Supporting Material is available for this work. For more information, follow the link from the Table of Contents to "Accessing Supporting Material".

Dynamics of Vulnerability – Modeling the Life Cycle of Software Vulnerabilities

By
Johannes Wiik
Jose J. Gonzalez
Faculty of Engineering and Science
Department of Information & Communication Technology
Agder University College
Grooseveien 36
NO-4876 Grimstad, Norway
Phone: +47 37 25 30 00
Email: Johannes.Wiik@hia.no Jose.J.Gonzalez@hia.no
Howard F. Lipson
Timothy J. Shimeall
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA, USA 15213-3890
Phone: +1-412-268-7700
Email: hfl@cert.org tjs@cert.org

Abstract

Many of the contributing factors to computer security problems are non-technical in nature – that is, they are dependent upon human and organizational actions and interactions in the political, social, legal, and economic realms. However, much of the research in computer security has had a predominantly technical focus. This paper represents a first attempt at using the concepts of system dynamics to model some of the human and organizational actions and interactions that impact the *software vulnerability life cycle*, which represents the relationship over time between the discovery of security vulnerabilities (i.e., flaws) in software and the occurrence of computer security incidents based on the exploitation of those vulnerabilities by attackers. Although our initial model relies on several simplifying assumptions, it points the way towards richer and more comprehensive models that can increase our capabilities and understanding in ways not possible through traditional computer security research approaches.

Introduction

Our society has gradually become more and more dependent on information and communication systems, in business, in education and in our private lives. Consequently, we are also increasingly vulnerable to errors in such systems. It is therefore important to understand how we can maintain such systems in a more secure state.

In his influential book, Schneier (2000) characterizes cyber security as a "process", not a "product", and a very complex process for that matter. Schneier writes: "The Internet is probably the *most complex system* [our emphasis] ever developed". He proceeds to describe information systems in much the same way a system dynamics experts look at them: As dynamic entities, with interacting components, propagating consequences, unexpected ("emergent") properties and delayed effects. Schneier stresses the vulnerability of information systems, threats and risks, policies and attack recovery.

Over the last six years, the number of incidents reported to the CERT® Coordination Center (CERT/CC)[1] has practically doubled every year.[2] Technology alone is not enough to prevent such incidents from taking place because of the complexity of today's information systems. There is no overall centralized control, as systems cut across organizational boundaries as well as other boundaries. Hence, security policies are not consistent with one another.

There is constantly an ongoing battle between attackers and defenders. On the one hand, attackers try to exploit a vulnerability. On the other hand, defenders try to manage the security of systems, for example through delivering and installing patches.

However, the surprising part of this story is that very often the solution that can remove the vulnerability is readily available before the vulnerability becomes exploited on a large scale (Arbaugh, Fithen, and McHugh 2000).

The scope of this paper is to enhance our understanding of the complex structure that is driving the number of intrusions for a single vulnerability over time, and to discuss some policies that can influence this problem by using a system dynamics-based simulation model.

® CERT is a registered trademark of Carnegie Mellon University.

[1] Founded in 1988 as the first Computer Emergency Response Team Coordination Center, the CERT/CC is currently operating with a much wider range of computer security activities than its original incident response function. The CERT/CC carries on activities and research within the areas of vulnerability analysis, incident handling, survivable enterprise management, education and training, and survivable systems engineering. For further information see: http://www.cert.org/meet_cert/meetcertcc.html#bkgd

[2] See CERT/CC statistics (2004) at http://www.cert.org/stats/

Computer Security Defined

Computer security has been formally defined as follows (Howard 1997)[3]:
"Computer security is preventing attackers from achieving objectives through unauthorized access or unauthorized use of computers and networks." An attacker's objectives can include theft (i.e., disclosure) of confidential data, corruption of data, or denying system services to legitimate users (e.g., by crashing a system or overloading its resources[4]). An attacker may wish to gain control of a computer system to make use of its resources, often for use in subsequent attacks on other systems.[5]

An attacker can use non-technical means as a first step towards compromising computer security, such as tricking a naive user into revealing his or her password. However, attackers typically achieve their objectives by exploiting flaws (*vulnerabilities*) in software (Hoglund 2004). Many software products are ubiquitous and once a vulnerability is discovered in a widely-used software product, a large population of computer systems that have those products installed are at risk.

The CERT/CC (CERT Coordination Center) has defined several security terms, and we will briefly summarize some of these definitions. See also Arbaugh, Fithen, and McHugh (2000) for a more extensive description and list of definitions. A *vulnerability* is a flaw or a defect in technology that can be exploited. A vulnerability exploit (possibly a scripted collection of commands or a program) can cause the technology to function insecurely. Hence, an intruder launches an *attack* if he or she tries to elicit a different behavior from the target system in order to reach a certain goal. From a defender's perspective an *attack* is any event that targets the defender's system in an insecure manner (based on the defender's perception of insecure). If the attack elicits a different behavior, it is considered to be an *intrusion*. We also need to differentiate between events and incidents. An *event* is the most basic unit of information that describes the aspects of network or host behavior. A group of events that relate to one another such as scanning and attacks is referred to as an *activity*. An *incident* is an activity that violates stated or implied security policy (and may be attempted or successful execution of vulnerability exploits). In other words, incidents refer to the aggregated activity generated by events. Collections of incidents are sometimes identified with the vulnerability or exploit involved in the incidents.

