
Simulation results for *Downtime due to Incidents*

VII. Discussion and Further Development

In the first two scenarios we learn how the system reacts toward the growth of vulnerabilities. System administrators will adjust the interval time to patch, but this always happens only after the risk goes up. They respond to the higher incident downtime and patch faster by adjusting the patch time interval after experiencing a high degree of vulnerability. When the risk is perceived to be low, they will adjust to a longer patch time or to patch interval. In other words, there is no pressure to patch more. As a consequence, in the next patching schedule a high risk of downtime occurs before patches are deployed. As referred before, the patching schedule of twice a year is based on the practice of one company that uses this method to avoid too much downtime. At this moment, whether an increased patch frequency will reduce the possibility of exploitation by malicious agents is only demonstrated by *Risk of Downtime due to Incidents*' variable. Based on existing model structures, the risk will go down as the patching frequency increases. In turn this policy may reduce also the downtime due to incidents.

This leads us to the term *proactive and reactive patching*. These issues come up when people discuss patching management and how to secure the network system (Packard 2006). The simulation results without a regular patch schedule reveal a behaviour that we could associate with reactive patching. The reaction for patching comes after incidents are occurred. Reactive patching is typically carried out under stressful conditions. It is particularly important during this kind of situation that care be taken to clearly identify the cause of the problem and then to fix only what is broken. The tendency to apply several fixes together, hoping that one of them will solve the problem, is not a secure approach. At best the problem will be fixed, but the solution may still remain unclear. At worst, the problem may be compounded, making restoration even more difficult. In practice, when using reactive patching, people still must devote sufficient time to diagnose the problem and to ensure that the patch to be installed will solve it.

Our results are in line with the study by Wiik et al. (2004), although they used a different perspective and made certain assumptions regarding software vulnerability. Wiik et al.'s study shows why vulnerabilities can be readily exploited. In their conclusion, this exploitation can be attributed to the reactive way in responding to the incidents.

The policy of proactive patching in our model is actually shown in the third scenario, when the growth of vulnerability and probability to attack is combined with the reduction of interval patching (i.e. by setting *Initial Patch Interval* to a lower value). We find more stable patterns in the third scenario in term of the risks, which stay at lower values. In practice, proactive patching, unlike reactive patching, can be scheduled and carefully controlled. The aim of proactive patching then is to prevent system downtime due to known defects/vulnerabilities. In some cases (Packard 2006), these are latent defects that will surface under certain conditions or at certain points in time. Other examples of defects that can be corrected through proactive patching include panics, hangs, security holes, and memory leaks. Our model indicates that regular patching helps to diminish the growth of these vulnerabilities.

Finally, all discussion and conclusions are still in a preliminary stage. Our model needs further development in order to account for all the important factors. More research is needed

to look into different network systems, various degrees of critical mission, functions (technical or office), kinds of vulnerabilities, and different requirements for patching.

Literature

Arbaugh W.A., W.L Fithen, and J.M Hugh. 2000. Windows of Vulnerability: A case Study Analysis. *Computer* 33 (12):52-59.

Beattie, S., S. Arnold, C Cowan, P Wagle, and C Wright. 2002. Timing the Application of Security Patches for Optimal Uptime. *LISA* XVI (November 3-8, 2002):101-110.

Bentley, A. 2006. *Developing a patch and vulnerability management strategy* 2005 [cited January, 25 2006]. Available from http://www.scmagazine.com/uk/news/article/523151/developing-patch-vulnerability-management-strategy/.

Easttom, C. 2006. *Computer Security Fundamentals*. New Jersey, Upper Saddle River: Prentice Hall.

Hulme, G.V. 2006. *Under Attack* 2004 [cited February 12 2006]. Available from http://www.informationweek.com/industries/showArticle.jhtml?articleID=22103493&pgno=1.

Hurley, E. 2006. *Keeping up with patch work near impossible* 2006 [cited January, 25 2006]. Available from <http://searchsecurity.techtarget.com/originalContent/0,289142,sid14_gci796744,00.html>.

Landwehr, C.E, A.R. Bull, J.P. Mc. Dermott, and W.S. Choi. 1994. A Taxonomy of Computer Program Security Flaws, with Examples. *ACM Computing Surveys* 26 (3).

Lynn, Tracy. 2006. *Vulnerability Risk mitigation--Patching the Microsoft Windows Environment* 2002 [cited February 3 2006]. Available from http://www.sans.org/rr/whitepapers/windows/291.php.

Martin, R.A. 2001. Managing Vulnerabilities in Networked Systems. *Computer* (November 2001):32-38.

Packard, Hewlett. 2006. *Patching Mission Critical* 2006 [cited February 10 2006]. Available from http://docs.hp.com/en/945/patching_mc.pdf.

Peltier, T.R. 2005. *Information Security Risk Analysis*. London: Auerbach Publications.

Qualys, Inc. Research Team. 2006. *The Laws of Vulnerabilities, 2005 Edition* 2006 [cited January, 30 2006]. Available from http://www.qualys.com/docs/law_of_vulnerabilities.pdf.

Schneier, Bruce. 2000. *Secrets and Lies: Digital Security in a Networked World*. New York: John Wiley & Sons, Inc.

Seacord, R.C., and A.D. Householder. 2005. *A Structured Approach to Classifying Security Vulnerabilities*. Carnegie Mellon Software Engineering Institute 2005 [cited December, 22 2005]. Available from http://www.sei.cmu.edu/pub/documents/05.reports/pdf/05tn003.pdf.

Voldal, D. 2006. *A Practical Methodology for Implementing a Patch Management Process* 2003 [cited January, 25 2006]. Available from http://www.sans.org/rr/whitepapers/bestprac/.

Wahlström, B. 2005. Risk Assessment and Safety Engineering; Applications for Computer Systems. Paper read at the 24th International Conference on Computer Safety, Reability and Security, at Fredrikstad, Norway.

Wiik, J., J.J. Gonzalez, H. F. Lipson, and TJ. Shimeall. 2004. Dynamics of Vulnerability--Modeling the Life Cycle of Software Vulnerabilities. Paper read at The 22nd International System Dynamics Conference, July 25-29, 2004, at Oxford, UK.