THESIS

AUTOMATED SECURITY ANALYSIS OF THE HOME COMPUTER

Submitted by

Malgorzata Urbanska

Department of Computer Science

In partial fulfillment of the requirements

For the Degree of Master of Science

Colorado State University

Fort Collins, Colorado

Spring 2014

ABSTRACT

AUTOMATED SECURITY ANALYSIS OF THE HOME COMPUTER

Home computer users pose special challenges to the security of their machines. Often home computer users do not realize that their computer activities have repercussions on computer security. Frequently, they are not aware about their role in keeping their home computer secure. Therefore, security analysis solutions for a home computer must differ significantly from standard security solutions. In addition to considering the properties of a single system, the characteristics of a home user have to be deliberated. Attack Graphs (AGs) are models that have been widely used for security analysis. A Personalized Attack Graph (PAG) extends the traditional AGs for this purpose. It characterizes the interplay between vulnerabilities, user actions, attacker strategies, and system activities. Success of such security analysis depends on the level of detailed information used to build the PAG. Because the PAG can have hundreds of elements and manual analysis can be error-prone and tedious, automation of this process is an essential component in the security analysis for the home computer user. Automated security analysis, which applies the PAG, requires information about user behavior, attacker and system actions, and vulnerabilities that are present in the home computer. In this thesis, we expatiate on 1) modeling home user behavior in order to obtain user specific information, 2) analyzing vulnerability information resources to get the most detailed vulnerability descriptions, and 3) transforming vulnerability information into a format useful for automated construction of the PAG.

We propose the Bayesian User Action model that quantitatively represents the relationships between different user characteristics and provides the likelihood of a user taking a specific cyber related action. This model complements the PAG by delivering information about the home user. We demonstrate how different user behavior affects exploit likelihood in the PAG. We compare different vulnerability information sources in order to identify the best source for security analysis of the home computer. We calculate contextual similarity of the vulnerability descriptions to identify the same vulnerabilities from different vulnerability databases. We measure the similarity of vulnerability descriptions of the same vulnerability from multiple sources in order to identify any additional information that can be used to construct the PAG. We demonstrate a methodology of transforming a textual vulnerability description into a more structured format. We use Information Extraction (IE) techniques that are based on regular expression rules and dictionaries of keywords. We extract five types of information: infected software, attacker/user/system pre-condition, and post-conditions of exploiting vulnerabilities. We evaluate the performance of our IE system by measuring accuracy for each type of extracted information.

Experiments on influence of user profile on the PAG show that probability of exploits differ depending on user personality. Results also suggest that exploits are sensitive to user actions and probability of exploits can change depending on evidence configuration. The results of similarity analysis of vulnerability descriptions show that contextual similarity can be used to identify the same vulnerability across different vulnerability databases. The results also show that the syntactic similarity does not imply additional vulnerability information. Results from performance analysis of our IE system show that it works very well for the majority of vulnerability descriptions. The possible issues with extraction are mainly caused by: 1) challenging to express vulnerability descriptions by regular expressions and 2) lack of explicitly included information in vulnerability descriptions.

ACKNOWLEDGMENTS

I would like to thank Professor Indrajit Ray and Professor Adele Howe for giving me the opportunity to work with them as Graduate Research Assistant. I would like to thank my committee members: Professor Indrajit Ray, Professor Adele Howe, and Professor Zinta Byrne for their involvement and for reading of this thesis. I also would like to acknowledge all my coworkers especially Mark Roberts for valuable advices during the time of my study.

I would like to thank my husband, Łukasz Urbański, my family and my friends for their love and support. I also would like to thank Marko Denić for proof reading of this thesis.

This material is based upon work supported by the National Science Foundation under Grant No. 0905232. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation.

DEDICATION

This thesis is dedicated to the memories of my grandparents Henryka and Stanisław Pietrzak who passed away while I was in Graduate School.

To my mom who has twice won fight with cancer.

TABLE OF CONTENTS

Chapter 1

Home Computer User and Security

An estimated 90 million households in the U.S. had personal computers in 2011 [CEN11]. Home computer users are not only interested in searching for content, but also in doing shopping, banking, entertainment and more [Pew11]. They are a highly diverse group in terms of needs, wants, resources and abilities. The problem is that many home computer users do not fully understand the impact of their activities and actions on home computer security [AC03, BFP08, HY10]. As a result, they are considered as one of the weakest links in computer security [SBW01]. Home users are a lucrative target for attackers. According to a Symantec report from 2007 [Sym07], 95% of all targeted attacks were directed towards home computer users. The standard computer security tools which employ a one-size-fits-all issues paradigm are not tailored to each user's needs nor each user's perception of her/his role in maintaining security.

For instance, let us consider an example attack shown in Figure 1.1. The attacker wants to achieve root access privilege on a user's machine (red box). To succeed with this attempt she/he sends an email with a link to a phishing website to a home computer user. The home computer user uses a machine that is vulnerable (has some security holes - blue box). When the user opens the email and clicks on the link, the attacker can successfully exploit the user's system's vulnerability.

By preventing the user from clicking on the phishing link, we can greatly reduce the likelihood of the attacker achieving root access privilege. Anecdotal evidence suggests that different users have different likelihoods of clicking on such email links. We believe that with a user behavior profile we can estimate this likelihood. Modeling user behavior and incorporating it into security analysis for the home computer can allow us to be both proactive in handling security threats as well as be responsive to user security needs as appropriate.

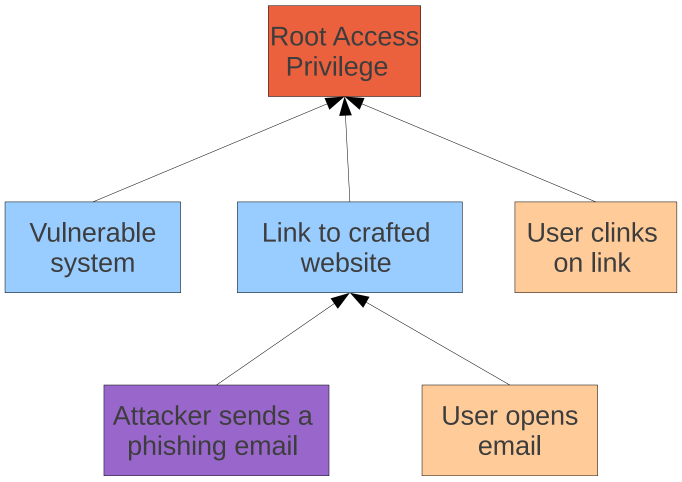


Figure 1.1: Example attack.

The security analysis of a home computer system is a process of evaluating system properties in order to identify weaknesses and improve security. System properties are, for example, system configuration, access privileges, relationship between system components, existence of vulnerabilities, etc. Since the user also influences home computer security, additional factors should be taken into consideration in security analysis of a home computer. These are user characteristics such as habits, preferences, and personality [AC03, Lea03, AC04].

Security analysis involves identifying attacks that can compromise a system. An attack on a system can be represented as a sequence of events that leads to the compromise. Such security scenarios are often modeled as graphs or trees, called Attack Graphs (AGs)/Attack Trees (ATs). An AT [DCH02, RP05, DPRW07, BEJS10] presents scenarios of possible attacks by enumerating the identified weaknesses of the system and capturing the relationships between vulnerabilities and other system properties as a conjunction-disjunction (And-Or) tree. AGs [SHJ+02, JSW02, APRS05] are data structures (analogous to ATs) that represent the different ways in which an attacker can exploit vulnerabilities to break into a system. ATs/AGs provide significant information for administrators to understand the threats and help them improve the security.

The Personalized Attack Graph (PAG) [RHR+11] model extends traditional AGs to focus on a single standalone system. It characterizes the interplay between vulnerabilities, user actions, attacker strategies, and system activities. The more specific information that is included in the PAG, the more accurate the security analysis can be. Consequently, a PAG can have hundreds of elements, and manual analysis can be error-prone and tedious. Moreover, the home computer user does not necessarily have the knowledge about security to perform such security analysis. Therefore, automation of this process is an essential component in security analysis for home computer users.

In order to construct the PAG we must have information about user behavior, attacker and system actions, and vulnerabilities that are present in the home computer. In this thesis we focus on: 1) modeling home user behavior in order to obtain user specific information, 2) analyzing vulnerability information resources to get the most detailed vulnerability descriptions, and 3) transforming vulnerability information into a format that is useful in automated construction of the PAG. In the following, we briefly describe the specific challenges with respect to these objectives.

1.1 Understanding the Home Computer User

Gaining information about the home computer user requires understanding her/his behavior. Different user characteristics can impact different vulnerabilities within the home computer system in a variety of ways. Relevant user characteristics need to be evaluated to tailor security measures for the home computer system. Different users can have different habits, behavior, and perceptions of a risk [HRR$^+$12]. These together constitute the user profile for the home computer system.

In order to model the home computer user, we have to identify which characteristics of the user should be considered. We have to understand how a computer user makes decisions and which factors influence these decisions. Another element that should be addressed is the relationships between different characteristics. For example, does a home computer user's self-efficacy depends on that user's computer knowledge and experiences?

For the purpose of security analysis that uses the PAG we need to define measurements of the likelihood of home computer user activities. Therefore, we are not only interested in identifying the relationships between the user characteristics but also their quantitative representation.

1.2 Vulnerability Information Resources

For the purpose of building the PAG we have to incorporate up-to-date information about vulnerabilities and their exploits. Each exploit has to be associated with user actions, attacker strategies, and system activities. Information about vulnerabilities can be obtained from multiple Vulnerability Databases (VDs). There is a variety of VDs; some of which are commercial and others are open-source. They provide information to users and security managers to help them learn about vulnerabilities, correct them, and not duplicate bugs.

Different VDs can contain different vulnerabilities which can be grouped into three categories: 1) shared for all, 2) unique for every, and 3) shared for some database. The first category groups all vulnerabilities that are common in all VDs. The second category gathers the vulnerability information that is unique for each VD. The last category groups vulnerability information that is present only in some VDs. Also, the vulnerability information can be expressed in a different way in each of the VDs that makes them incompatible with one another. Additionally, new vulnerabilities are posted at different times which makes some of them more up to date than others [Pol05]. Therefore, it is important to obtain vulnerability information from multiple sources.

The challenge is to decide which VDs should be used as a source for building the PAG. What factors should the decision be based on? Which characteristic of the VD should determine the choice? One characteristic that could be taken under consideration is Common Vulnerability and Exposure (CVE) names (or numbers) [CVE13] which are unique and standardized identifiers of publicly known security vulnerabilities. CVE numbers are also used across different VDs. Therefore, the usage of such identifiers can be very helpful in a process of comparing different VDs.

Another concern is the level of detailed vulnerability information which is provided by the different VDs. Constructing the PAG requires specific information about user actions, attacker strategies, and system activities. However, this information cannot always be found in a single vulnerability description. Therefore, we would like to consider, the possibility of adding any additional vulnerability information from different sources. However, redundant and insignificant information is undesirable. Therefore, we would like to check how similar are the vulnerability descriptions before combining them.

1.3 Transforming Vulnerability Description

Vulnerabilities are exploited via sequences of events (actions on the part of users and attackers); so to understand whether a vulnerability can be exploited for a given system, one needs to know the sequence. However, the standard resources for capturing vulnerability information, VDs, contain textual descriptions that do not necessarily describe the chain of events that can lead to an exploit. Moreover, the textual description is not useful in automated analysis. Hence, we would like to transform such a textual vulnerability description into a more structured one that clearly identifies the pre- and the post-conditions of the actions leading to the exploit. We would like to extract information such as: infected software, pre-conditions (user actions, attacker strategies, and system activities and configuration) and post-conditions of exploiting a particular vulnerability.

Information Extraction (IE) is a process of automatically extracting a small portion of information from an unstructured source. These small portions are: entities, relationships between them, and attributes that describe them. The entities are usually noun phrases that contain one or more tokens. The tokens are atomic parse elements from an unstructured text that are grouped into specified classes (e.g. names of people, organizations, locations, etc.).

The main challenges in the process of transforming vulnerability description in order to build the PAG are: 1) How to recognize interesting elements from the text? 2) How much information is enough? 3) How do we know if we are missing something? 4) What to do if necessary information is not in vulnerability description?

1.4 Thesis Contributions

In this thesis, we make three main contributions. First, we propose a conceptual model of home computer user behavior. Our model is significantly influenced by two prior models of human behavior in computer security: Ng, Kankanhalli and Xu's model [NKX09] and Claar's model [Cla11]. The conceptual model helps us to represent the relationships between different user's characteristics. In order to measure the likelihood of home computer user activities we translate the conceptual model into a probabilistic model - Bayesian network, that is called Bayesian User Action (BUA). In the BUA, the relationships between the user's characteristics from the conceptual model are quantified into probability values. The BUA provides the likelihood of user action which is used in automatic security analysis with the PAG.

Second, we analyze information collected from three different VDs (NVD, BugTraq, OS-VDB) over a period of time. We focus on measuring similarities between vulnerability descriptions from these VDs. We check whether using a similarity score we can identify the same vulnerability across different VDs. Subsequently, we measure the similarity score between vulnerability descriptions of the same vulnerability from different VDs in order to identify any additional vulnerability information.

Third, we propose a method of transforming vulnerability information into a format useful in automated construction of the PAG. Our IE method is based on manually created rules that are supported by regular expressions and dictionaries of keywords. We extract information such as infected software, attacker pre-conditions, user pre-conditions, system pre-conditions, and post-conditions of exploiting vulnerabilities. We evaluate our IE system by measuring its accuracy.

1.5 Thesis Organization

The rest of this thesis is organized as follows. Chapter 2 reviews the background of security analysis which uses AGs and presents the formal model of the PAG. Next, Chapter 3 discusses the home computer user model that provides the information about user behavior to the PAG. Comparison of vulnerability sources is provided in Chapter 4, followed by the description of the method of structuring vulnerability descriptions in Chapter 5. Finally, concluding remarks and future work are presented in Chapter 6.

Chapter 2

Background on Security Analysis Using Attack Graphs

Attack Graphs (AGs) [SHJ+02, JSW02, APRS05]/Attack Trees (ATs) [DCH02, RP05, DPRW07, BEJS10] are models used to analyze security risk in a system. Depending on the design, an AG/AT can answer questions about how an attack can happen or why the attack can happen. In other words, they present possible scenarios of an attack, or the causes and effects of actions. They capture all the possible ways in which a system can be attacked and compromised. They help to analyze the possible attack scenarios by showing the relationship between the vulnerabilities and system configurations.

For the sake of better understanding this paradigm, let us consider an example AT shown in Figure 2.1. The nodes in the AT represent different states of the system (possibly compromised states). These capture the potential subgoals of the attacker. The edges specify the relationship between the nodes. The root node describes the attacker's final goal of the attack, and the interior nodes represent sub-goals that lead to the attack. The leaf nodes represent initial states which exist in a system. To launch an attack the attacker has to exploit one or more of these leaf nodes. The attack paths are represented by the branches in the attack tree. Each transition from one state to another is modeled as a conjunction (AND) or disjunction (OR) of actions. For example, the nodes (or sub goals) F and G taken together represent a way in which the root node (goal) can be achieved.

2.1 Modeling Attack Graph as a Bayesian Network

The knowledge of possible attack paths is not enough in security analysis. It is not easy to predict which of the attack paths will be taken by an attacker. Therefore, researchers have studied quantitative techniques of security analysis. They have proposed techniques which

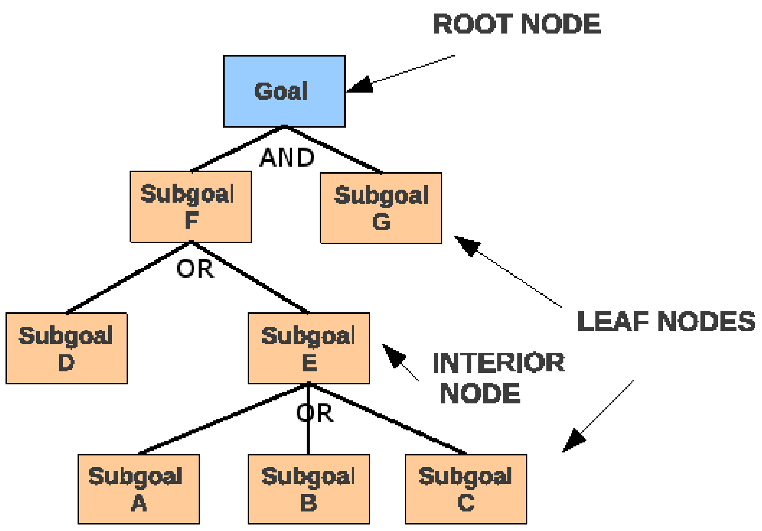


Figure 2.1: Example AT.

address the most likely attack paths and identify the weakest points [JSW02]. Researchers have also studied techniques to improve network security by measuring the quantity of security ensured by different network configurations [WNJ06, NJOJ03, DPRW07, PDR12]. One of the techniques is based on the Bayesian network (BN) probabilistic model which allows one to represent and reason about an uncertain domain [KN11, Kri01].

In the BN representation, nodes correspond to the set of random variables from the uncertain domain and arcs correspond to direct relationships between them. The variables can be discrete or continuous. Each node has associated values which the node can take. The relationship between directly connected nodes is represented as a probability value - conditional probability. The set of conditional probabilities for all nodes' relations is called "conditional probability distribution." The BN models can be used to reason about the domain using conditional probability distributions by taking values from observation nodes (or evidence) and recalculating the probabilities of any concerned nodes based on that evidence. The product of this recalculation is a posterior probability which is calculated using Bayes' Theorem [KN11, Kri01].