---

[3] For an in-depth discussion of the derivation of this definition (and its historical context), see Chapter 5 of *An Analysis of Security Incidents on the Internet 1989 - 1995* (Howard 1997), which is available on-line at: http://www.cert.org/research/JHThesis/Chapter5.html

[4] Such attacks are classified as *denial-of-service (DoS)* attacks.

[5] Attacking through intermediate machines, or *stepping stones*, makes it more difficult to track the originator of an attack. Gaining control of a large number of machines allows an attacker to coordinate an overwhelming *distributed denial-of-service attack (DDoS)* against a selected target.

Systems typically oscillate between different states with respect to vulnerabilities. We can distinguish between (Arbaugh, Fithen, and McHugh 2000):

1. *Hardened*
2. *Vulnerable*
3. *Compromised*

A system is *hardened* when all security corrections have been installed. As new vulnerabilities are discovered and disclosed, the system enters a *vulnerable* state. The system administrators need to install security corrections such as patches in order to get the system back to a hardened state. Otherwise, there is a risk that one or several vulnerabilities might be exploited. In that case, the system enters a *compromised* state.

The Problem with Cyber Data

Unfortunately, systematically collected information about attacks on information assets is generally not available. There are several reasons for this:

1. The attackers generally act to conceal as many aspects of their attacks as they can.
2. The defenders gather data on attacks for quite narrow purposes
3. Organizations controlling information assets are very reluctant to share data on attacks on those assets.

Firstly, attackers typically try to conceal their objectives and methods for several reasons. They do not want to be detected, and, in addition, they do not want to reveal how they work. Otherwise, the element of surprise attack would probably soon be lost.

Secondly, from a defenders perspective, information gathering is very often limited to a specific defensive task or for legal evidence gathering for a specific incident. Hence, data is not available in a generic format - something that renders it difficult and time consuming to analyze.

Thirdly, information sharing is hindered, not only for legal reasons, but also for fear of negative publicity and potential loss of reputation. The fear that others will copy a certain type of attack is another reason for not sharing information. Ironically, this latest fear may even help attackers keep the element of surprise relative to other organizations and targets.

Consequently, system dynamics modeling of computer security may have to rely more on expert knowledge than hard systematically collected data. However, such a modeling process may in turn generate new insights into the problem domain and identify areas where systematic data collection might be of great importance for future research and future modeling. Even though written sources of information and numerical data are important, effective modeling building is dependent on human knowledge, and most often, essential information is drawn from the mental database of people (Forrester 1994). Hence, a useful model can be made even with limited data available.

Reference Modes

When we want to develop a system dynamics model, our point of departure is very often the reference modes of the problem. Typically, a reference mode shows a time profile with data gathered for a key variable related to the dynamic problem at hand, but it can also be a drawn graph that focuses less on the exact values but rather on the behavior pattern related to the problem.

The information available for the single vulnerability life cycle is rather limited. Figure 1 presents an idealized graph of the vulnerability exploit cycle. There are several key assumptions behind the behavior of exploits or intrusions over time. At first the number of exploits for a certain vulnerability is low as such attacks are launched without the use of automated tools. As exploits tools, such as scripts, are developed and spread within the hacker community, the rate of exploits increase leading to a wide-spread use of the tools. This process triggers reactive defense mechanisms such as the development of patches that can be installed as the general awareness about the problem increases. Hence, the rate of exploits starts to decline.

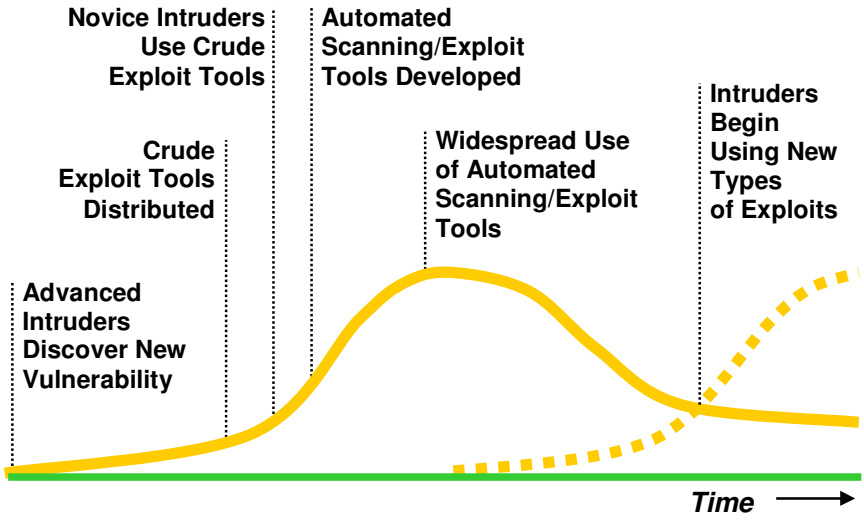

Figure 1: Vulnerability Exploit Cycle[6]

Even though the figure above gives us some information about the problem, not all single vulnerability life cycles may develop as described above. Three case studies described by Arbaugh et al. (Arbaugh, Fithen, and McHugh 2000) tell a slightly different story and add more detail to the development and implementation of patches and other preventive measures.

In the story above we assumed that a growing rate of intrusions will force the software vendor to develop a patch that can remove the vulnerability being exploited. However, in all three case studies, a patch or correction to the problem was available

[6] Source: CERT Coordination Center, © 2002 by Carnegie Mellon University. See Lipson 2002 for some additional details.

long before the peak in incidents actually took place. This phenomenon is illustrated in the Phf[7] incident histogram in the picture below taken from Arbaugh et al. (2000). The Phf vulnerability made hackers capable of executing commands on web servers at the privileged level of the HTTP server daemon[8]. The vulnerability could easily be exploited with or without scripting. As the graph shows, there were quite a few intrusions before a script was available, but the majority of the intrusions took place after the script was available. Most of these intrusions have been attributed to script-kiddies – less experienced hackers that depend on tools made by others to launch an attack (Arbaugh, Fithen, and McHugh 2000). As such actors are not always aware of what such scripts can be used for, they may even launch attacks more or less blindly.