Let h be a hypothesis which we would like to check upon some evidence e which can be written $Pr(h|e)$. The $Pr(e)$ is the probability of the evidence being true, without regard for the outcome. The $Pr(h)$ is the probability of h prior to any evidence. The $Pr(e|h)$ is the probability of the evidence, given that hypothesis is true.

$$Pr(h|e) = \frac{Pr(e|h)Pr(h)}{Pr(e)}$$

In [FWSJ08, LM05, DKC09, PDR12], a BN is used to model the states of a network and encode the probabilistic property of the network vulnerabilities. Liu and Man [LM05] define the BN as a pair $(S, \mathbf{P})$ where S is a network configuration and $\mathbf{P}$ is a set of local probability distributions. Each node encodes a single compromised state, and each edge represents an exploitation of one or more vulnerabilities. Frigault et al. [FWSJ08] define an AG as a Dynamic Bayesian Network similar to [LM05]. They base the calculation of prior probability on the Common Vulnerability Scoring System (CVSS) [MSR07] metric. Poolsappasit et al.'s [PDR12] Bayesian Attack Graph (BAG) model is derived from the work by Dewri et al. [DPRW07] and Liu and Man [LM05]. The nodes in the BAG are attributes which represent "generic properties of system" such as vulnerabilities, system configuration, execution of operation, and access privileges.

2.2 Personalized Attack Graph (PAG)

A PAG[1] is built around a set of *exploit trees*. The exploit tree is similar to a BAG except that it includes computer user activities. An example PAG with six *exploit trees* resulting in a compromised system (yellow node) is shown in Figure 2.2. The terminal nodes of the PAG (red, blue, green) collect together known exploits and represent different possible security compromised states for a home computer system. The leaf nodes represent the initial state of the system, in Figure 2.2: 1) the software installed in the system (grey), 2) attacker strategies (violet), and 3) user activities that are involved in attack (orange). Layers in the graph indicate preconditions, but across the graph, the layers are otherwise insignificant. An arc in the graph is used to represent a state transition that contributes to a system compromise. The simplest transition is between two nodes (as in the box labeled

[1]Published first in [URR$^+$13]

B1). Conjunctive (AND) nodes (as in the box labeled B2) require all preconditions to be met for a state transition. Disjunctive (OR) branches (as in the box labeled B3) require only a single branch to be true.

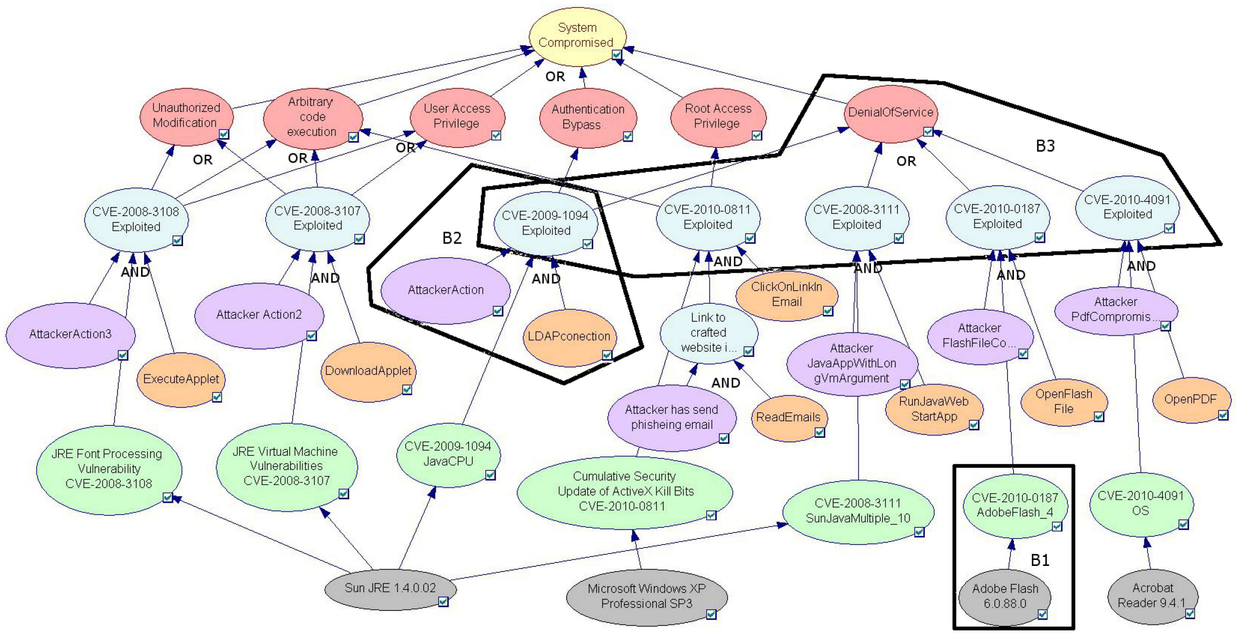


Figure 2.2: Example PAG.

2.2.1 Formal Model of the PAG

The formal model of the PAG is based on the AT presented in [DPRW07]. It is called "personalized" AG because it explicitly captures user, attacker and system actions, and tailors the representation to specific home computer systems.

Definition 1 *A* System Attribute Template *(SAT) is a generic property of a system that can contribute towards a system compromise. It can include, but is not limited to the following:*

- *system vulnerabilities as reported in different vulnerability databases,*
- *system configuration, e.g., data availability, use of security tools, open ports, un-patched software,*
- *access privileges, e.g., root account, guest account.*

In the bottom left of Figure 2.2, the "SunJRE 1.4.0.02" is an instance of a system configuration SAT, while its parent "CVE-2009-1094 Java CPU" is an instance of a system vulnerability SAT.

Only some instances of SATs are relevant for a specific system. A successful security compromise depends on which relevant instances of SATs are present or absent (that is true or false). Instantiating an attribute template with truth values on the specific instances allows us to implicitly capture the susceptibility of the system. We define a *System Attribute* with such a concept in mind.

Definition 2 *A* System Attribute*, s_i, is a Bernoulli random variable representing the state of an instance of a System Attribute Template. It is associated with a state – $True/1$ or $False/0$ – and a probability value, $Pr(s_i)$, indicating the probability of the state being $True/1$.*

For example, for the system in Figure 2.2, s_1="CVE-2009-1094 Java CPU" is a system attribute when associated with a truth value, signifying whether the specific vulnerability exists or not. $Pr(s_1)$ is the probability of the attribute being in state $True$.

Definition 3 *A* User Attribute Template (UAT) *is a generic property of a user that helps describe the influence of the user on home computer security. It is specified in terms of parameters that include but are not limited to:*

- *user system configuration choices, e.g., use of a specific browser,*
- *user habits or activities, e.g., checking email at specific intervals, clicking indiscriminately on links,*
- *a user's sensitive information (assets) that need to be protected.*

The User Attribute Template helps capture a user's impact on security much the same way as SAT helps capture the system characteristics. Thus, the UAT contains only those parameters that are relevant for securing the home system.

Definition 4 *A* User Attribute*, u_i, is a Bernoulli random variable representing the state of an instance of a User Attribute Template. It is associated with a state – $True/1$ or $False/0$ – and a probability value, $Pr(u_i)$, indicating the probability of the state being $True$.*

For example, Figure 2.2 shows that the Denial of Service attack can be achieved when the user opens a flash file with Adobe Flash version 6.0.88.0. Opening a flash file is an instance of the user habits UAT.

Definition 5 *An* Attack Attribute Template (AAT) *is a generic representation of the conditions set up by an attacker (in terms of actions that the attacker can/has taken) that lead to exploitation of a vulnerability and enable a successful attack.*
It includes, but is not limited to:

- *performing scanning of a system*
- *installing malicious software*
- *delivering specially crafted messages*

Referring to Figure 2.2, "Attacker pdf compromised" is an instance of delivering a specially crafted component AAT.

Definition 6 *An* Attack Attribute, a_i, *is a Bernoulli random variable representing the state of an instance of an Attack Attribute Template. It is associated with a state – $True/1$ or $False/0$ – and a probability value, $Pr(a_i)$, indicating the probability of the state being $True$.*

To analyze a system for potential compromise, we assume that all potential attacks are known. For a successful attack to take place, the corresponding attributes should have the value of true. If corresponding values are false (or are rendered false), an attack will not be successful. Consider, for example, the attacker attribute "Attacker pdf compromised" (extreme right side of PAG in Figure 2.2). If we expect that the attacker cannot ever successfully deliver a specially crafted pdf document, this attribute will be false. Thus, we can be assured that the exploit described by this scenario will never occur. Modeling these attributes as Bernoulli random variables allows us to compute the probability of a system being compromised.

Definition 7 Atomic Exploit: *Let S be a set of system attributes, U be a set of user attributes, and A be a set of attacker attributes. Let $X = S \cup U \cup A$. Let $s_j \in S, u_k \in U, a_l \in A$*

and $x_i = (s_j, u_k, a_l) \in X$*. Let* $\mathcal{F}$*, a conditional dependency between a pair of attributes in X, be defined as* $\mathcal{F} : X \times X \to [0, 1]$*. Let* $x_{pre}, x_{post} \in X$ *be two attributes. Then* $AtmExp : x_{pre} \to x_{post}$ *is called an atomic exploit iff*

1. $x_{pre} \neq x_{post}$*, and*
2. *if* $x_{post} = True$ *with probability* $\mathcal{F}(x_{pre}, x_{post}) > 0$*, then* $x_{pre} = True$

The attribute x_{pre} *is the pre-condition of the exploit denoted as* $pre(AtmExp)$ *and* x_{post} *the post condition denoted as* $post(AtmExp)$*.*

An atomic exploit allows an attribute x_{post} to be transformed from x_{pre} with some probability $\mathcal{F}(x_{pre}, x_{post})$. It is the simplest state transition that potentially leads to some security breach in the system. It can be visualized as a graph with two nodes x_{pre} and x_{post} with an arc from x_{pre} to x_{post} (box B1 in Figure 2.2).

Definition 8 Branch-Decomposed Exploit *In order to build more complex exploits, let* $BranchExp = \{x_{pre_1}, \ldots, x_{pre_k}, x_{post}\} \subseteq X$ *be a set of attributes such that if* $x_{post} = True$ *with some non-zero probability,*

1. $\forall i, x_{pre_i} = True$*, or*
2. $\exists i, x_{pre_i} = True$

then BranchExp is called a Branch-Decomposed Exploit*. Case (1) is called an* and- decomposition *and has the precondition:* $pre(BranchExp) = \{x_{pre_1}, \ \ldots, \ x_{pre_k}\}$*. Case (2) is called an* or-decomposition *and has the precondition:* $pre(BranchExp) = x_{pre_i}, \forall i = 1, \ldots k$*. The postcondition of both cases is:* $post(BranchExp) = x_{post}$*.*

A branch-decomposed exploit which is an and/or-decomposition is visually represented as a set of nodes $x_{pre_1}, \ldots, x_{pre_k}, x_{post}$ with arcs from x_{pre_i} to x_{post}. In Figure 2.2, an example of an or-decomposed branch exploit is the set of attributes enclosed by B3, while an and-decomposed exploit is the set of attributes enclosed by B2. We will call a set E of attributes an exploit, if either E is an atomic exploit or a branch-decomposed exploit.

Definition 9 Exploit Tree – *Let X be a set of attributes and E be either an atomic exploit or a branch-decomposed exploit. An* Exploit Tree *is a tuple $ET = \langle \epsilon_{root}, \mathcal{E}, \mathcal{P} \rangle$, where:*

1. *$\mathcal{E} = \{E_1, E_2, \ldots . E_n\}$ is a set of exploits defined over the set of attributes X.*
   - *$x \in X \leftrightarrow \exists E_i \mid x \in E_i$*
   - *If $x \in E_i, x \neq \epsilon_{root} | x = post(E_i)$ then $\exists E_j, j \neq i \mid x \in pre(E_j) \wedge \nexists E_k, k \neq j \neq i | x \in pre(E_k)$*
2. *$\epsilon_{root} \in X$ is a goal attribute that the attacker wants to be true such that $\nexists E_i \in \mathcal{E} \mid \epsilon_{root} \in pre(E_i)$*
3. *$\mathcal{P}$ is a set of estimated probability distributions. The elements of $\mathcal{P}$ are all the $Pr(x)$'s associated with attributes x's in ET.*

By the above definition, any proper subtree of an exploit tree is also an exploit tree. An exploit tree is characterized more by the goal attribute, ϵ_{root}, that the attacker wants to be true (as perceived by a security analyst), rather than the other attributes and the associated state transitions.

A home computer system may have only one exploit tree. However, more often than not, several goal attributes will be "of interest" to the attacker, requiring several exploit trees. Moreover, these exploit trees can be related to one another in the sense that rendering a goal attribute to be true in one tree leads to an attribute in another tree being true. To model this scenario, we introduce the notion of Personalized Attack Graph.

Definition 10 *A* Personalized Attack Graph *is a set of related exploit trees. It is represented by a tuple $PAG = \langle \mathcal{G}_1, \mathcal{G}_2, \mathcal{V}_1, \mathcal{V}_2 \rangle$, where:*

1. *$\mathcal{G}_1 = \mathcal{E}_p, \ldots, \mathcal{E}_q$ and $\mathcal{G}_2 = \mathcal{E}_m, \ldots, \mathcal{E}_n$ are* disjoint *sets of exploit trees such that $\mathcal{E}_i \in \mathcal{G}_1 \leftrightarrow \mathcal{E}_i \notin \mathcal{G}_2$.*
2. *Let $\mathcal{V}_1$ be the set of goal attributes of exploit trees in $\mathcal{G}_1$ and $\mathcal{V}_2$ the set of goal attributes in $\mathcal{G}_2$ such that $\mathcal{V}_1 \cup \mathcal{V}_2 = \mathcal{V}$, the set of all attributes in $\mathcal{G}_1$ and $\mathcal{G}_2$ and $\mathcal{V}_1 \cap \mathcal{V}_2 = \phi$. A goal attribute $v_i \in \mathcal{V}_1$ iff $\nexists x_k \in \mathcal{E}_d \in \mathcal{G}_2 \mid pre(x_k) = v_i$. A goal attribute $v_j \in \mathcal{V}_2$ iff $\exists x_l \in \mathcal{E}_b \in \mathcal{G}_1 \mid pre(x_l) = v_j$.*

Essentially, a PAG is a graph constructed out of the exploit trees, E_i's, present in a home system. The set of exploit trees is partitioned into two sets $\mathcal{G}_1$ and $\mathcal{G}_2$. The set $\mathcal{G}_1$ consists of all those exploit trees that have those goal attributes which are goals in themselves and do not lead to different attributes in other exploit trees being set to true; these goal attributes are not pre-conditions of any attribute of any exploit tree. These goal attributes are the *terminal nodes* of the PAG. The set $\mathcal{G}_2$, on the other hand, consists of all those exploit trees that have goal attributes that, if set to true, can lead to further attributes in other exploit trees to be set to true as well; these goal attributes are pre-conditions of some other attributes. To prevent cycles, we explicitly forbid the goal attributes in $\mathcal{V}$ to be pre-conditions of attributes of exploit trees in $\mathcal{G}_2$. A cycle in a PAG (if it was allowed to exist) would contain a sequence of goal attributes of the form $v_1, v_2, \ldots, v_n, v_1$ such that $v_1 \in pre(x_a) \in pre(x_b), \ldots, \in pre(v_2), \in pre(x_k) \ldots \in pre(v_3) \ldots \in pre(v_n) \ldots \in pre(v_1)$. By following this sequence the attacker sets to true what has already been set to true, and is essentially of no value to further risk analysis; this follows from the monotonicity property [AWK02].

2.3 Vulnerability Databases

The National Vulnerability Database (NVD) [NVD13] is maintained by National Institute of Standards and Technology Computer Security Division, Information Technology's Laboratory and sponsored by the Department of Homeland Security's National Cyber Security Division. The NVD combines the information from all government and some commercial vulnerability resources. It offers a comprehensive search capability, delivers vulnerability information statistics, and is updated hourly. The updates can be downloaded from its web page `http://nvd.nist.gov/download.cfm`

The NVD is based on Common Vulnerability and Exposure (CVE) names (or numbers) [CVE13]. CVE names are distinct identifiers of publicly known security vulnerabilities. A unique identifier is assigned to each vulnerability or exposure by the CVE Numbering Authority, which also posts them on the CVE website `http://cve.mitre.org`.

Open Source Vulnerability Database (OSVDB) [OSVDB13] is an autonomous and web-based vulnerability database. It was created in August 2002 at the Black Hat [Mos13a] and Defcon [Mos13b] conferences. It provides exact, detailed, recent, and objective technical information on security vulnerabilities. This project bridges the commercial and public institutions. The OSVDB feeds are updated every morning at 1:00am EST, and they include all stable data from the database. The feeds are available via an API in CSV or XML format. OSVDB is a relational database, in contrast to NVD's XML database. OSVDB is based on its unique vulnerability identifiers. It also includes CVE names.

BugTraq [Sec13] "is a full disclosure moderated mailing list for the *detailed* discussion and announcement of computer security vulnerabilities: what they are, how to exploit them, and how to fix them" [Sec13]. It was created in 1993 by Scott Chasin to address the issues of Internet security. In spite of vendor opposition it publishes full information about vulnerabilities as soon as it is identified. Since 1995 BugTraq has been a property of Security Focus and its vulnerability information is published on Security Focus website with its unique vulnerability identifiers. It also includes CVE names.

Chapter 3

Home User Security Risk Model

This chapter concentrates on modeling home computer user behavior for computer security. The goal is to obtain user specific information that is required to construct a Personalized Attack Graph (PAG) [URR+13] (see Section 2.2). As a reminder the PAG characterizes the software vulnerabilities and other attributes within a single system that can lead to exploits. The PAG captures the interplay between these vulnerabilities, user activities, attacker strategies, and system activities. The user information is represented in the PAG as user attributes. These attributes are Bernoulli random variables that represent the states of possible characteristics of the user (e.g., preferences, habits, assets). We model user attributes as user actions that describe user's characteristics. For instance, the user habit of using Internet Explorer is expressed as a user action "uses Internet Explorer" or user interest in using multimedia can be expressed as a user action "opens flash file."

Let us consider the example PAG shown in Figure 2.2. The orange nodes represent the possible user actions that are associated with exploiting particular threats. For example, on the right side the user action which is associated with exploiting vulnerability CVE-2010-4091 is "OpenPDF." The attack will be successful if the user opens crafted pdf file. The PAG requires several types of information about the user and leverages that information to identify the vulnerabilities that are most severe or likely for a specific home computer system. Thus, we would like to measure the likelihood of user activities. These probabilities are critical to determining what poses the strongest threats to a specific home computer system.

Modeling a user behavior is a difficult problem because of the complexity of human nature. We have to consider a lot of factors such as: how people understand their computer security, what their perception of the risk is, how much of their own security are they able to sacrifice to achieve some goals, and how they perceive threats. Based on available user behavior models [Con06, NKX09, CJ10, Cla11] we construct our home user model. Subsequently, we use this conceptual model as a framework to our Bayesian User Action model, which helps us measure the likelihoods of user actions. We perform three experiments to check how the home user model influences the PAG[1]. We also check the impact of the input accuracy of estimated prior probabilities from BUA on the user model.

3.1 Background

Over the past two decades, researchers have investigated user behavior in the scope of computer security, and proposed predictive models [Con06, NKX09, CJ10, Cla11]. The goal of these models is to predict computer security related behavior of users. The researchers have considered organizational users [NKX09] as well as home computer users [Con06, CJ10, Cla11]. They focus on the adoption of security technologies [Con06, NKX09, CJ10, Cla11] rather than modeling home computer user activities.

Conklin [Con06] bases his *Home PC Users Computer Security Behavior* model on Diffusion of Innovation Theory, which is used to describe the support of new ideas in response to an observed need of their usefulness. The application of security to a computer system is considered to be an adoption of an innovation. The security is applied to the computer because of a home user's intention.

Conklin's model (Figure 3.1) includes five elements:

- Adopter Decision Process - is the central element, which characterizes the process of decision-making. It influences the rest of the model components and is measured by the intention of using security software.

- Adopter Characteristics - specifies home user characteristics that influence user intention with regard to security.

[1]Part of the experiments results and evaluation published in [URR$^+$13]

- Characteristics of the Innovation - describes the importance of the specific innovation.
- Communications Channels - are the connections between the source of innovation (e.g., news, family, friends, vendors) and system administrator (home user).
- Social Consequences of Adoption - includes home user security experience.

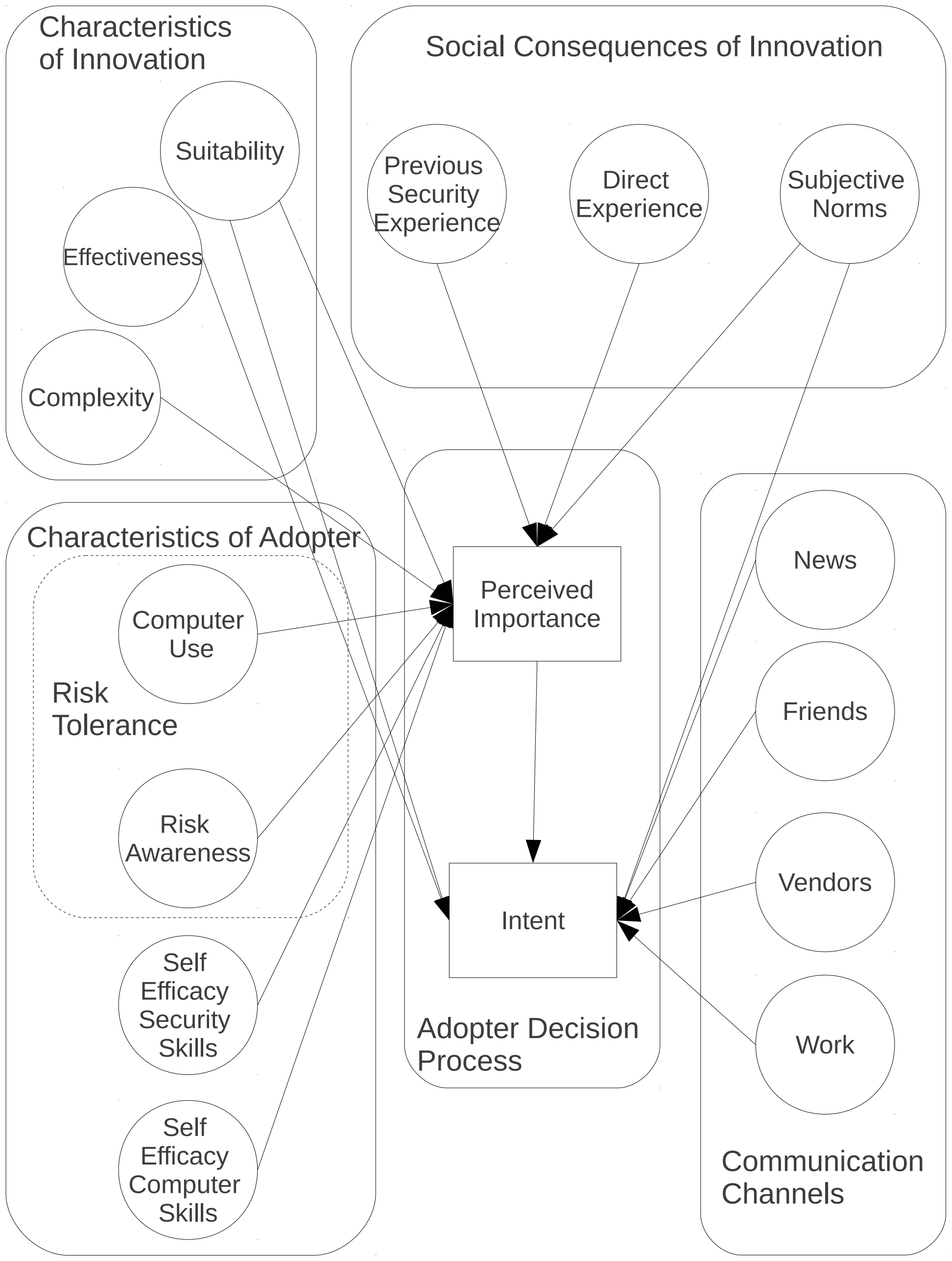


Figure 3.1: Conklin's research model [Con06] p. 57.

Claar [Cla11] extends the Health Belief Model (HBM) [Ros66, RSB88] to computer security. HBM is a psychological model that tries to explain and predict protective health behavior. It was proposed to address decision-making on health for long or short term diseases. This theory assumes that health-related actions that a person will take, depend on the person's beliefs about the following:

- person can avoid certain health disorders,
- person is susceptible to certain health problems,
- following specific advice would be useful in decreasing certain threats.

Figure 3.2 shows Claar's research model. He adopts the HBM from [RSB88] to include self-efficacy and demographic factors. These changes help to describe a person's confidence in performed actions and expresses the impact of demographic factors. Claar addresses the behavior cause of adoption and use of only three security technologies: anti-virus, firewall, and anti-spy-ware.

Ng's et al.'s [NKX09] approach also uses the HBM. Figure 3.3 illustrates the authors' research model. Ng et al. do not consider demographic factors. Also, in contrast to Claar's model they use a new component in their research model, *General security orientation*, which denotes a user's tendency and interest in regard to usage of computer security tools. As with the previous model, this is a qualitative model which is quantitatively tested using survey data. The authors have examined the user behavior related to computer security in organizations. They focus on adoption of technical security controls in organizations and on the level of user familiarity with computer security tools.

3.2 Conceptual Model of the Home Computer User

Our conceptual model in Figure 3.4 has been significantly influenced by HBM [RSB88] and two prior models of human behavior in computer security: Ng, Kankanhalli and Xu's [NKX09] and Claar's [Cla11]. Both models include six primary factors and one moderating factor which can predict a person's decisions about security. We use these primary

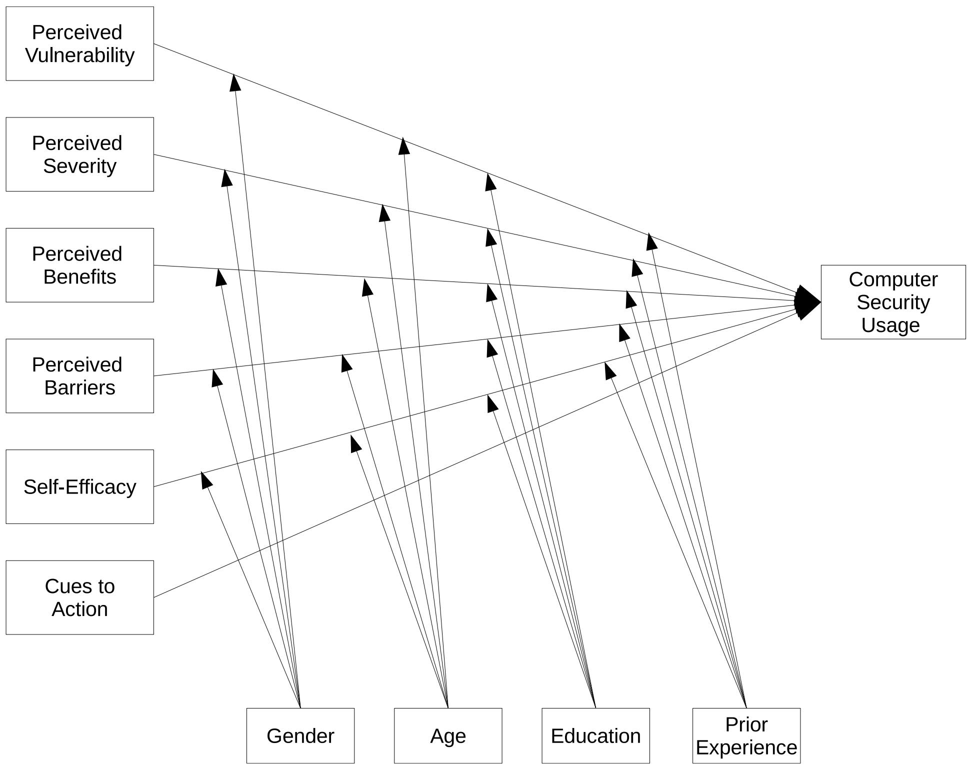


Figure 3.2: Claar's research model [Cla11] p. 49.

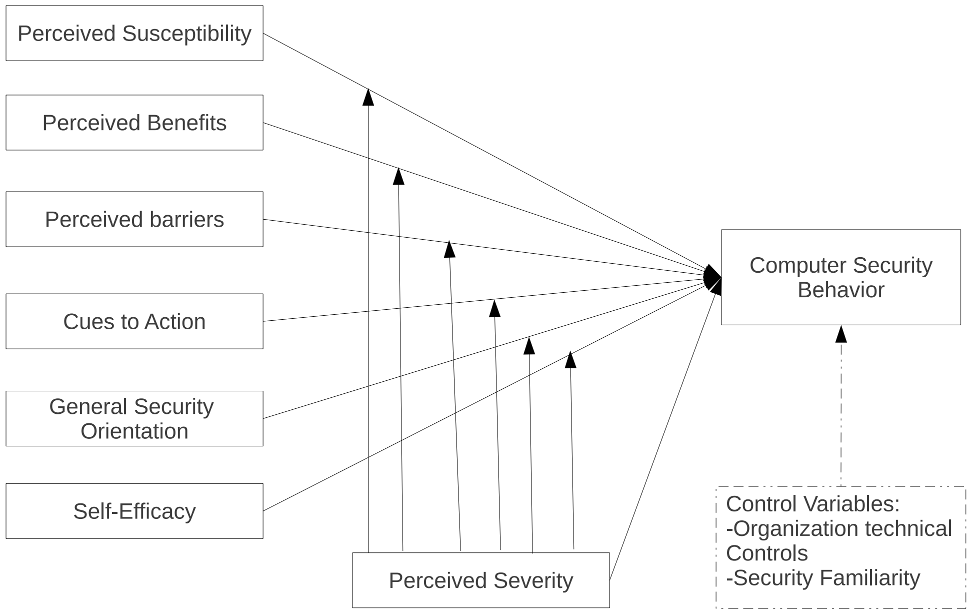


Figure 3.3: Ng, Kankanhalli and Xu's research model [NKX09].

factors (perceived susceptibility [NKX09]/vulnerability [Cla11], perceived severity, perceived benefits, perceived barriers, self-efficacy, and cues to action) and the moderating factor (computer security usage [Cla11]/behavior [NKX09]) in our model. The moderating factor is the output node which is called the *target node*. The six primary factors we use are:

- *Perceived severity*

  Originally HBM [JB84, Ros66] describes the perception of seriousness or severity of a disease. Claar [Cla11] and Ng, Kankanhalli and Xu [NKX09] define "Perceived severity" as a user perception of seriousness of a security attack on her/his computer. Ng et al. also consider possible connection of security incident on user's job and organization. We define it as beliefs in the seriousness of possible security violation from a specific activity and its impact on the user's home computer security.

  The lower the perceived severity about taking an action is, the greater is the motivation of taking it. It can be influenced by the user's knowledge or possible effect on user's home computer security.

- *Perceived benefits*

  According to HBM [JB84, Ros66], the perceived benefits variable refers to the perception of the effectiveness of adopting a predictive action to reduce the risk. Both Claar [Cla11] and Ng et al. [NKX09] define it as a user's belief in the benefit associated with the usage of security controls. In our research it captures a user's perception of effectiveness or benefit of adopting an action or a specific preference. The higher the perception of the benefits is, the greater is the motivation of taking certain action.

- *Perceived barriers*

  In relation to HBM, perceived barriers variable is related to the perception of the cost and/or difficulties in taking some health actions. Perceived barriers is considered as a cost benefit analysis performed by the person who evaluates the action's usefulness against perceptions that it may be costly, risky, uncomfortable, etc. Similarly to Claar [Cla11] and Ng et al. [NKX09], we define the perceived barriers as user's perception of cost or disadvantages associated with specific actions or preferences.

  The greater perception of the barriers is, the smaller is the motivation of taking particular action.

- *Risk tolerance*

  In original HBM [Ros66], this variable illustrates the susceptibility or personal risk. We define it as an individual's ability to handle or undertake different degrees of potentially harmful activities. It is intended to account for the result of studies that have shown that users are willing to accept risk if the potential benefit is viewed as more important [Pew11, GDG$^+$05]. Ng et al. [NKX09] call this factor, *Perceived susceptibility*, while Claar [Cla11] terms it, *Perceived vulnerability*.

  We believe that the term "Risk Tolerance" is more appropriate, keeping the nature of the factor in mind. It presents user beliefs about the likelihood of the occurrence of a security incident. We are considering it as general risk awareness according to particular user activities. It expresses general user personality (risk taking, harm avoidance, distrust, etc.).

  The greater the risk tolerance is, the higher is the inclination to take particular action.

- *Self-efficacy*

  According to Bandura [Ban77], self-efficacy describes a person's beliefs that the person is capable of doing something. It was added to the HBM by Rosenstock et al. [RSB88]. Both Ng et al. [NKX09] and Claar [Cla11] define self-efficacy as a "user self-confident in his/her skills or ability to practice computer security." In our research, "Self-efficacy" captures a user's belief that he or she is capable of taking specific action. It has been observed to be an important factor in several home user studies [AC03, AC04, BFP08, MLC09]. The greater self-efficacy is, the higher is the motivation for taking certain action.

- *Cues to action*

  As indicated by HBM [JB84, Ros66], cues to action are motivations to change the health behavior. Similarly to Claar [Cla11] and Ng et al. [NKX09], in our research the "Cues to action" demonstrate the user's motivation to cause a change in behavior. We consider cues such as media reports, friends, security software feedback, etc. Some studies [AC04, FBP07, SF09] have shown the importance of cues to action that encourage users to undertake certain activities. The greater the cues to action are, the higher is the motivation to take particular action.

Figure 3.4 illustrates our conceptual home user model with all factors and relationships between them. As in Claar's work [Cla11], our model includes demographic factors (*gender*, *age*, *socio-economic*, and *education*) as predictors of a user's decisions about security. Inclusion of these demographic factors is supported by other studies: Szewczyk et al.[SF09] (socio-economic factors) and [FHH$^+$02, MLC09, FBP07] (age and gender). Several studies [Was10, BWL$^+$12, FBP07] have also shown that a user's *prior experience* with computers and security may affect how the user perceives and acts on security threats. In order to get a more precise prediction, we divide *prior experience* into *good* and *bad experience* and include those as two other factors that can predict a user's decisions about security.

3.3 Predicting User Actions

In order to measure the likelihood of user actions, we use the conceptual home user model in Figure 3.4 as a schema to the probabilistic model. Because previous researchers have been successful in modeling users with a Bayesian network (BN) [BFS05, KSO$^+$01, ZA01], we choose BN as our probabilistic model. We build Bayesian User Action (BUA) model, shown in Figure 3.5, where according to the graph terminology: 12 user factors are the source nodes, the target node (*user computer action* (see Figure 3.4)) is a terminal node, and the interior nodes are children of pairs of user factors. The terminal node of the BN gives the probability for a given user's actions related to the user attributes in the PAG.