The graph in Figure 2 shows not only the reported incidents, but also the point in time when the vulnerability was discovered, corrected and scripted. In addition the continuous curve shows a new idealized reference mode slightly modified compared to the graph in Figure 1. The difference between the two reference modes reveals some interesting findings.

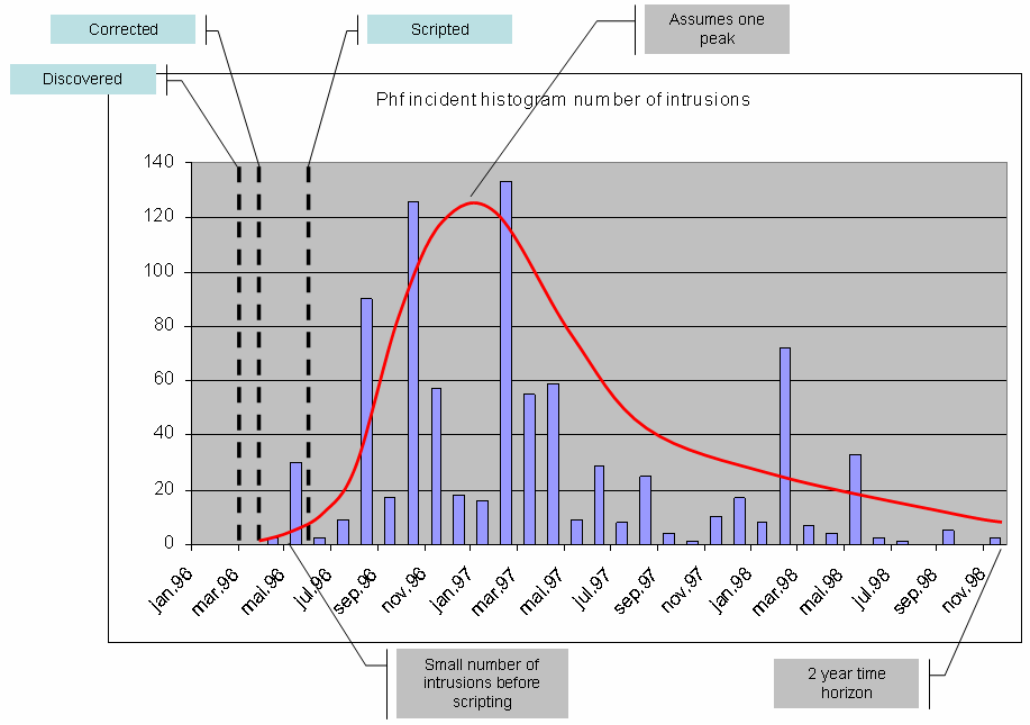

Figure 2: Phf incident histogram of intrusions

[7] A well-known CGI bug was in the program Phf. This program is used as a telephone book database script. Attackers could use a bug in Phf that allowed them to run the Phf program and input a new line character (%0a) to enter a new command into the shell, the Phf program did not check input and presumed the user was inputting a valid command. This technique was mostly used by an attacker to grab the /etc/passwd file on a vulnerable system. This file contains all user and encrypted password information which an attacker would then proceed to crack and attempt to gain access to the system.
[8] Daemon: Processes in the UNIX system are either user, daemon or kernel processes. Daemon processes are not associated with any users but perform system-wide functions, such as administration.

Firstly, and most importantly, there were no intrusions before there was a correction available. This is very interesting as it means that a solution was available before the incident occurred as opposed to the reference mode in Figure 1. Unfortunately, the solution was not used quickly enough. The two other case studies by Arbaugh et al. gave similar findings. There might be many reasons why system administrators do not patch their systems in time. It takes time for the news about a patch to spread and time to install it. System administrators are cautious about installing corrections without prior testing as a patch might create other problems in the system (Arbaugh, Fithen, and McHugh 2000). The problem may also be attributed to the decline of the expertise of the average system administrator (Lipson 2002). The rapid growth in the number of attacks is attributed to the automation of exploitation. That is, the development of tools that reduce the knowledge needed to make an attack increases the number of potential hackers that are able to exploit the vulnerability. Most of these hackers are therefore script kiddies. As use of these tools spreads in the hacker community, attacks and intrusions increase exponentially (Arbaugh, Fithen, and McHugh 2000).

Secondly, the shape of the curve indicates that there is a rapid growth in the intrusion rate. However, it does not decline as rapidly as it grows. Consequently, the problem persists for a long time after the awareness of the problem was raised and a correction was made available. Much of the decline can be attributed to the loss of interest among the intruders as more and more systems get patched and consequently enter a hardened state (Arbaugh, Fithen, and McHugh 2000).

How can we generalize to a generic system dynamics model? As several life cycles of different vulnerabilities tend to develop along a similar pattern, we may assume that there is a common causal feedback structure determining this behavior. Although the reported intrusion data is admittedly inaccurate, as long as we capture the feedback structure we should be able to get results that generate a similar behavior pattern that we can rely on based upon available of data and on past research.