For every user attribute from the PAG that is relevant for a given user, we calculate its user action probability - BUA. A collection of these BUAs is called a *Bayesian User Profile* (BUP). Each of the BUAs from a particular BUP provides the posterior probability value for a specific user attribute in the PAG. To estimate the prior probabilities of the terminal nodes we rely on expert knowledge and literature.

In this section, we present a more detailed description of the BUA. After finishing the structure construction of the BUA, we quantify the relationships between connected nodes.

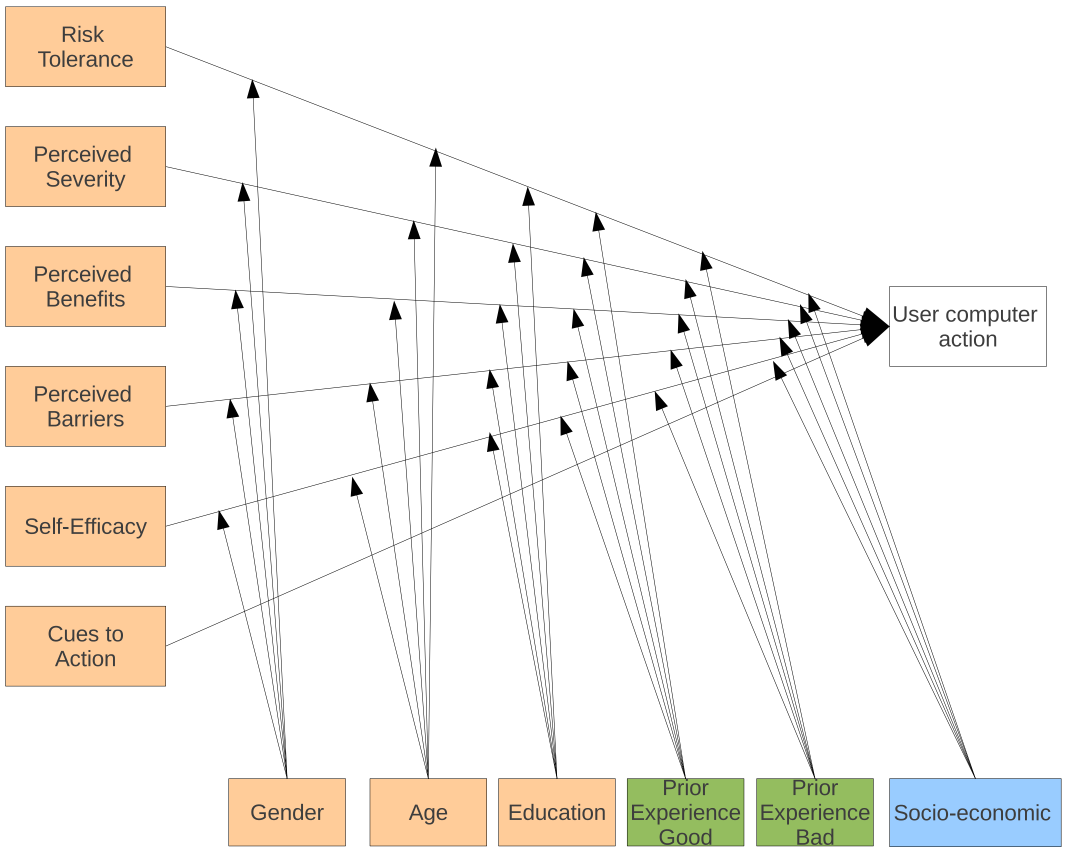


Figure 3.4: Conceptual Model of Home Computer User; orange - six primary factors from [NKX09, Cla11] and demographic factors from [Cla11]; green - similar to *Prior Experience* form [Cla11] and *General security orientation* from [NKX09]; blue - proposed by us as a new demographic factor.

In other words, we build the Conditional Probability Tables (CPT) for each node in the BUA. To estimate the prior probabilities for the interior nodes, we analyze the ways in which the demographics influence the primary factors, based on our own studies [BWL$^+$12], and the analysis of the literature. For instance, according to Szewczyk et al. [SF09], people with low annual income perceive a low possibility that they can become a target of attack. Therefore, we believe that the probability of taking some computer actions for those people, whose perceived severity of action is low and have low income, is higher than for people with higher socio-economic values. Table 3.1a presents a sample CPT for Perceived Severity (primary factor) and socio-economic (demographics factor).

The 12 user factors nodes in the BUA (squares on the left, right and top in Figure 3.5) are used to set evidence (e.g., Low/Medium/High, gender: M/F) for corresponding user factors.

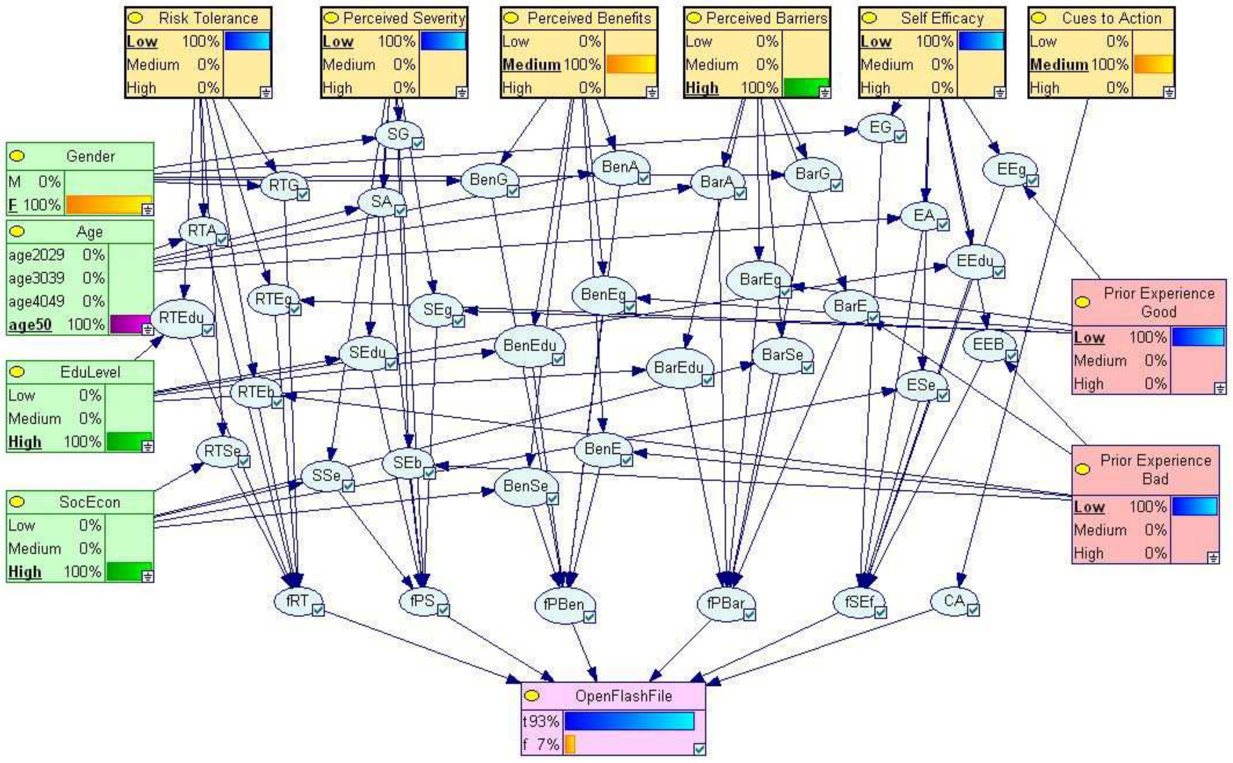


Figure 3.5: Instantiated Bayesian User Action for likelihood of OpenFlashFile for the hypothetical user; source nodes: 1) green - demographic factors, 2) red - two other factors, 3) yellow - independent variables; terminal node: pink - dependent variable (target node); Intermediate nodes: blue.

Table 3.1: CPT tables from BUA.

(a) CPT for Perceived Severity influenced by socio-economic (SecEcon in Figure 3.5).

Perceived Severity	socio-economic	T	F
low	low	0.92	0.08
low	medium	0.57	0.43
low	high	0.41	0.59
medium	low	0.75	0.25
medium	medium	0.38	0.62
medium	high	0.25	0.75
high	low	0.57	0.43
high	medium	0.18	0.82
high	high	0.04	0.96

(b) CPT tables for all Independent Variable Nodes

Independent Variable Values	value
LOW	0.06
MEDIUM	0.29
HIGH	0.65

The evidence set for demographic factors is static for particular user. The evidence set for the rest of the factors may change depending on user actions. We assume that the 12 user factors nodes are independent and the example CPT for these nodes is shown in Table 3.1b. Additionally, these CPTs are the same for all user factors.

3.4 Impact of Input Accuracy on User Model

In this section, we investigate effect of possible inaccurate probability estimation on calculated probabilities. The minor changes in user information should not significantly impact the target node probability because in the case of large inaccuracy of probability estimation the calculated target node probability is erroneous.

We define the accuracy as the precision to which the values of estimated probability from BUA agree with the reality. We set BUA settings for "OpenFlashFile" action for a synthetic user (Section 3.5.1). The baseline configuration is shown in Figure 3.6 with corresponding probability value of target node=0.929. This probability is used further as a baseline for comparison for the model sensitivity. We change one user information setting at a time and determine the probability from the target node. In this experiment we are interested in any changes that occur in probability of target node. Therefore, for analysis we use absolute the value of the difference between the baseline and new resulting probability.

Figure 3.7 shows the degree of changes within the probability of a target node for a subset of all possible combinations of user information settings. Most of the changes are less than 0.02, which is very small. Only "PerceivedSeverity" set to "High" produces changes greater than 0.03. Therefore, we can reason that minor changes to user information do not significantly impact target node probability. Consequently, the accuracy of predicted user actions is satisfactory.

3.5 Influence of Bayesian User Profile on PAG

In this section, we examine how different user profiles influence the probability of compromise of the computer system. Do substantial differences between users translate to different probabilities? We use the example PAG shown in Figure 3.8. First, we examine which exploits are most crucial for a specific user profile. Second, we analyze how the existence of

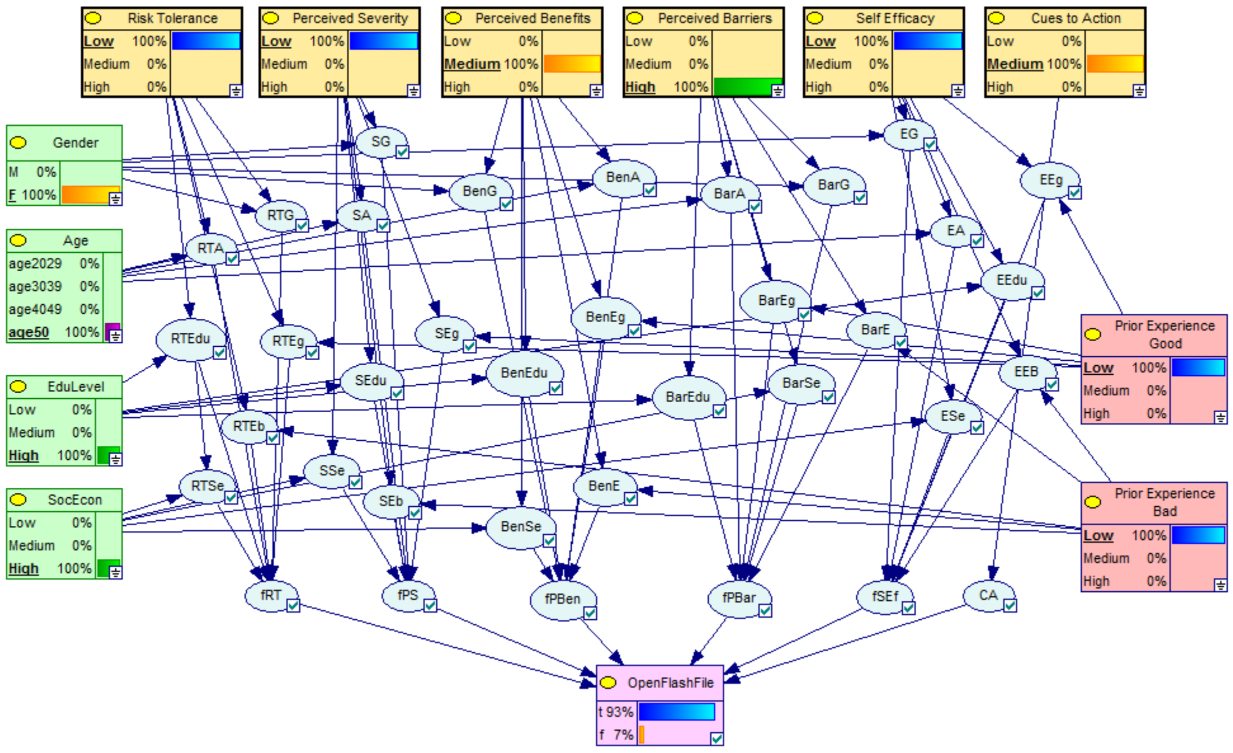


Figure 3.6: Baseline settings for action “OpenFlashFile” for synthetic user profile.

some user attributes in the PAG influence the final probability of the exploits. In the last experiment we study how the existence of different configurations of user/attacker/system attributes influences the probability of exploits.

For building and simulating each BUA we use the GeNIe2.0 [DSL13] (development environment for graphical decision-theoretic models) and SMILE (Structural Modeling, Inference, and Learning Engine) [DSL13]. The SMILE is a fully portable libraries of C++ classes implementing graphical decision-theoretic methods. The BUA is implemented in C++ and the example PAG is built using GeNIe.

3.5.1 User’s Profile

To explain how user profiles can lead to different BUPs with distinct target nodes values, we present three hypothetical users’ profiles. We set evidence of user’s factors nodes for those users accordingly to their profiles.

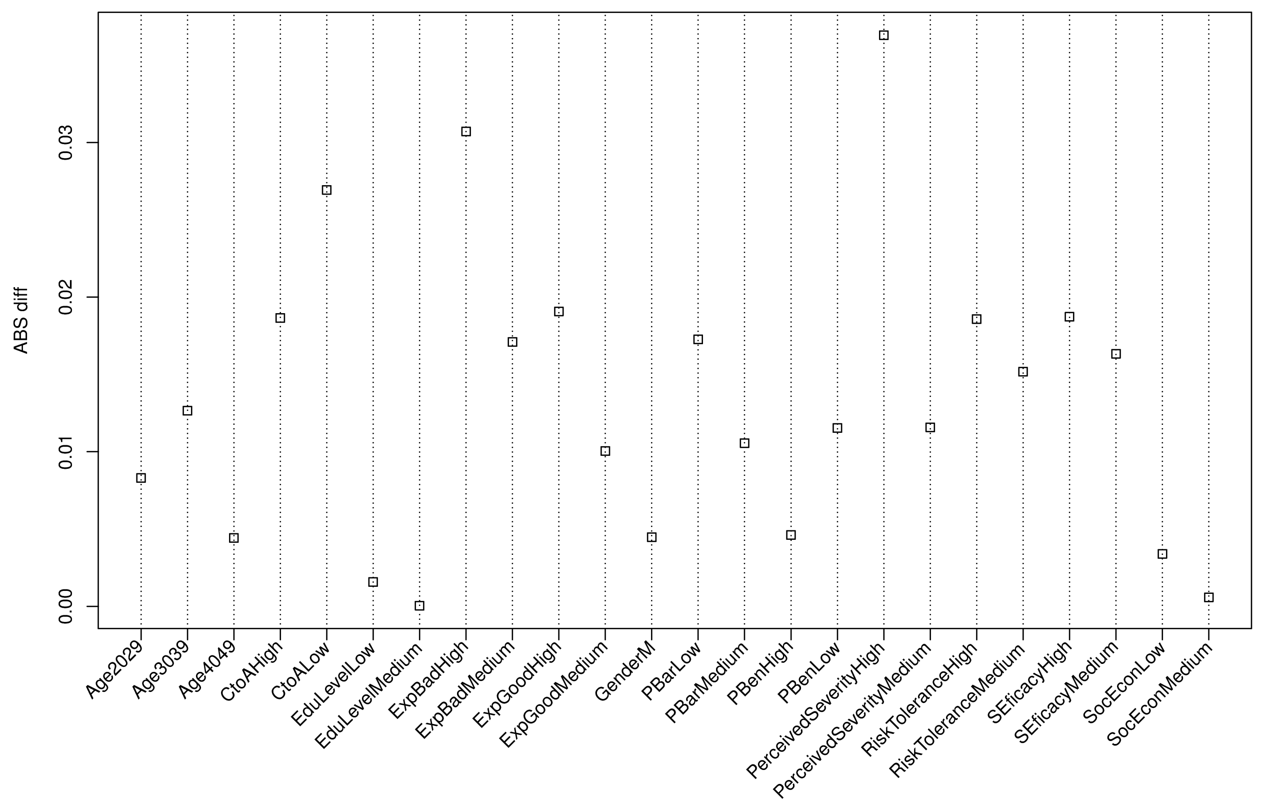


Figure 3.7: Results showing changes of target node probability depending on user information settings. On the X axis are user information settings. On Y axis (ABS diff) are the absolute values of the differences between baseline and probability of the target node with new user settings.

UserA is a retired person who was recently given a Windows XP machine that runs Internet explorer (IE). UserA is familiar with the inventory computer system from a recent job but is a new user of the Internet and email. UserB is a 20-year-old college student with a portable laptop running Windows7. UserB has used computers since kindergarten, and is very confident when using them. UserB automatically accepts any dialog that the browser displays. UserC is a 22-year-old college student who happens to be a computer science major. UserC is aware of security concerns and is diligent about installing updates and being observant of what her computer downloads.

As an example, Table 3.2 shows the input to the BUA for UserA and action "OpenFlash-File." The inputs to the BUAs for all users are shown in Appendix A. Table 3.3 shows the

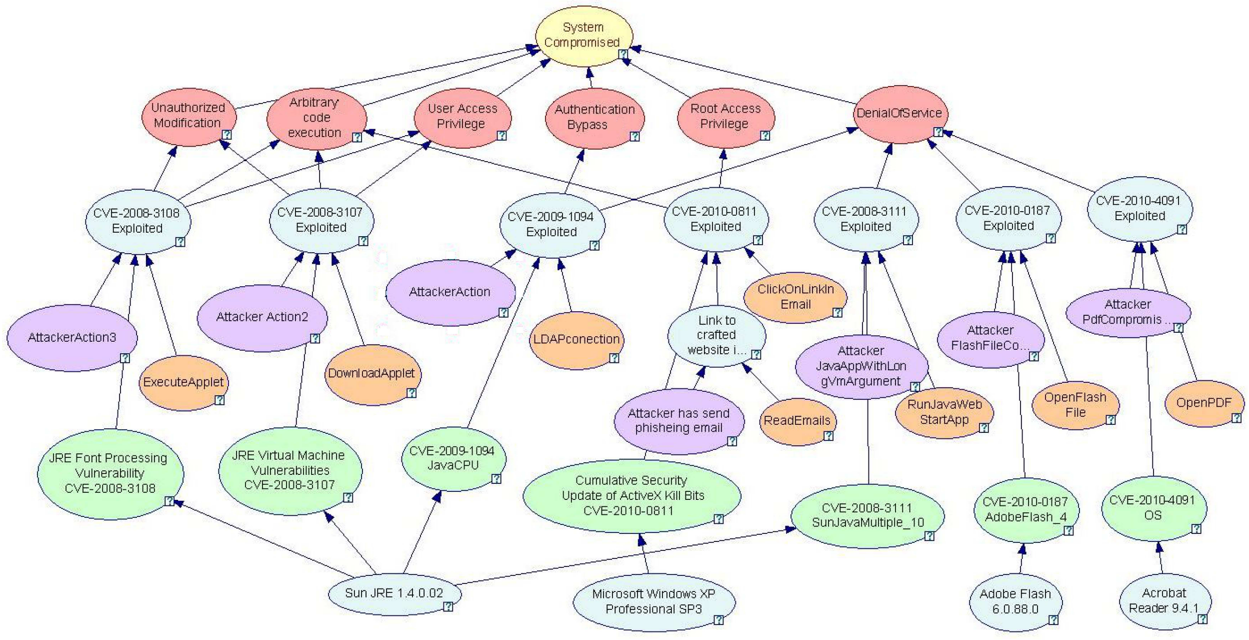


Figure 3.8: Example PAG.

Table 3.2: Input to the BUA for UserA and action "OpenFlashFile."

User Information	Settings
RiskTolerance	Low
PerceivedSeverity	Low
PBen	Medium
PBar	High
SEficacy	Low
CtoA	Medium
Gender	F
Age	age50
EduLevel	High
SocEcon	High
ExpGood	Low
ExpBad	Low

BUP for the three synthetic users and a set of the user attributes from the example PAG (Figure 3.8). The value 0.015 indicates that the user is unlikely to take this action. The rest of values are calculated using the BUA for each user attribute from the PAG.

Let us consider the example configuration of the BUA for hypothetical UserA for user action "OpensFlashFile." According to the description given above, we can assume that UserA is unlikely to use any social network or read PDF files. However, there is still a

Table 3.3: User attributes probabilities of engaging in specific activities.

User Attribute	UserA	UserB	UserC
ClickOnLinkInEmail	0.918919	0.963956	0.95791
OpenFlashFile	0.928524	0.972087	0.962467
DownloadApplet	0.913726	0.963662	0.963956
ExecuteApplet	0.949352	0.959211	0.94205
LDAPconection	0.015	0.978913	0.015
OpenPDF	0.015	0.974756	0.972216
RunJavaWebStartApp	0.867202	0.956325	0.015
ReadEmails	0.967239	0.974509	0.973062

nonzero probability that user can take these actions. For this reason, we assign probability equal to 0.015 to these attributes. For this example, let us further assume that UserA is a highly educated female at age of 60 with a very good socio-economic standing, but she has very low experience in Internet. She has a harm avoidance personality (Risk Tolerance on low level), and she perceives lower severity for taking this action because she does not know much about security concerns in Internet. This flash file also contains interesting information about an upcoming political event; the perceived benefit is at a medium level. But because she does not know much about the Internet, she is also not sure how to find and run this file (Perceived Barriers). In addition, she does not feel confident with using computer (Self Efficacy is at a low level). Nevertheless, because her good friend recommended that she opens the file, the Cues to Action are set at a medium level. Table 3.2 presents discussed evidence set configuration.

For each of the user action, similar scenarios are considered, and appropriate BUA configuration is assigned (see Appendix A). The effect of these characteristics is translated to probability values for the user actions by the corresponding BNs for the other values presented in Table 3.3.

3.5.2 PAG Configuration

Table 3.4 shows values used for building BN to represent the PAG. We based PAG configuration on information obtained from the NVD. For each of the PAG's vulnerabilities we used the metrics defined in NIST's Common Vulnerability Scoring System (CVSS) [MSR07]. The CVSS score is a number from 0 to 10, and consists of three metric groups: base, temporal, and environmental. The base metric group measures the basic characteristics of vulnerability. It includes two subscores: the exploitability (related exploit range (B_AV), attack complexity (B_AC), and level of authentication needed (B_AU)) and the impact (confidentiality, integrity, and availability impacts) [CVSScall13]. The temporal metric group measures a vulnerability change over the time. The environmental metric group measures the influence of vulnerability within an environment.

The probabilities of vulnerability existence $p(e)$ (e.g., PAG node "CVE-2009-1094 Java-CPU") are calculated according to expression given by Poolsappasit et al. [PDR12].

$$p(e) = 2 \times B\_AV \times B\_AC \times B\_AU \tag{3.1}$$

The Base Score is used to define the probabilities of exploiting vulnerability (e.g., PAG node "CVE-2008-3108 Exploited"). The Impact Subscore expresses the strength of the exploited threat (e.g., PAG node "Arbitrary code execution"). The Exploitability Subscore is used in attacker attributes to determine the level of attacker skills to exploit particular vulnerability. The score values are divided by 10 to obtain the probability range from 0 to 1. In order to use the probabilities in a BN, the probability 1 is represented as 0.99.

Table 3.4: Probability estimates of system attributes and attacker attributes from NVD's CVSS scores.

	p(e)	Base	Impact	Exploitability
CVE-2009-1094	0.99968	10	10	10
CVE-2010-4091	0.85888	9.3	10	8.6
CVE-2010-0187	0.85888	4.3	2.9	8.6
CVE-2008-3111	0.99968	10	10	10
CVE-2008-3107	0.99968	10	10	10
CVE-2008-3108	0.99968	10	10	10
CVE-2010-0811	0.85888	9.3	10	8.6

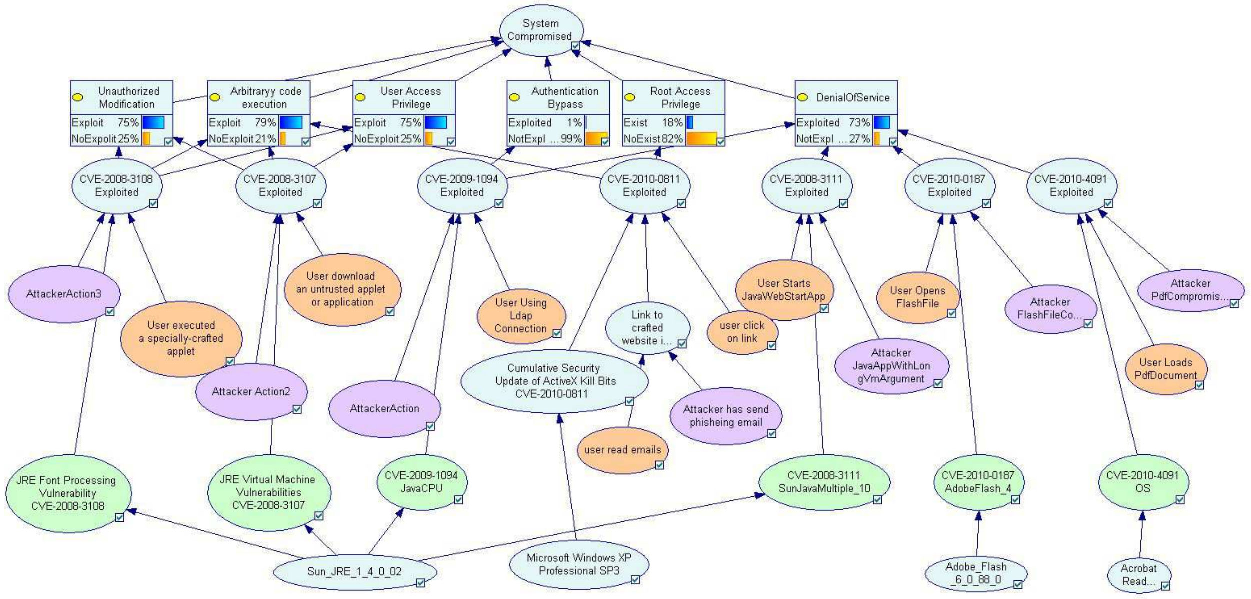


Figure 3.9: Instantiated PAG for UserA (Refer to Appendix B for the corresponding CPTs).

3.5.3 Experiment Results

Experiment 1 We begin by analyzing how different user profiles impact threats likelihood. We would like to determine which threats are most critical for a specific user profile. To perform the experiment, we apply the three user profiles and observe the posterior probabilities of the six exploits from the PAG (Figure 3.8 top red nodes and Figure 3.9 top square nodes). In this experiment, we do not set any evidence related to system or attacker attributes. Figure 3.9 illustrates the instantiated PAG for UserA.

The probability values for different users profiles that are collected from the top square nodes in Figure 3.9 are shown in Table 3.5. As can be seen, the probabilities of the threats do change based on the user profile. In this case, "Arbitrary Code Execution" is critical for all users because it has the highest probability value for all users. The "DenialOfService" threat has higher degree of changes between users' profiles due to the large number of user attributes in that branch in the PAG. In summary, different user profiles have distinct effect on threat likelihood.

Table 3.5: Posterior probabilities for six exploits for each user profile.

	Without user attributes	UserA	UserB	UserC
Unauthorized Modification	0.75	0.748	0.749	0.749
Arbitrary Code Execution	0.799	0.792	0.796	0.795
User Access Privilege	0.75	0.748	0.749	0.749
Authentication Bypass	0.489	0.007	0.479	0.007
Root Access Privilege	0.198	0.176	0.186	0.184
Denial of Service	0.843	0.726	0.84	0.685

Experiment 2 We next examine how changes in the presence of user attribute evidence impacts the final probability of the exploits. We manually set the evidence of specific nodes to *NotExist* or *Exist* as applicable. We then update the BN and read the value of the exploit nodes (Figure 3.9 top square nodes). We repeat those steps for all users. The *no-* prefix in some user attributes indicates the evidence set to *NotExist*.

Figure 3.10 shows the degree of changes in the probability of the exploits for synthetic users. We show only some user actions that do affect the probability of exploits. The exploit "Authentication Bypass," associated with user action "LDAPconection," is critical for all users. There is a significant increase in the probability of exploit for UserA and UserC for "LDAPconection" action with evidence set to *Exist*. For UserB, probability of exploit drops greatly when the evidence of "LDAPconection" action is set to *NotExist*.

UserA is also susceptible on "RunJavaWebStartApp," "ReadEmails," and "ClickOnLink-InEmail" for which the probabilities vary about 23% between *Exist* and *NotExist* evidence. The most crucial actions for UserB are "ReadEmails," "ClickOnLinkInEmail," and "ExecuteApplet noDownloadApplet" for which the probability differs about 20% between *Exist* and *NotExist* evidences. UserC is sensitive to actions "OpenPDF," "RunJavaWebStartApp," "ReadEmails," and "ClickOnLinkInEmail" for which the probability varies about 18% between *Exist* and *NotExist* evidences.

These results suggest that the exploits are sensitive to the user's actions, and the probability can jump dramatically if the user takes the worst-case actions. The results also suggest that having a clear understanding of the user's current actions or likely actions can contribute to selecting which action(s) are important to observe.

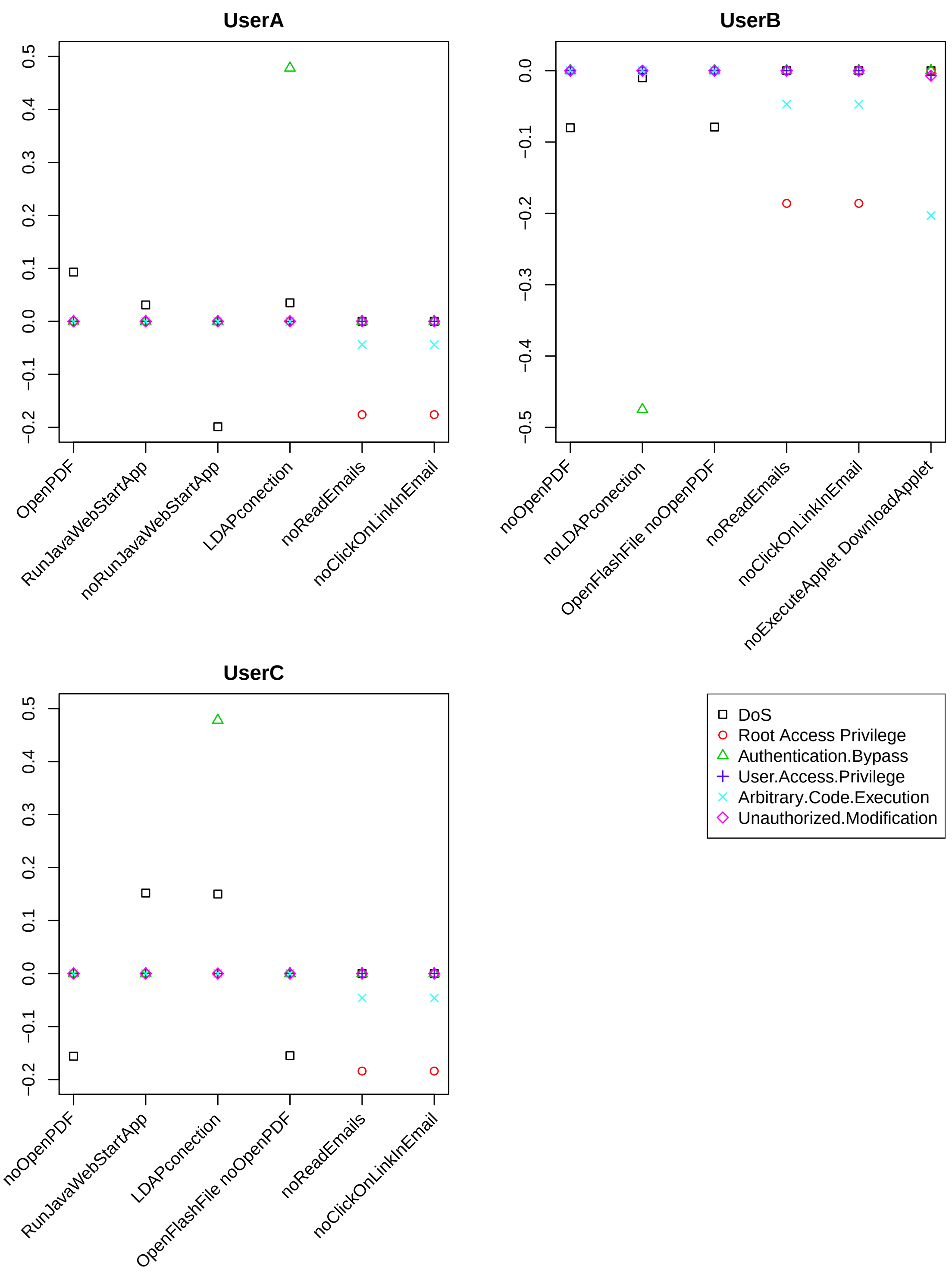


Figure 3.10: Results showing how evidence set at specific user nodes impacts the exploit likelihoods. On the X axes are the combinations of user attributes existence. On Y axes (diff) are the differences between exploit probability without any evidence and with evidence set to *Exist/NotExist* (prefix “no-” indicates *NotExist*). The 0 corresponds to the baseline for comparison.

Experiment 3 In the following experiment we would like to examine how the presence of different configurations of evidence for user/attacker/system attributes influences the probability of exploits. We set evidence only to the initial conditions in the PAG which lead to particular vulnerabilities. For example, in Figure 3.9 for branch *CVE-2010-4091 Exploited* we set evidence only to the nodes: "OpenPDF," "Attacker PdfCompromised," and "User Loads PdfDocument." First, we calculate the exploit probability without evidence set (baseline) then we measure the probability of exploits with some configuration of evidence set. We assess how much the exploit probability changes as a result. The "*no-*" prefix in some PAG attributes indicates the evidence set to *NotExist*.

Figure 3.11 shows the degree of changes in the probability of the exploits for a chosen subset of all possible combinations of node existence for all users. The most critical configuration contains "Sun JRE 1.4.0.02," "LDAPconection," and "RunJavaWebStartApp." However, for all users the probabilities of exploit can be decreased by: disabling vulnerable software (can be very expensive), applying updates if they are available, or simply preventing the user from taking these critical actions.