The Model Structure

Based on the reference modes, literature, and expert knowledge, we built a generic model of the dynamic problem: the single vulnerability life cycle. That is, the model is not made to represent a particular incident, and must therefore be treated as such.

The total model is pictured in Figure 3. We have not included the actually discovery of the vulnerability in the model. The assumption is that the vulnerability has been disclosed and the information is available to administrators, as well as hackers, within the time frame of the model.

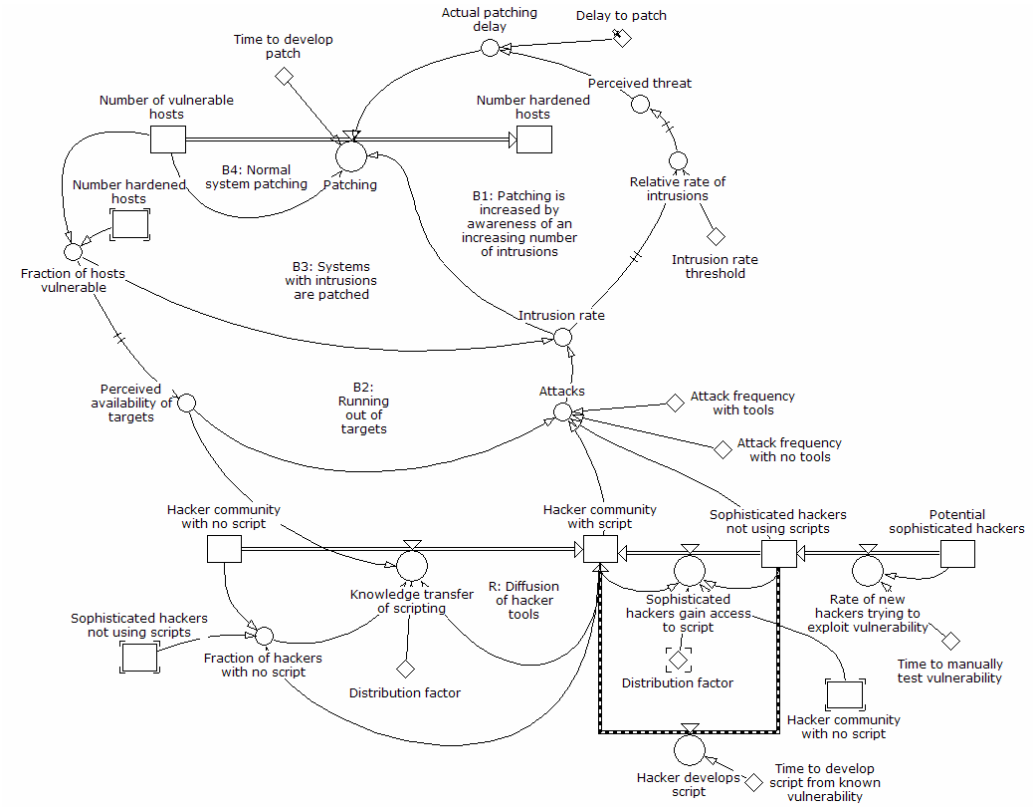

Figure 3: An overall view of the stock and flow model

We have divided the model into two main sectors:

1. An attack sector describing how hackers exchange information and use this information to make attacks on hosts.
2. A defense sector describing how host administrators react to a threat regarding a single vulnerability.

In many ways this structuring of the model follows Schneier's notion that in order to understand the process of computer security, it is necessary to consider attacks, defenses and the relationship between them (Schneier 2000, p. 273). From a system dynamics perspective, this can be interpreted as the feedback processes between attack policies and defense policies.

The Attack Sector

According to the reference modes discussed previously, the first type of attack is made by advanced hackers. Gradually advanced hackers will start testing out ways of attacking vulnerable hosts with a certain frequency. After a delay, the knowledge from such attacks is used to develop a script that can automate the exploitation of a particular vulnerability. We did not include the process of further improved attack tools in this model, but we have kept it on an aggregated level with one type of automated tool that we call script. Hence, we used a single stock of hackers with script.

The availability of a script has several important implications: Firstly, the attack frequency of hackers that use the script will increase because it will make each hacker able to attack more targets per day than if it had been done manually. Secondly, a script makes the potential number of hackers available to exploit the vulnerability much larger as the necessary knowledge to use a script is much lower than to use a manual approach.[9] Thirdly, as a consequence, the script will start spreading exponentially through word of mouth as more and more hackers gain access to the script. This reinforcing feedback process is captured in the reinforcing feedback loop "R: Diffusion of hacker tools" as seen in figure 4.

The above mentioned factors all contribute to an increase in the number of attacks and consequently the intrusion rate as well.