In order to understand how probabilities change with the same system configuration and attacker involvement exploits, depending on the user profile, let us take a closer look into "DoS" exploit. If UserA does not take "RunJavaWebStartApp" action, the "DoS" probability drops below baseline. Different situation is for UserB where in order to decrease the "DoS" probability not only "RunJavaWebStartApp," but also "LDAPconection" has to be prevented. For UserC "DoS" probability drops below zero when "ExecuteApplet" or "OpenPDF" action is set to *NotExist*.

These results suggest that the probability of exploits is sensitive to different configurations of evidence for user/attacker/system attributes. Depending on the different user profiles the probability of exploits can drop or increase compared to the baseline (without any evidence).

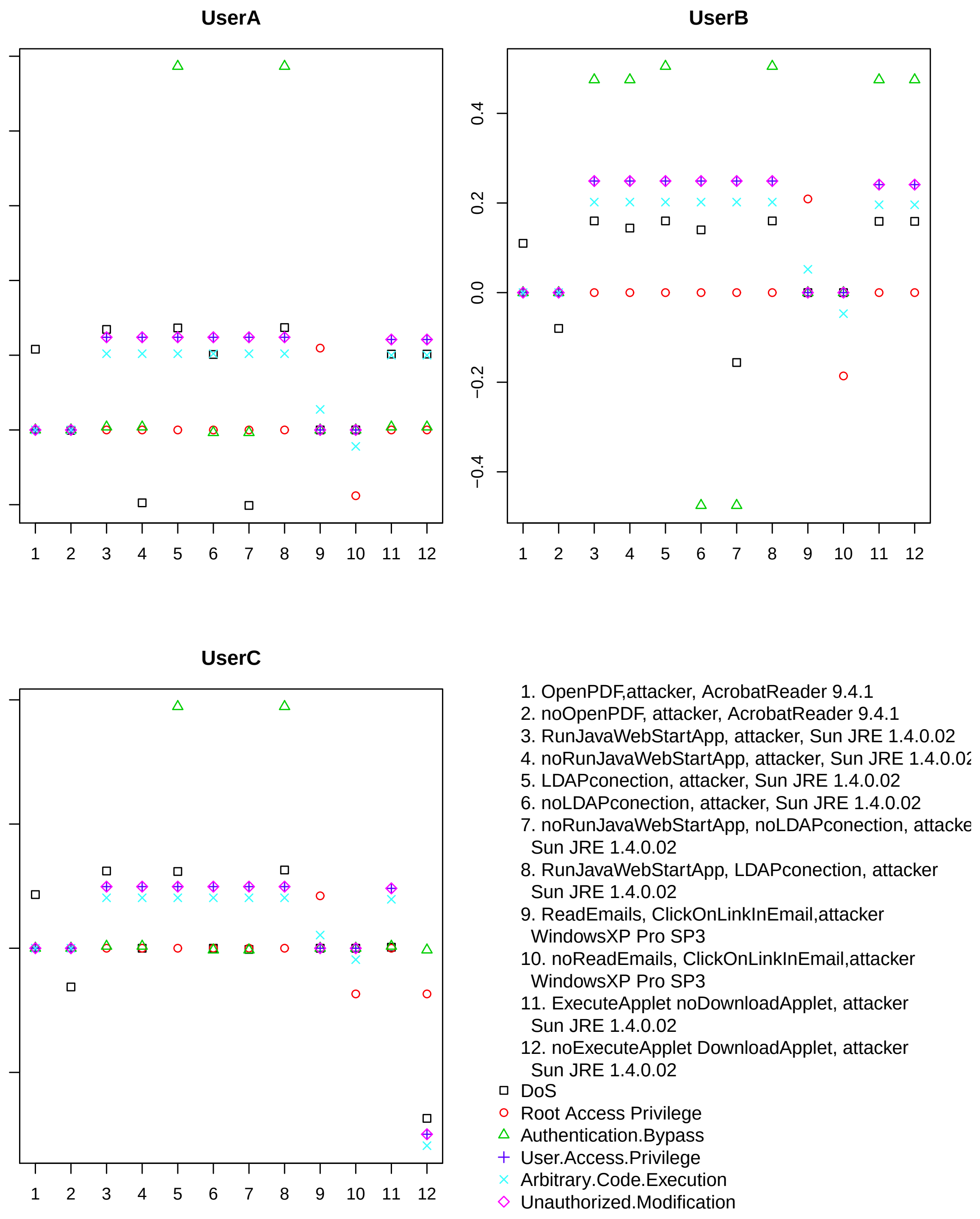


Figure 3.11: Results showing how evidence set at specific user/attacker/system nodes impacts the exploits likelihood. The combinations of node existence are on the X axes, each number corresponding to different combination. The differences between exploit probability without any evidence and with evidences set to *Exist/NotExist* are on the Y axes. The 0 corresponds to the baseline for comparison.

3.6 Discussion

The home computer users group consists of people of different ages, with different levels of experiences, personality, interest, etc. In addition, members of this group are vulnerable to security attacks when they are on the Internet because many of them are not knowledgeable about computer security issues. In this chapter, we have shown how to model home computer users to predict the likelihood of their computer activities. We have built a conceptual home user model and Bayesian User Action. The conceptual home user model has helped us to identify necessary components of human personality and characterize the relationship between them. The BUA has been used to obtain the likelihood of user actions. We have also evaluated the BUA in order to check accuracy of predictions. We have demonstrated that minor changes do not influence much probability of the target node.

Furthermore, we have developed a proof-of-concept to show how our approach works. We have shown that we can successfully measure the probability of user action depending on user personality. We have examined different evidence configurations to study their influence on the probability of exploits.

In future work, we are going to develop a method of automatic quantification of the relationships between connected nodes (building CPT). Moreover, the structure of the BUA should be rebuilt to be able to express user personality in more detail. We are planning to address these two by applying our ongoing psychology study on home computer user.

The last part is automation of the model. This should be addressed to streamline the process of modeling user and obtaining probability values which are used in the PAG. The user will be led through a series of questions, the answer to which will initialize specific configuration for a given user's settings. For example, user will be asked for age and gender to set up appropriate demographic factors. In order to set up the six primary factors (risk tolerance, perceived severity, perceived benefits, perceived barriers, self-efficacy, and cues to action) user will answer on specially constructed sets of questions from which the appropriate conclusions will be derived.

Chapter 4

Comparing Vulnerability Sources

Automated security analysis for the home computer requires up-to-date information of possible exploits of vulnerabilities. Each exploit has to be associated with some user, attacker and system attributes. This information can be collected from multiple sources (VDs) (see Section 2.3). The question is which VD should be used.

In this chapter we analyze information collected from different VDs. We gather information from the most known VDs - NVD, OSVDB, and BugTraq (see Section 2.3). Because they provide the information in diverse ways and formats, we use different methods to gather the information over a one week period.

In order to understand the magnitude of differences or similarities between different VDs, we perform text mining on collected datasets [SHHB00, HMFW12, Tie05, WIZD04]. For example, let us consider the descriptions of the same vulnerability from the studied VDs shown in Table 4.1. Each of the descriptions refers to CVE-2013-2503, but in different words, and provides different levels of information which can be used in further analysis of a vulnerability.

The knowledge about the level of similarities can help us build a more comprehensive dataset of vulnerability information. We can extend a vulnerability description with new information if the similarity score of two vulnerability descriptions indicates that the vulnerabilities are the same. For example, from Table 4.1 in BugTraq column, the following sentence can be found "The Privoxy user will then be prompted for a username and password that appears to originate from the Privoxy software" which is not present in the other two. This sentence gives additional information about this vulnerability required to construct the PAG.

Figure 4.1 shows an *exploit tree* (definition 9 Section 2.2.1) for the vulnerability CVE-2013-2503. As we can see, based on the NVD description we can obtain information about nodes 1,2, and 4. However, this is not enough to build an exploit tree. Therefore, we need additional information about this vulnerability which can be obtained from different VDs such as BugTraq (3 and 5) and OSVDB (6).

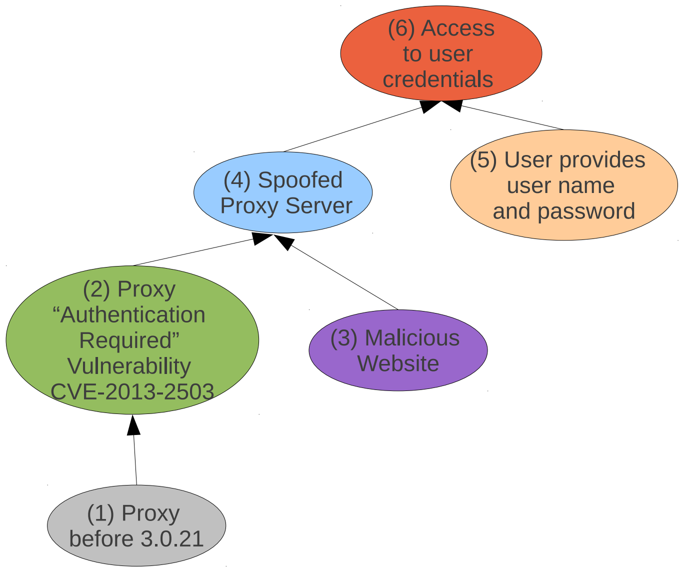

(6) Access to user credentials
(4) Spoofed Proxy Server
(5) User provides user name and password
(2) Proxy "Authentication Required" Vulnerability CVE-2013-2503
(3) Malicious Website
(1) Proxy before 3.0.21

Figure 4.1: Exploit tree from the PAG for CVE-2013-2503.

First, we analyze collected information from each of the VDs. This helps us to identify which VD provides the most extensive dataset. Omitting any known vulnerability information can lead to an incorrect security analysis.

Second, we measure similarities of collected documents. This helps us to recognize how vulnerability descriptions differ from different sources from each other. Collecting information about the same vulnerability from different sources can result in a more comprehensive description which includes user, attacker, and system attributes.

4.1 Background

The similarity of two documents can be measured as semantic or syntactic similarity. Semantic similarity defines how similar the concepts of two documents are, while syntactic similarity describes the similarity between two documents with respect to their structure.

Text documents can be represented as vectors using an algebra model called *vector space model* [SWY75, MRS08]. Each of the two documents is treated as a vector in n-dimensional

Table 4.1: The descriptions of the CVE-2013-2503 from the studied VDs.

NVD	OSVDB	BugTraq
Privoxy before 3.0.21 does not properly handle Proxy-Authenticate and Proxy-Authorization headers in the client-server data stream, which makes it easier for remote HTTP servers to spoof the intended proxy service via a 407 (aka Proxy Authentication Required) HTTP status code.	Privoxy contains a flaw that is triggered during the parsing of a request that contains no Proxy-Authentication header. The following response is parsed through a user's browser without any modifications, which may allow a remote attacker to spoof a Proxy-Authentication header and more easily gain access to user credentials.	During research into browser and proxy server handling of HTTP Response Codes, an issue with the way that Privoxy handles HTTP Response code 407 "Proxy Authentication Required" was discovered. Privoxy in versions 3.0.20 (and possibly prior) ignores the presence of "Proxy-Authenticate" and "Proxy-Authorization" headers and allows these values to be passed to and from a remote server without modification. The resulting behavior could allow a malicious websites to spoof a Proxy-Authentication response appearing to originate from the Privoxy service. The Privoxy user will then be prompted for a username and password that appears to originate from the Privoxy software.

space, where n is the number of unique terms (words or group of words) in the documents. The terms have weights that represent how often they occur in document. For example, Figure 4.2 shows 3-dimensional space of three unique terms: t_1, t_2, t_3 and two documents represented as vectors d_1 and d_2.

The vector representation of documents in n-dimensional space is called *term-document matrix*. In such a $M \times N$ matrix rows represent the M terms (dimensions) and columns denote the N documents. The term-document matrix can have thousands of rows (unique terms) and hundreds or thousands of columns (documents).

The distance between two document vectors is a measure of the similarity between them (Figure 4.2). One of the most common ways of calculating distance is the cosine of the angle

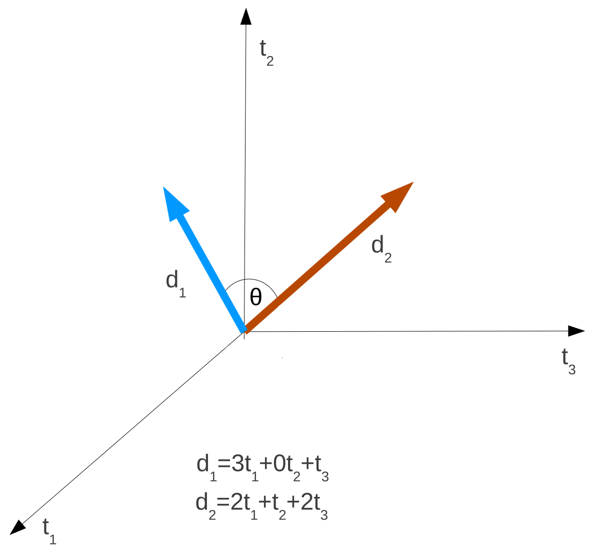


Figure 4.2: Vector documents representation in 3-dimensional space; d_1, d_2 - documents.

between these two vectors [SM86]. Equation 4.1 shows the formula. The inner product of the two vectors is divided by the product of their vector lengths. In practice this is calculated as a sum of the pairwise multiplied vectors' elements divided by the square roots of the sum of squares of the elements of each vector.

$$cos\theta = cos(\vec{d_1}, \vec{d_2}) = \frac{\vec{d_1} \cdot \vec{d_1}}{\|\vec{d_1}\| \ \|\vec{d_2}\|} = \frac{\sum_{i=1}^{n} d_{1i} d_{2i}}{\sqrt{\sum_{i=1}^{n} d_{1i}^2} * \sqrt{\sum_{i=1}^{n} d_{2i}^2}} \tag{4.1}$$

The cosine similarity value is measured between -1 and 1, where negative indicates dissimilarity and positive some levels of similarities. However, the cosine similarity measure for syntactic similarity is based on term frequencies and is a value between 0 and 1. This is because term frequencies can not be negative. Documents that do not share any terms have similarity equal to 0. The similarity value increases with the number of common terms.

Huang et al. [HTZJ10] use text mining techniques to classify vulnerability descriptions. They are interested in identifying vulnerability patterns, causes and system dependencies. However, during the classification process one vulnerability description can fit into more than one class - the taxonomy overlap. The authors' goal is to solve this issue. They represent the vulnerability description as a text vector and use the Term Frequency-Inverse Document Frequency (TF-IDF) method to calculate the text feature. The TF-IDF is a feature weighting method, where “TF” stands for term frequency in the text, and “IDF” indicates the term frequency across all studied documents.

Lin et al. [LJL13] propose a new method of similarity measurement. They measure similarity of documents based on the presence or absence of specified features. They also use the vector text representation and TF-IDF method to calculate the features of compared documents. We believe that this methodology can be very successful for vulnerability description similarity measurement.

However, using term frequency to calculate document similarity results in losing order among words in a sentence. Therefore, two sentences with the same set of words which are used in different context will be considered to be very similar. Moreover, the vector space representation fails to recognize the synonyms, which are represented in term vector as separate terms. Therefore, in order to measure the semantic similarity, some transformation of *term-document matrix* has to be performed in order to preserve the context of the documents. *Semantic vectors* are used to calculate the semantic similarity between documents. They are based on *word-space* model. In this representation the words and concepts are represented as points in mathematical space. The concepts of similar meaning are close to each other in that space.

Pandya et al. [PB05] demonstrate a methodology of calculating similarity based on semantics of the text. They represent the text as a set of concepts described in the text. They use “semantic relatedness” which identifies hidden relationship between the words. For example, “waiter” and “restaurant” are semantically related but not similar.

Atlam et al. [AFMA03] demonstrate another technique of measuring text similarity, field association similarity (FA-Sim) method. They consider a document as a sequence of fields. Their method is based on field association (FA) terms which describe the subjects of the text. The FA term can be composed of single or multiple terms occurring in document field and cannot be divided into smaller pieces without losing semantic meaning. The FA terms allow to measure similarity between documents fields without the need of comparing all information in these documents.

4.1.1 Tools

4.1.1.1 Apache Lucene

One of the most common full-featured text search engine libraries is Apache Lucene [Luc13] written in Java. It is an open source and cross-platform project. It provides indexing and searching functionality. The documents are represented as a sequence of fields. A field is a named sequence of terms and term is a sequence of bytes. The information about the terms is stored in an index. Lucene uses an inverted index, which means that it can list documents that contain a particular term. In *term-document matrix* notation, columns represent all fields of a particular instance of content and rows are all the instances of the content in the index.

The information flow in Lucene is shown on Figure 4.3. The documents are analyzed by Analyzer which is responsible for constructing terms. Lucene provides a set of built in analyzers (e.g., StandardAnalyzer, SimpleAnalyzer), but also provides the opportunity to construct customized analyzers. The Index Writer builds the index for documents. The index contains different statistics about terms (e.g., name of document fields in which term occurs). After constructing the Index (see Figure 4.3), searching for a specific query can be performed. The results are the similarity scores for matching documents.