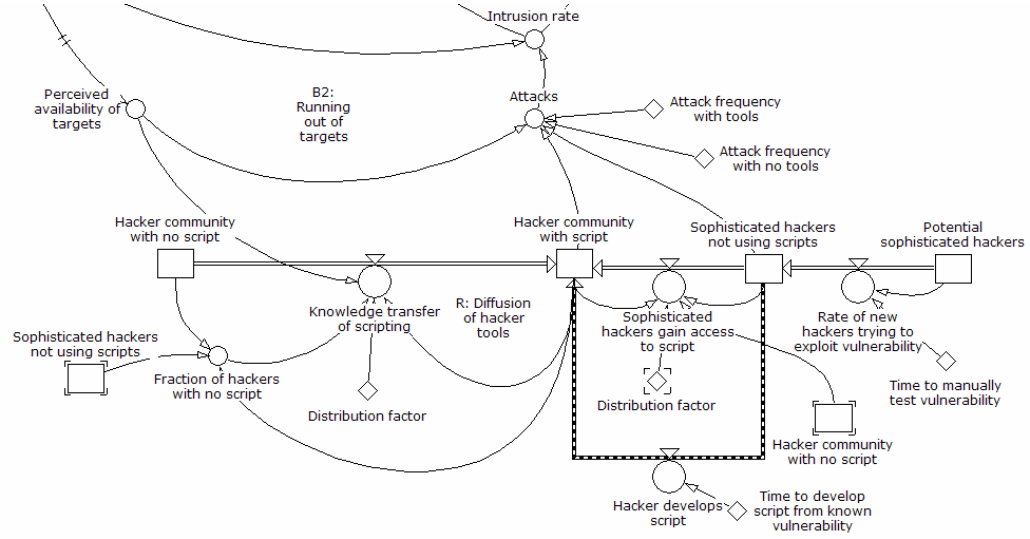

Figure 4: The attack sector of the model

[9] Very often, scripts also bundle multiple vulnerabilities to increase the probability of a successful attack. However, this is beyond the scope of the model of this paper as we only investigate the life cycle of a single vulnerability.

The Defense Sector

There are many ways of measuring vulnerable targets, but in our model we have used hosts as described by Arbaugh et al. (Arbaugh, Fithen, and McHugh 2000). As previously mentioned, systems normally oscillate between hardened and vulnerable. Of course, sometimes they might get compromised.

In our model, we have only included hardened and vulnerable hosts explicitly, as we consider this to be sufficient in order to describe the vulnerability life cycle. Compromised hosts are also included in the sense that successful attacks create intrusions, but the model assumes that these hosts are patched immediately after a successful attack.

Patching is the main defense mechanism in the model. We will not consider other defense mechanisms. For example, for some vulnerabilities, changes in firewall rules may thwart an attack. Nor will we consider improvements in engineering processes that can reduce future vulnerabilities based on lessons learned. On average, there is a long delay for a large number of hosts to become patched and some hosts may never be patched at all. However, this delay time can be reduced if the perceived threat is high. In other words, if the incident rate is quickly increasing and becomes sufficiently high, awareness about the vulnerability will rise. Consequently, system managers will be much more eager to patch their systems. In general system managers are reluctant to update their systems unless they see an imminent threat.

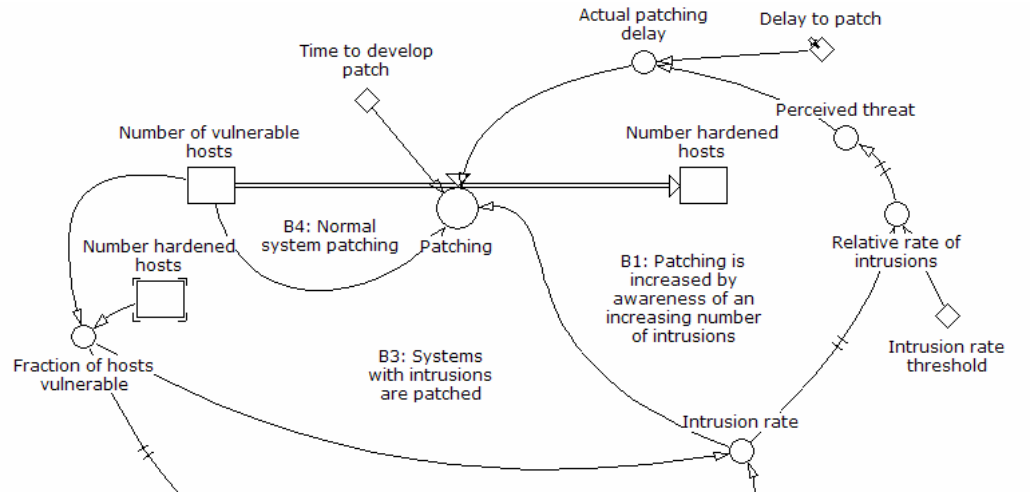

Figure 5: The defense sector of the model

As more and more systems are patched, the number of targets available will drop. This has an imminent effect on the incident rate, but with a delay, it also influences the rate of attack as hackers find fewer targets. In this way, the balancing loop "B2: Running out of targets" is closed. In addition, the diffusion of scripts will be reduced for much the same reason.

The Dynamics of the Model

As we have only made a generic model, the graphs presented in this section can only be considered notional as they are not based on actual data. As discussed in the

section with the reference modes, the data available has been too limited to quantify the model in such a way that it can represent a particular incident.

We will start to investigate the attack process. The number of hackers not using script will gradually start growing as more and more sophisticated hackers try to exploit the vulnerability. However, this does not happen immediately as they need to become aware of the vulnerability through various information channels. After a certain time lag (in the model 1 month), at least one hacker will make a script and start using this. This creates the small dip in the graph shown below as the hacker enters another state. Still some hackers will try to exploit the vulnerability without scripts, and the number of such hackers continues to increase in a goal seeking manner.

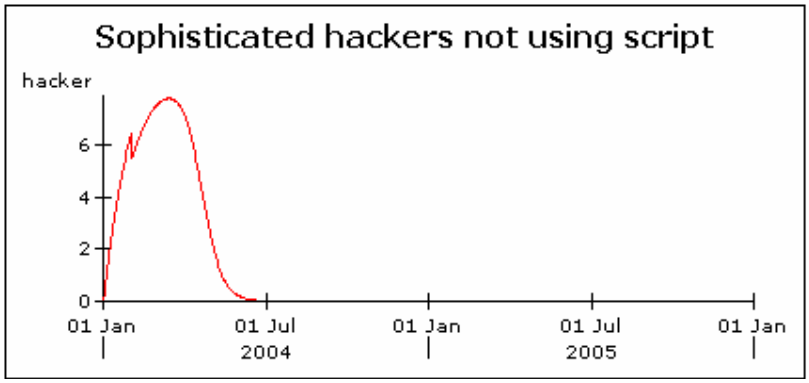

Figure 6: Sophisticated hackers not using script

However, more and more hackers will start using the script once it is made available. Most of those are rather less sophisticated hackers. Consequently, the reinforcing word of mouth loop among hackers starts dominating and the number of hackers in possession of a script starts growing exponentially.

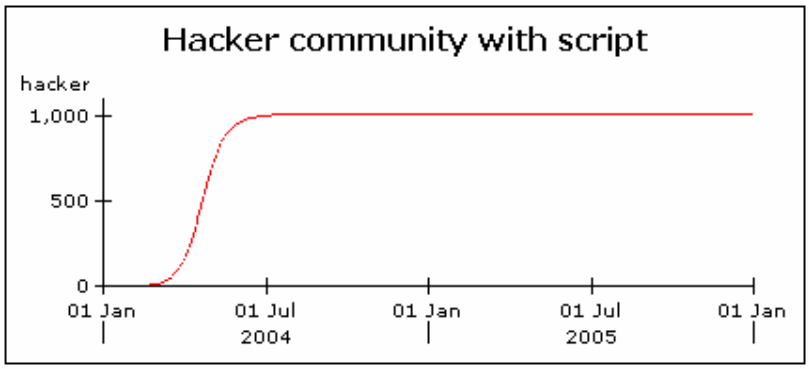

Figure 7: Hacker community with script

As most hackers become aware about the script, the word of mouth process levels off. The impact of this growth process can be seen in the incident rate and the rate of attacks.

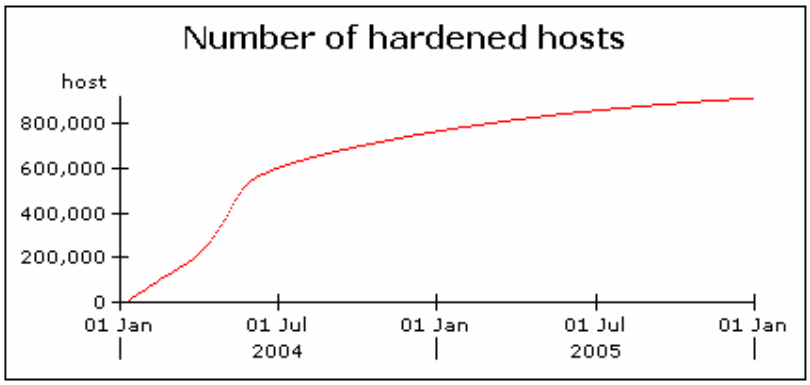

Figure 8: Hardened hosts

Initially, most attacks are successful intrusions but this increases the awareness about the vulnerability, and system administrators start patching the systems more quickly. Consequently, even though the rate of intrusions continues to rise due to a larger number of hackers with script, more and more attacks fail.

As there are less and less targets available for attack, the hackers become discouraged and their interest and capability for exploiting the vulnerability decreases. Consequently, the rate of attacks and intrusions start to fall.

We can trace this effect if we look at the development of hardened hosts as well. Once a patch is released, some system administrators will be quick to patch their systems, but the majority of the administrators will not react until they perceive the danger to be imminent. That is, when the number of attacks grows rapidly to high rate and gets a lot of publicity. Hence, we get the steep increase in the number of hardened hosts at the same time as the peak in attacks occurs.

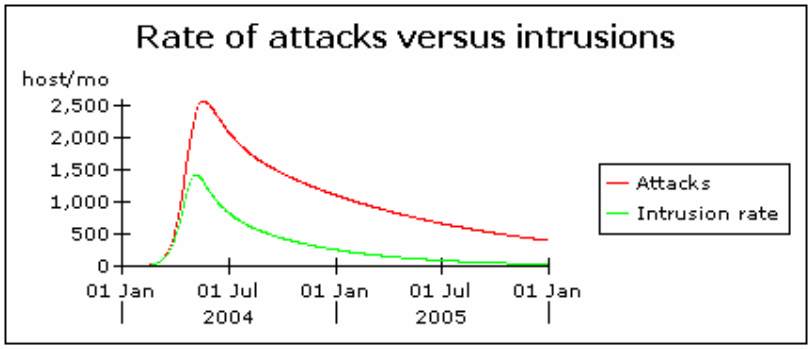

Figure 9: Rate of attacks and rate of intrusions

As the hackers start losing interest and the rate of attack and publicity around the vulnerability deceases, the threat is not perceived as high as before. Consequently, the rate of increase of the number of hardened hosts goes down and some systems may never be patched at all, even years after the vulnerability was discovered.

Policy Analysis

Even though we have a much aggregated generic model, we can experiment with simple scenarios. As indicated by the model and the information it is based on, an interesting finding is that the vast majority of intrusions occur after a correction is available. Correction only occurs after the threat is perceived to be high. That is, if the intrusion rate is high.

An interesting scenario would therefore be to run the model with a shorter average patching time. The assumption is that system administrators are much more aggressive to patch their systems even before the threat is perceived to be high. Consequently, we simply halved the delay time in order to illustrate such a change in policy. The result can be viewed in Figure 10.