4.1.1.2 SemanticVector

SemanticVector [Uni13, WF08] is an open source Java package built by University of Pittsburgh. It constructs the *semantic vectors* for terms (words) and documents. Figure 4.4 shows the information flow for the package. The SemanticVector is based on Lucene index which is a *term-document matrix*. It applies Random Projection (in Figure 4.4 Semantic Vector Index) in order to reduce dimensionality and to produce the semantic vectors [BM01, Sah05] (in Figure 4.4 termVectors, documentVectors). Random Projection assumes that randomly chosen high dimensional vectors are *nearly orthogonal* [WF08]. The reduction of orthogonal axes (term vectors) decreases dimensions from the original number [WF08, WC10]. As a result of the matrix transformation, the term vectors and docu-

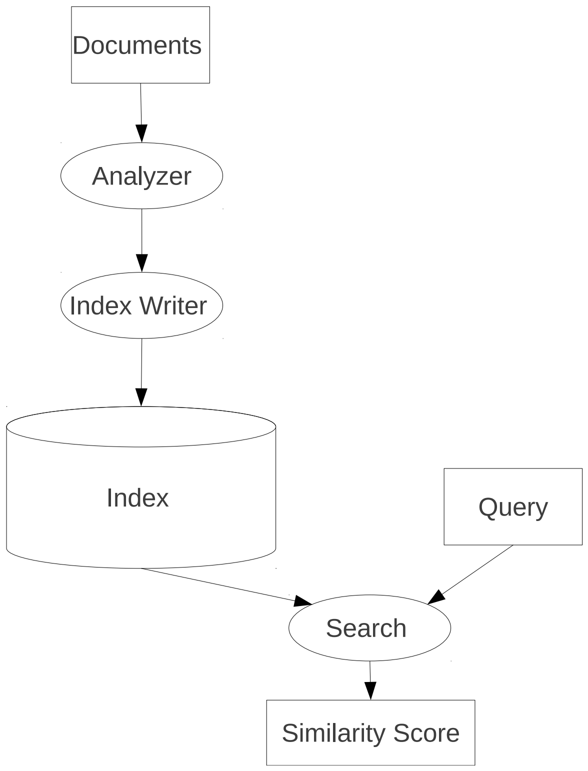


Figure 4.3: Information Flow in Lucene.

ment vectors are created. The document vector is a "weighted sum of the term vectors of their constituent terms"[WF08]. The Searcher searches for a specific query and produces the semantic similarity score. The score is calculated between two vectors based on cosine similarity formula.

4.2 Corpus Collection

In order to identify the source of vulnerability information, we collect vulnerability descriptions from three VDs: NVD, OSVDB and BugTraq. The data were collected during one week from March 11th to March 18th 2013. Only the newest available feeds were downloaded on a daily basis using different techniques. The data from NVD was obtained by using a shell script, OSVDB manually and BugTraq by subscribing to the mailing list. During that time the NVD was occasionally unavailable because it was recovering from a hacker attack. Therefore, all vulnerabilities information from the NVD was downloaded at once on March 18th which included vulnerabilities from the past week.

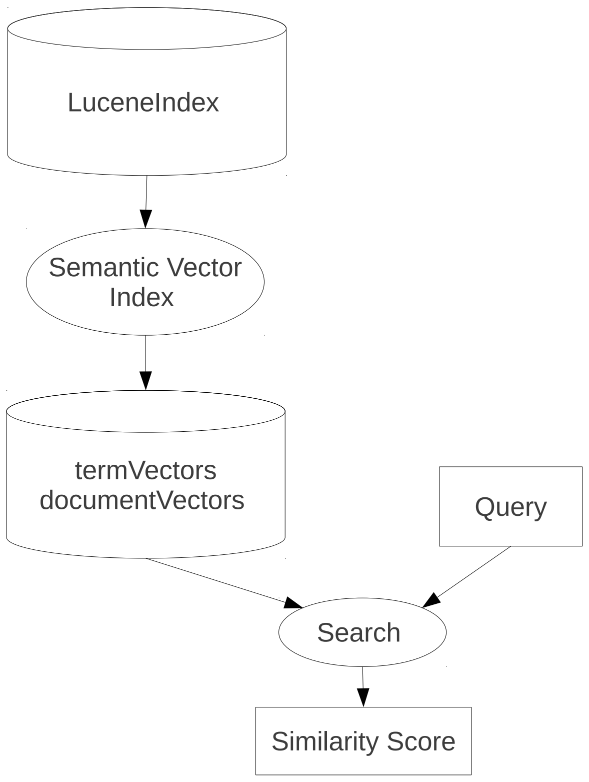


Figure 4.4: Information Flow in SemanticVector package.

The number of the collected vulnerabilities is a significant part of identifying which VD provides the most extensive dataset. We count the number of vulnerabilities collected from the different VD. We would like to determine which vulnerability information is newly published or updated.

Because of the different ways in which information is provided, we have to process it differently. Both NVD and OSVDB feeds are in XML format. Therefore, we apply an XML parser to extract the desirable information. However, they differ in database schema which results in two different parser approaches. Information from BugTraq is provided as plain-text; for this reason, we apply a customized scanner to work on text descriptions.

Table 4.2 presents summary statistics of our corpus. The “# of collected entries” indicates how many XML entries or plain-text documents have been collected. The “# of distinct vulnerabilities” gives the actual number of vulnerabilities. The “updates” column is everything from the “# of distinct vulnerabilities” which 1) for BugTraq refers to CVE numbers from the year 2012 or earlier, 2) for OSVDB and NVD is precisely specified as an update. The “# of new vulnerabilities” is everything from the “# of distinct vulnerabilities” that is not considered to be an update. The last column shows the statistics for the textual description of vulnerabilities for each VD. The length indicates the number of words in the description. All this information is collected via Java programs as well as manually. For example, to determine the updates in OSVDB, we build an XML parser which searches for tag “updated-on.” Note that BugTraq entries include multiple descriptions.

NVD provides the greatest number of distinct vulnerabilities (196). It not only includes the newly identified vulnerabilities but also provides updates for older entries. OSVDB also provides both new vulnerabilities and updates. However, the amount of provided information is much smaller than the NVD. BugTraq offers new vulnerabilities and few updates. Moreover, because vulnerability information is distributed by email list in BugTraq, it includes some information not related to vulnerabilities which we consider as noise. Also, BugTraq vulnerability descriptions have the greatest mean text length. NVD and OSVDB have similar mean lengths of textual vulnerability descriptions.

Table 4.2: Analysis of collected vulnerabilities.

VD	# of collected entries	# of distinct vulnerabilities	updates	# of new vulnerabilities	Lengths of textual description		
					Min	Max	Mean
NVD	196	196	37	159	19	89	38.96
OSVDB	50	50	44	6	26	73	47.08
BugTraq	37	80	15	65	56	2132	486.76

4.3 Similarity Measurement

We represent the vulnerability descriptions as text vectors. We have analyzed both semantic and syntactic similarity between two vulnerability descriptions which are calculated using cosine similarity measurement. We calculate semantic similarity in order to identify the same vulnerability from different VDs. Next, we calculate syntactic similarity in order to check does the score indicate addition vulnerability information.

Because NVD is based on CVE numbers, which are standardized and can be found in other VDs, we use the NVD set as a source of queries. The vulnerability descriptions from NVD are compared with each vulnerability description from the other VDs. Also, the CVE numbers help us with similarity score validation, which is computed in order to check the correctness of our result. In validation process, we only consider those VDs entries which have the CVE number matching to CVE number from the NVD set.

4.3.1 Syntactic Similarity

Algorithm 1 shows our similarity measurement procedure for syntactic similarity. The

Algorithm 1 The syntactic similarity measurement procedure.

```
Require: $QueriesSet = \{files\ with\ vulnerability\ descriptions\ from\ NVD\}$
    $VDSet = \{BugTraq, OSVDB\}$
    $AnalyzerSet = \{StandardAnalyzer, bi-gramAnalyzer\}$
Ensure: $cosineSimilarityScore$
 1: for each $Analyzer$ in $AnalyzerSet$ do
 2:   for each $query$ in $QueriesSet$ do
 3:     for each $VD$ in $VDSet$ do
 4:       for each $doc$ in $VD$ do
 5:         if $query$.CVE=$doc$.CVE then
 6:           $term\text{-}document\ matrix$ = indexer($doc$,$query$,$Analyzer$)
 7:           $TermFrequencyVectorDoc$=
              constructTermFrequencyVector($term\text{-}document\ matrix$($doc$))
 8:           $TermFrequencyVectorQuery$=
              constructTermFrequencyVector($term\text{-}document\ matrix$($query$))
 9:           $cosineSimilarityScore$=cosineSimilarity($TermFrequencyVectorDoc$,
              $TermFrequencyVectorQuery$)
10:         end if
11:       end for
12:     end for
13:   end for
14: end for
```

similarity score is calculated between a vulnerability description from the NVD set and a vulnerability description from the other VDs (OSVDB and BugTraq) for the same CVE number (line 5). In line 6 the Lucene index (or *term-document matrix*) for two documents (*query* and *doc*) is constructed. The Analyzer processes *documents* and constructs the terms.

In our implementation, we use Lucene's Standard Analyzer and our own made bi-gram Analyzer. We believe that using bi-grams can help us obtain more precise similarity scores. The same software can have multiple vulnerabilities, and its name will be repeated in each of the vulnerability descriptions. Therefore, it can lead to a deceptive similarity score.

Standard Analyzer first tokenizes the *documents* according to white space and punctuation. For example, let us consider the phrase "Unspecified vulnerability in HP System Insight Manager." It will be tokenized:
[Unspecified] [vulnerability] [in] [HP] [System] [Insight] [Manager].
Next, the Standard Analyzer applies the lower case filter and finally removes stop words. Therefore, the output of Standard Analyzer looks like this:
[unspecified] [vulnerability] [hp] [system] [insight] [manager].
Our bi-gram Analyzer performs the same steps as the Standard Analyzer, in addition to which it splits tokenized *documents* into shingles (contiguous subsequences of tokens) of size 2. Consequently, the output looks as follows:
[unspecified vulnerability][vulnerability in][in hp][hp system][system insight][insight manager].
For both analyzers, each element in square brackets is a single term which is later used to build the term frequency vector for each document (lines 7 and 8). Neither of the analyzers use a stemming filter. Finally, in line 9 the cosine similarity of the two term frequency vectors (*doc* and *query*) is calculated.

Cosine Similarity Calculation

Let as consider Standard Analyzer and the previous phrase with terms set:
[unspecified] [vulnerability] [hp] [system] [insight] [manager]
and query: "System vulnerability in HP system." which is transformed to the terms set:
[vulnerability][hp][system]
The term frequency vectors for *doc* and *query* are shown in the Table 4.3a Consequently the term frequency vectors are:

Table 4.3: Terms frequencies in doc and query.

(a) Standard Analyzer

Term	doc	query
unspecified	1	0
vulnerability	1	1
hp	1	1
system	1	2
insight	1	0
manager	1	0

(b) Bi-gram Analyzer

Term	doc	query
unspecified vulnerability	1	0
vulnerability in	1	1
in hp	1	1
hp system	1	1
system insight	1	0
insight manager	1	0
system vulnerability	0	1

TermFrequencyVectorDoc={1,1,1,1,1,1}

TermFrequencyVectorQuery={0,1,1,2,0,0}

Therefore, cosine similarity score is:

$$cosineSimilarityScore = \frac{1*0+1*1+1*1+1*2+1*0+1*0}{\sqrt{1^2+1^2+1^2+1^2+1^2+1^2}*\sqrt{0+1^2+1^2+2^2+0+0}} = 0.67$$

The same steps are performed with our bi-gram Analyzer but the documents are represented as bi-grams:

doc: [unspecified vulnerability][vulnerability in][in hp][hp system][system insight][insight manager]

and *query*: [system vulnerability] [vulnerability in][in hp] [hp system]

The term frequency vectors for *doc* and *query* are shown in the Table 4.3b. The cosine similarity score is:

$$cosineSimilarityScore = 0.61$$

4.3.2 Semantic Similarity

Algorithm 2 shows our similarity measurement procedure for semantic similarity. We use SemanticVector package [Uni13, WF08] to calculate semantic similarity between two documents. The similarity is calculated between each NVD vulnerability description (*query*) and vulnerability description from the other VDs (*doc*). First, (line 4) *term-document matrix* is constructed by the LuceneIndexer with StandardAnalyzer. Next, using the matrix factorization method (Random Projection) implemented in the SemanticVector the new $termVectors$

for each term and $documentVectors$ for each document are created (line 5). Finally, in line 6 the cosine similarity between two document vectors (*query* and *doc*) is calculated based on equation 4.1.

Algorithm 2 The Semantic Similarity Measurement Procedure.

```
Require: $QueriesSet = \{files\ with\ vulnerability\ descriptions\ from\ NVD\}$
    $VDSet = \{BugTraq, OSVDB\}$
    $StandardAnalyzer = \{text\ analyzer\}$
Ensure: cosineSimilarityScore
1: for each $query$ in $QueriesSet$ do
2:   for each $VD$ in $VDSet$ do
3:     for each $doc$ in $VD$ do
4:       $term\text{-}document\ matrix$=LuceneIndexer($doc, query, StandardAnalyzer$)
5:       [$termVectors, documentVectors$]= RandomProjection($term\text{-}document\ matrix$)
6:       $cosineSimilarityScore$ = ($documentVectors[query]$, $documentVectors[doc]$)
7:     end for
8:   end for
9: end for
```

Cosine Similarity Calculation

Let us consider again the two sentences from Section 4.3.1. The SemanticVector package creates, for each sentence, an appropriate *document vector*. The corresponding vectors of dimension 10[1] are:

documentVector[query]={0.18985608 0.17592032 0.3755809 -0.20466293 -0.12300006 -0.6336946 -0.21024674 0.31266034 0.26139787 0.35081604}

documentVector[doc]={0.268126 0.31391203 0.4206999 -0.022390557 0.19427659 -0.6342604 -0.20027597 0.122592606 0.12114932 0.3771361}

where the numbers are coordinates of the points in mathematical space.

Therefore, cosine similarity score is:

$$cosineSimilarityScore = \frac{0.891}{\sqrt{0.999} * \sqrt{0.999}} = 0.891$$

[1]Dimension 10 is chosen only for the purpose of explanation.

4.4 Similarity Score Analysis

The similarity calculation is implemented in JAVA using the Lucene 4.3.0 [Luc13], SemanticVector package [Uni13, WF08] and the Apache Commons Mathematics Library version 3 [Math13].

In Section 4.4.1, we calculate the semantic similarity between vulnerability description from NVD and vulnerability description from other VDs in order to identify the same vulnerability. Next, for both VDs we check whether there is any relationship between the maximum similarity score for each NVD vulnerability description and fact that the two compared documents refer to the same CVE number. Following, we validate the correctness of our semantic similarity calculation. Does the max similarity score indicate the matching documents?

Section 4.4.2 analyzes the magnitude of differences/similarities between VDs. We check whether there is any relationship between the syntactic similarity score and additional information from different vulnerability descriptions for the same CVE. We consider only those vulnerability descriptions from NVD that have their equivalent in the OSVDB/BugTraq set. We calculate the syntactic similarity between two vulnerability descriptions of the same vulnerability (the same CVE number) taken from NVD and other VDs. Subsequently, we run the test of association between the similarity score and additional information from VDs.

4.4.1 Identifying the Same Vulnerability Across Different VDs

Table 4.4 summarizes the results for max score of the semantic similarity measure between each vulnerability description from NVD vulnerability descriptions set and each vulnerability description from other VDs. OSVDB has a smaller score than BugTraq. Almost 70% of the scores for OSVDB are between 0 and 0.4, while for BugTraq about 73% of the scores are between 0.4 and 0.7. This indicates that vulnerability descriptions from BugTraq are more similar to NVD's descriptions. Therefore, we can assume that OSVDB provides vulnerability descriptions that are: 1) very different from NVD descriptions and/or 2) new vulnerabilities.

Let us now again consider the example vulnerability descriptions from Table 4.1. Based only on human judgment, the BugTraq description looks more similar to NVD. The ex-

Table 4.4: Summary of the similarity scores between NVD and OSVDB/BugTraq.

	Standard Analyzer	
interval of similarity s	OSVDB	BugTraq
0<s to s<0.3	29.08%	5.10%
s≥0.3 to s<0.4	40.82%	13.78%
s≥0.4 to s<0.5	22.45%	27.04%
s≥0.5 to s<0.6	4.59%	27.55%
s≥0.6 to s<0.7	1.53%	18.88%
s≥0.7 to s<0.8	1.02%	6.63%
s≥0.8 to s≤1	0.51%	1.02%

perimental results validate the human observations. The semantic similarity score for the BugTraq is 0.73, and for OSVDB 0.34.

4.4.1.1 Test of Association

In this section we check, for both VDs - OSVDB and BugTraq, whether there is any relationship between the maximum similarity score for each NVD vulnerability description and the fact that the two compared documents refer to the same CVE number. We run the statistical test for association - Fisher Exact Test [Fis22] for small sample size because the number of expected values that are smaller than 1 is more than 5%. Our hypothesis (H_A) is that there is an association between score and correct CVE identification. The null (H_0) hypothesis is that there is no association between max score and the same vulnerability identification. We reject H_0 if p-value $< \alpha = 0.05$. The contingency tables for both VDs are shown in Table 4.5. The "Same" column indicates the counts for two documents (NVD and studied VDs) corresponding to the same CVE number.

The results of the test are shown in Table 4.6. For both VDs we reject H_0. This means that there is an association between max similarity score and the fact that the two compared documents refer to the same CVE number. Using semantic similarity measurement thus appears to be appropriate for identifying same/new vulnerabilities.

Table 4.5: Contingency Table; Same-vulnerability is identified correctly; Not Same-vulnerability is not identified or identified incorrectly.