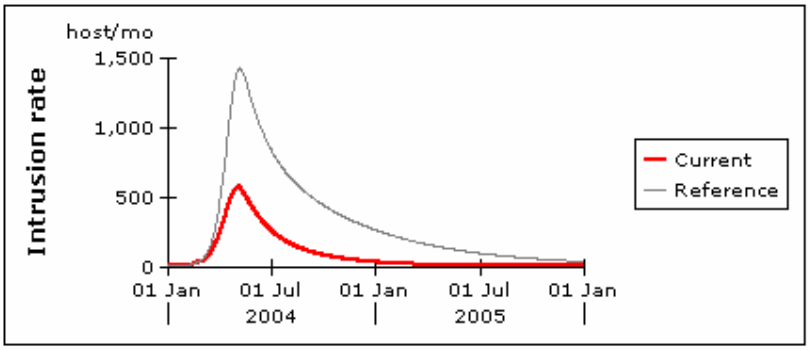

Figure 10: Result when the average time to patch is half of the base case. Current is the new scenario and Reference is the base case

The intrusion rate has the same shape, indicating that the same feedback processes dominate this scenario as well, but the peak is lower as there are fewer targets available due to correction.

These are just preliminary results, and further research and evidence is needed to make any final conclusions on the right approach. However, the model indicates that there is a high degree of leverage in the hands of system administrators. Thus, the model results support the same conclusion made by Arbaugh et al. (Arbaugh, Fithen, and McHugh 2000). However, history has shown that it is very difficult to influence system administrators in practice. How this process can be improved is beyond the scope of this paper.

Future Research

The attacker-defender motivations and scenarios described in this paper are plausible, and worthy of modeling, but eventually we want to validate (or disprove) the assumptions in the model with more empirical data. The model is currently at a preliminary stage and must go through more validation, not only through validation with more empirical data, but also through model interaction with experts and other types of validation typically used in system dynamics modeling.

Beyond improving the current model for the single vulnerability life cycle, the model boundary can also be expanded to incorporate new and important factors. Our current work has indicated that more research is needed in the following areas regarding vulnerability life cycle:

1. Multiple vulnerabilities
2. Tracking and tracing
3. Improve system administrators' correction policies.

Multiple Vulnerabilities:

As hacker tools very often combine exploits of different vulnerabilities it may be interesting to model the relationship between different vulnerabilities as well. As we saw in our simple model, the automation of an exploit creates the tremendous growth in exploits. Consequently, as single script can actually influence the life cycle of several vulnerabilities simultaneously.

Another reason for investigating multiple vulnerabilities is to investigate how exploits may compete for attention within hacker groups as well as among defenders.

Tracking and Tracing:

Even though the model indicated that there might be some leverage in a defensive patching approach, it has proven difficult in real life to influence system administrators as a group in this way. Lipson even argues that such an approach will never be enough, and that tracking and tracing will become increasingly vital to deter exploits (Lipson 2002, p. 11).

Improve System Administrators' Correction Policies:

In the model, we only made a very simple policy analysis of the patching process in the defense sector of the model. However, there are several types of corrections that can be made. Patching is just one of them, but simple corrections such as changes to firewall rules can also be used. In addition, more refinement in this part of the model can also give a better understanding as to why system administrators are so hard to influence as a group and what can be done to make them comply with security warnings in a better way.

One area of particular interest in this context is the interaction between Computer Security Incident Response Teams (CSIRTs) and their constituencies. CSIRTs can be helpful filters for information, raise awareness about threats, ease sharing of information between organizations, make recommendations for handling incidents and train system administrators to increase their level of security competence. Thus, they can influence how system administrators respond to new threats.

Conclusion

We have been able to put forward and test assumptions about why a vulnerability can be readily exploited even after a correction is available in a system dynamics based simulation model. The most important factors can be attributed to the reactive way system administrators typically respond to an incident. Instead of taking proactive

measures and correcting their vulnerable systems before an exploit tool starts to spread in the hacker community, administrators often wait until the rate of intrusion becomes high and exploits of the vulnerability are perceived as an imminent threat. This wait-and-see attitude can be very harmful as a large number of vulnerable targets might make exploitation of the particular vulnerability more popular in the hacker community.

Indeed, the model also indicated that a more proactive patching policy has a high degree of leverage. On the other hand, in practice, it has proven difficult to influence system administrators as a group in order to make them more willing to patch or correct their systems to harden them. Consequently, more research is needed to understand this problem. One possible way forward is to investigate how CSIRTs can play a role in this matter.

Other topics for future research include enhancing the current model to deal with multiple vulnerabilities in order to understand the composite effects and interactions among multiple vulnerability life cycles. In addition, research is needed to investigate and model more aggressive measures to reduce (e.g., deter) the exploits of vulnerabilities. For example, tracking and tracing of cyber attacks is one such measure to explore,

Even with the limited information resources available, the preliminary model presented in this paper has been able to give insights into the single vulnerability life cycle. This appears to indicate that system dynamics modeling can be a useful method in order for enhancing our understanding of complex dynamic computer security problems.

Appendix: Model Equations

```
mainmodel Component 1 {
aux Actual patching delay {
autotype Real
autounit mo
def 'Delay to patch'
*(1-'Perceived threat')
}
const Attack frequency with no tools {
autotype Real
autounit host/(mo*hacker)
init 1<<host/mo/hacker>>
}
const Attack frequency with tools {
autotype Real
autounit host/(wk*hacker)
init 1<<host/wk/hacker>>
}
aux Attacks {
autotype Real
unit host/mo
def ('attack frequency with tools'*'Hacker community with script'
+'attack frequency with no tools'*'Sophisticated hackers not using scripts')
*'Perceived availability of targets'
}
const Delay to patch {
autotype Real
autounit mo
init 12<<mo>>
}
const Distribution factor {
autotype Real
autounit %/wk
init 'Look up frequency'*'Probability of gaining script'
}
aux Fraction of hackers with no script {
autotype Real
def 'Hacker community with no script'/('Hacker community with no script'+'Hacker community with
script'+
'Sophisticated hackers not using scripts')
}
aux Fraction of hosts vulnerable {
autotype Real
unit %
def 'Number of vulnerable hosts'/('Number of vulnerable hosts'+'Number hardened hosts')
}l
evel Hacker community with no script {
autotype Real
autounit hacker
init 1000<<hacker>>
outflow { autodef 'knowledge transfer of scripting' }
}l
evel Hacker community with script {
autotype Real
autounit hacker
init 0<<hacker>>
inflow { autodef 'knowledge transfer of scripting' }
inflow { autodef 'sofisticated hackers gain access to script' }
inflow { autodef 'hacker develops script' }
}
aux Hacker develops script {
autotype Real
```

```
autounit Hacker
def IF(TIMEIS(STARTTIME+'Time to develop script from known
vulnerability'),1<<hacker>>,0<<hacker>>
)
zeroorder
}
aux Intrusion rate {
autotype Real
unit host/mo
def Attacks*'Fraction of hosts vulnerable'
}
const Intrusion rate threshold {
autotype Real
autounit host/mo
init 100<<host/mo>>
}
aux Knowledge transfer of scripting {
autotype Real
autounit hacker/wk
def 'Perceived availability of targets'*
'Hacker community with script'*'Distribution factor'*'Fraction of hackers with no script'
}
const Look up frequency {
autotype Real
autounit wk^-1
init 1<<1/wk>>
}l
evel Number of hardened hosts {
autotype Real
autounit host
init 0<<host>>
inflow { autodef patching }
}l
evel Number of vulnerable hosts {
autotype Real
autounit host
init 1000000<<host>>
outflow { autodef patching }
}
aux Patching {
autotype Real
autounit host/mo
def ('Number of vulnerable hosts'DIVZ0'Actual patching delay'
+'Intrusion rate')
*IF(TIME>STARTTIME+'Time to develop patch',1,0)
}
aux Perceived availability of targets {
autotype Real
autounit %
def DELAYINF('Fraction of hosts vulnerable',1<<mo>>)
}
aux Perceived threat {
autotype Real
unit %
def 1-DELAYINF(GRAPH('Relative rate of intrusions',0,0.2,{1,1,1,1,1,1,0.82,0.39,0.12,0.07,0.07//Min:-
1;
Max:1//}),1<<mo>>
)
}l
evel Potential sophisticated hackers {
autotype Real
autounit hacker
init 10<<hacker>>
outflow { autodef 'rate of new hackers trying to exploit vulnerability' }
```

```
}
const Probability of gaining script {
autotype Real
autounit %
init 70%
}
aux Rate of new hackers trying to exploit vulnerability {
autotype Real
autounit hacker/mo
def 'Potential sophisticated hackers'/'Time to manually test vulnerability'
}
aux Relative rate of intrusions {
autotype Real
def 'Intrusion rate'DIVZ0
MAX('Intrusion rate threshold',DELAYINF('Intrusion rate',1<<mo>>))
}
aux Sophisticated hackers gain access to script {
autotype Real
autounit hacker/wk
def 'Distribution factor'*'Hacker community with script'*
'Sophisticated hackers not using scripts'/
('Hacker community with no script'+'Hacker community with script'+'Sophisticated hackers not using
scripts')
}l
evel Sophisticated hackers not using scripts {
autotype Real
autounit hacker
init 0<<hacker>>
outflow { autodef 'sofisticated hackers gain access to script' }
outflow { autodef 'hacker develops script' }
inflow { autodef 'rate of new hackers trying to exploit vulnerability' }
}
const Time to develop patch {
autotype Real
autounit wk
init 1<<wk>>
}
const Time to develop script from known vulnerability {
autotype Real
autounit mo
init 1<<mo>>
}
const Time to manually test vulnerability {
autotype Real
autounit mo
init 1<<mo>>
}
}
unit hacker {
def ATOMIC
}
unit host {
def ATOMIC
}
unit incident {
def ATOMIC

}
```

References

Arbaugh, William A, William L Fithen, and John McHugh. 2000. Windows of Vulnerability: A Case Study Analysis. *Computer* 33 (12):52-59.

Forrester, Jay. 1994. Policies, Decisions and Information Sources for Modeling. In *Modeling for Learning Organizations*, edited by J. D. W. M. a. J. D. Sterman. Portland, Oregon: Productivity Press.

Hoglund, Greg and Gary McGraw. 2004. Exploiting Software: How to Break Code, Addison-Wesley, Boston.

Howard, John. 1997. An Analysis of Security Incidents on the Internet 1989 - 1995. Doctoral Thesis, Carnegie Mellon University, Pittsburgh. Available online at: http://www.cert.org/research/JHThesis/Start.html

Lipson, Howard F. 2002. Tracking and Tracing Cyber-Attacks: Technical Challenges and Global Policy Issues. Pittsburgh: CERT Coordination Center, Software Engineering Institute, Carnegie Mellon University. Available online at: http://www.cert.org/archive/pdf/02sr009.pdf

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

Back to the Top