(a) OSVDB

Interval of Similarity s	Same	Not Same	Total
0<s to s<0.3	0	57	57
s≥0.3 to s<0.4	2	78	80
s≥0.4 to s<0.5	6	38	44
s≥0.5 to s<0.6	2	7	9
s≥0.6 to s<0.7	3	0	3
s≥0.7 to s<0.8	1	1	2
s≥0.8 to s≤1	1	0	1
Total	15	181	196

(b) BugTraq

Interval of Similarity s	Same	Not Same	Total
0<s to s<0.3	0	10	10
s≥0.3 to s<0.4	0	27	27
s≥0.4 to s<0.5	3	50	53
s≥0.5 to s<0.6	6	48	54
s≥0.6 to s<0.7	2	35	37
s≥0.7 to s<0.8	6	7	13
s≥0.8 to s≤1	2	0	2
Total	19	177	196

Table 4.6: Fisher's Exact Test for Count Data with α=0.05.

	p-value	Conclusion
OSVDB	5.06E-08	Reject H_0
BugTraq	3.05E-05	Reject H_0

4.4.1.2 Similarity Score Validation

Next, we validate the similarity measurements in order to check their correctness. The steps of validation are shown in Algorithm 3. For both OSVDB and BugTraq the same steps are performed. First, we identify the matching vulnerability descriptions. We check if any CVE numbers from the NVD set exist in any document from the queried VDs (line 4). Additionally, we check manually online entry of both VDs for each CVE number from NVD set (line 4). The match is found (line 5) if both compared vulnerability descriptions have the same CVE number. Next, for matched vulnerability descriptions the similarity score for *doc* (line 6) is taken along with the maximum similarity score corresponding to particular *cve* (line 7) from NVD vulnerability descriptions set. The result will be correct if the similarity score from a match is equal to the maximum similarity score.

Table 4.7 presents the outcome of the validation steps. We identify 17 matches for OSVDB and 23 for BugTraq. 11% of matching documents for OSVDB and 23% for BugTraq have a similarity score different from max score for the corresponding CVE number. How-

Algorithm 3 Validation of similarity measurements

```
Require: $CVE = \{cve_1 \dots cve_{196}\}$set of CVE numbers from NVD
    $VD$=$\{vd_1$=BugTraq,$vd_2$= OSVDB$\}$
    $doc$ - vulnerability description from the $vd$
Ensure: similarityScore=[$Same$,$Max$]
 1: for each $cve_i$ in $CVE$ do
 2:    for each $vd_i$ in $VD$ do
 3:       for each $doc$ in $vd$ do
 4:          if ($doc$.contain($cve$)=true || searchOnlineEntry.$vd$($cve$)=$doc$) then
 5:             matchFound =true
 6:             $Same$ = TakeSimilarityScore($doc$)
 7:             $Max$ = TakeSimilarityScore($cve$)
 8:          end if
 9:       end for
10:    end for
11: end for
```

ever, the majority of outcomes is recognized correctly. Therefore, we can conclude that our approach usually (OSVDB 89% and BugTraq 77%) gives the correct result.

According to Table 4.7, the smallest similarity score for OSVDB is 0.338; hence, we can conclude that OSVDB entries with smaller than 0.338 similarity score are new vulnerabilities. BugTraq is different as the similarity scores vary between 0.814 and 0.181. Therefore, making the same conclusion as for the OSVDB seems to be incorrect. Such a wide range of results can be caused by a lot of noise in BugTraq textual vulnerability descriptions. As a reminder, the BugTraq vulnerability descriptions are provided as email messages with much information non-related to vulnerability which we consider to be a noise.

4.4.2 Identifying Additional Vulnerability Information

In this experiment we would like to check whether there is any relationship between the syntactic similarity score and additional information from different vulnerability descriptions for the same CVE. The additional information is vulnerability information that is not included in NVD vulnerability description but is significant for building the PAG. The additional information is identified manually by us by comparing the NVD vulnerability descriptions with vulnerability descriptions from others VDs for the same CVE number.

Table 4.7: Validation results for studied VDs; Same indicates the similarity score for the two documents corresponding to the same CVE number; Max Score is the highest score for particular CVE number in NVD matched to any entry from considered VDs.

(a) OSVDB

CVE number	Max Score	Same
CVE-2013-0913	0.682	0.682
CVE-2013-0969	0.646	0.646
CVE-2013-0970	0.846	0.846
CVE-2013-0973	0.636	0.636
CVE-2013-0976	0.76	0.76
CVE-2013-1814	0.446	0.446
CVE-2013-2503	0.338	0.338
CVE-2013-2549	0.413	0.413
CVE-2013-2550	0.501	0.501
CVE-2013-2551	0.421	0.421
CVE-2013-2552	0.465	0.465
CVE-2013-2553	0.471	0.471
CVE-2013-2554	0.518	0.518
CVE-2013-2555	0.458	**0.451**
CVE-2013-2556	0.494	0.494
CVE-2013-2557	0.526	**0.494**
CVE-2013-2566	0.345	0.345
11% of not matching documents		

(b) BugTraq

CVE name	Max Score	Same
CVE-2011-3058	0.584	0.584
CVE-2012-2088	0.416	**0.382**
CVE-2012-3488	0.508	0.508
CVE-2012-3489	0.502	0.502
CVE-2012-3525	0.551	**0.357**
CVE-2012-3749	0.643	0.643
CVE-2012-3756	0.74	0.74
CVE-2013-0156	0.458	0.458
CVE-2013-0333	0.537	0.537
CVE-2013-0787	0.44	0.44
CVE-2013-0963	0.544	0.544
CVE-2013-0966	0.715	0.715
CVE-2013-0967	0.814	0.814
CVE-2013-0969	0.71	0.71
CVE-2013-0970	0.817	0.817
CVE-2013-0971	0.69	0.69
CVE-2013-0973	0.74	0.74
CVE-2013-0976	0.779	0.779
CVE-2013-1667	0.574	**0.231**
CVE-2013-1814	0.589	**0.566**
CVE-2013-2492	0.436	**0.181**
CVE-2013-2503	0.732	0.732
CVE-2013-2560	0.507	**0.291**
23% of not matching documents		

We use vulnerability sets constructed in Section 4.4.1.2 Table 4.7 (OSVDB 17 documents and BugTraq 23 documents). To obtain more accurate similarity prediction for the BugTraq set, we fix its vulnerability descriptions. We substitute original email messages with vulnerability descriptions from BugTraq online entry which contain pure vulnerability descriptions.

We measure syntactic similarity between two vulnerability descriptions for the same CVE number from NVD and OSVDB/BugTraq. We calculate the syntactic similarity score using the previously described analyzers. This analysis can help us with identifying the relationship between the syntactic similarity score and possible additional information included in different vulnerability descriptions.

Table 4.8 summarizes the similarity scores between NVD vulnerability descriptions and studied VDs for both analyzers. The range of similarity scores for bi-gram Analyzer for both OSVDB and BugTraq looks very similar. For the Standard Analyzer, OSVDB has about 70% of the scores between 0.2 and 0.4. For BugTraq and Standard Analyzer, the scores are spread-out more evenly than for OSVDB.

Table 4.8: Summary of the similarity scores between NVD and OSVDB/BugTraq.

	OSVDB		BugTraq	
Interval of Similarity s	Standard	bi-gram	Standard	bi-gram
0<s to s<0.1	0.00%	11.76%	8.70 %	43.48%
s≥0.1 to s<0.2	5.88%	47.06%	17.39%	30.43%
s≥0.2 to s<0.3	23.53%	35.29%	34.78%	21.74%
s≥0.3 to s<0.4	47.06%	5.88%	17.39%	4.35%
s≥0.4 to s<0.5	11.76%	0.00%	17.39%	0.00%
s≥0.5 to s≤1	11.76%	0.00%	4.35%	0.00%

4.4.2.1 Test of Association

In order to check whether there is any relationship between the syntactic similarity score and additional information from different vulnerability descriptions for the same CVE, we run the Fisher Exact Test [Fis22] of association. Our hypothesis (H_A) is that there is an association between score and additional information. The null (H_0) hypothesis is that there is no association between score and additional information. We reject H_0 if p-value $< \alpha = 0.05$.

The contingency tables for both VDs and both Analyzers are shown in Tables 4.9 and 4.10. The "Add. Inf." indicates the counts for vulnerability descriptions with any additional information from studied VDs compared to NVD description for the same CVE number.

The results of the test are shown in Table 4.11. For both VDs and both analyzers we fail to reject H_0 which means that there is no association between score and additional information. Therefore, we cannot conclude that syntactic similarity score can help us with identifying additional information from vulnerability descriptions.

Table 4.9: Contingency Table for OSVDB; Add. Inf. - additional information; No Add. Inf. - no additional information.

(a) Standard

Interval of Similarity s	Add. Inf.	No Add. Inf.	Total
0<s to s<0.1	0	0	0
s≥0.1 to s<0.2	1	0	1
s≥0.2 to s<0.3	2	2	4
s≥0.3 to s<0.4	2	6	8
s≥0.4 to s<0.5	0	2	2
s≥0.5 to s≤1	0	2	2
Total	5	12	17

(b) bi-gram

Interval of Similarity s	Add. Inf.	No Add. Inf.	Total
0<s to s<0.1	2	0	2
s≥0.1 to s<0.2	2	6	8
s≥0.2 to s<0.3	1	5	6
s≥0.3 to s<0.4	0	1	1
s≥0.4 to s<0.5	0	0	0
s≥0.5 to s≤1	0	0	0
Total	5	12	17

Table 4.10: Contingency Table BugTraq; Add. Inf. - additional information; No Add. Inf. - no additional information.

(a) Standard

Interval of Similarity s	Add. Inf.	No Add. Inf.	Total
0<s to s<0.1	0	2	2
s≥0.1 to s<0.2	1	3	4
s≥0.2 to s<0.3	2	6	8
s≥0.3 to s<0.4	2	2	4
s≥0.4 to s<0.5	4	0	4
s≥0.5 to s≤1	0	1	1
Total	9	14	23

(b) bi-gram

Interval of Similarity s	Add. Inf.	No Add. Inf.	Total
0<s to s<0.1	4	6	10
s≥0.1 to s<0.2	3	4	7
s≥0.2 to s<0.3	3	2	5
s≥0.3 to s<0.4	0	1	1
s≥0.4 to s<0.5	0	0	0
s≥0.5 to s≤1	0	0	0
Total	10	13	23

Table 4.11: Fisher's Exact Test for Count Data with α=0.05.

	p-value	
	Standard	bi-gram
OSVDB	0.3988	0.1946
BugTraq	0.08055	0.9229

4.5 Discussions

In this chapter we have analyzed three VDs: NVD, OSVDB and BugTraq. We have examined the amount of new information they offer during a one week period. The most extensive set is provided by NVD. As we expected, semantic similarity can help with identifying the same vulnerability from different sources. However, contrary to our expectations, the syntactic similarity score does not suggest that vulnerability descriptions from other VDs contain information based on which the NVD vulnerability descriptions can be extended. Moreover, the question arises of how important this additional information is and how we can measure this importance. This is a significant issue because we would like to extend vulnerability descriptions to contain only significant information. The solution could be to search for key words which will indicate the concept on which we would like to extend information. This can be the future direction of this work.

Also, in the future work we are going to extend similarity score measurements on boosting with key words which would provide us with more relevant information. Therefore, we can gain more revealing similarity scores. Moreover, we would like to check similarity level between OSVDB and BugTraq because now we perceive only the similarity between those two and NVD.

Chapter 5

Extracting Vulnerability Descriptions

Automatic construction of the PAG is one of the main challenges in automatic security analysis of the home computer. In order to build the PAG, we need information such as infected software, pre-conditions (user actions, attacker strategies, and system activities and configuration) and post-conditions of exploiting vulnerabilities. However, the standard resources for capturing vulnerability information, Vulnerability Databases (VDs), contain textual descriptions which do not explicitly describe the chain of events that lead to an exploit. Such descriptions are also difficult to work with in an automated process.

Let us consider the example of textual vulnerability description shown in Figure 5.1. The top part describes the vulnerability in plain text and the bottom shows information about the infected software. As can be seen, we have not clearly specified the chain of events that lead to exploit.

The EScript.api plugin in Adobe Reader and Acrobat 10.x before 10.0.1, 9.x before 9.4.1, and 8.x before 8.2.6 on Windows and Mac OS X allows remote attackers to execute arbitrary code or cause a denial of service (application crash) via a crafted PDF document that triggers memory corruption, involving the printSeps function. NOTE: some of these details are obtained from third party information.
$*cpe : /a : adobe : acrobat\_reader : 10.0 * cpe : /a : adobe : acrobat\_reader : 9.0 * cpe : /a : adobe : acrobat\_reader : 9.1*$

Figure 5.1: Fragment of CVE-2010-4091 vulnerability description

Table 5.1 presents information that we wish to extract. Please notice that we do not have any direct information about the possible user actions. Therefore, we have to reason, according to provided information about what the possible user action(s) that can trigger the exploit is/are. For Figure 5.1 it is “user opens pdf file.”

In this chapter, we show how a textual vulnerability description can be transformed into a more structured one that is necessary to construct the PAG. We have developed a system

Table 5.1: Information from CVE-2010-4091 needed to build the PAG.

Infected Software	adobe:acrobat_reader:10.0 adobe:acrobat_reader:9.0 adobe:acrobat_reader:9.1
Precondition Attacker	a crafted PDF document that triggers memory corruption
Precondition User	user opens pdf file
Precondition System	The EScript.api plugin involving the printSeps function
Postcondition of Attack	execute arbitrary code cause a denial of service (application crash)

for automatically extracting information from NVD [NVD13] for a set of vulnerabilities identified by vulnerability scanner OpenVAS [Ope12][1].

We apply an automated Information Extraction (IE) method which is based on manually created rules. These rules are created by us and based mostly on regular expressions. First, we discuss our system architecture. We explain how the set of vulnerabilities is constructed. Next, we explain in detail each element of the system. Finally, we discuss the performance results of our system.

5.1 Background

5.1.1 Structuring Vulnerability Description

Le et al. [LSHL09] present Vulnerability Property Relationship Graphs (VPRG), a formal model of web-based vulnerabilities, represented as cause/consequence chains. However, this model is constructed manually. Subsequently, Le et al. [LL11] present an approach for automated extraction of the VPRG model from a plain text vulnerability description. The system relies on the Natural Language Processing Tools Kit, supported by the "Dictionary of Terms and Relationships" which is responsible for providing information about terms, concepts, and relationships in the vulnerability descriptions. However, constructing the dictionary requires significant effort and is prone to errors. Each sentence is partitioned into

[1]Published in [URHR12]

entities (noun, prepositional and verb phrases) and behavior (properties which describe the functionality of the web-application), and restated as a cause/consequence chain.

Mulwad et al. [MLJ$^+$11] use Wikitology as a knowledge based system and computer security taxonomy to extract vulnerability information. They collect information from different sources and classify it to identify possible vulnerability descriptions. Next, the information extractor extracts relevant information using knowledge based system taxonomy. This information is then coded into Web Ontology Language (OWL) for further automated analysis. The authors provide the methodology to extract only the methods of attack and the post-condition of attack from vulnerability description. We are interested in extracting more information such as: infected software and all pre-conditions and post-conditions of exploiting a particular vulnerability that lead to an attack.

5.1.2 Tools

5.1.2.1 OpenVAS

The Open Vulnerability Assessment System (OpenVAS) [Ope12] is an open source vulnerability scanning and management system that provides complete analysis by combining services and security tools. The security scanner is supported by the daily updated feed of the Network Vulnerability Tests (NVTs), which is responsible for the detection of known and potential security vulnerabilities. The OpenVAS offers over 25,000 NVTs (as of May 2012) and is available under the GNU General Public License (GNU GPL). It offers a variety of audit report formats e.g. XML, HTML, LaTeX, PDF, TXT, etc.

5.1.2.2 Parsers

For the purpose of extraction we use two different parsers:

SAXparser (Simple API for XML) [Meg13] is an event-driven, serial-access mechanism for accessing XML documents.

Jsoup [Jso12] is an open source project available under the liberal MIT license. It is a Java library which works with HTML website code. It offers handy API for extraction and data management. It can parse HTML from an URL, file, or string sources.

5.2 System Design

Our objectives are to extract information about the necessary pre-conditions of exploiting a vulnerability and the consequences of its exploit, from plain text vulnerability description. Similar to [LL11], we use dictionaries to guide the extraction process. However, we store only the keywords, such as the name of the software. Our approach is based on a set of rules which use also keywords or phrases e.g., "by," "via," "allows remote attacker to."

Figure 5.2 presents the information flow for our system to convert NVD entries to a structured form. We use NVD as our source of textual vulnerability descriptions, as well as the OpenVAS scanner to identify vulnerabilities in a single host. As in [JNO05, WLI07], a computer is scanned, and an audit report is created and parsed by the customized XML parser. The output of this step is a list of vulnerabilities. The CVE numbers are extracted from this list; the vulnerability record for each CVE number in the list obtained from the NVD in HTML format. The relevant vulnerability information is extracted by the HTML parser. The result at this point is a textual vulnerability description (Figure 5.1 the NVD entry CVE-2010-4091).

Further, text processing operations are done on this description by applying a set of filters. The main goal of the filters is to extract necessary causal information such as infected software names, action pre-conditions, and post-condition of exploit. The new structured vulnerability description is constructed by applying all the filters.

5.2.1 CVE Number Extraction

The vulnerabilities identified by OpenVAS are grouped into sets with the same NVT (see Section 5.1.2.1). The XML report created by the scanner contains a lot of XML elements, many of which are not relevant to our analysis (such as scanned port, names of the tasks, etc.). The parser uses sets of rules to parse the XML tree by looking for the relevant XML elements such as ⟨result⟩, ⟨nvt⟩ and ⟨cve⟩. For each ⟨cve⟩ element, the list of CVE numbers associated with the same NVT is extracted. Finally, a single list with all extracted CVEs is created.

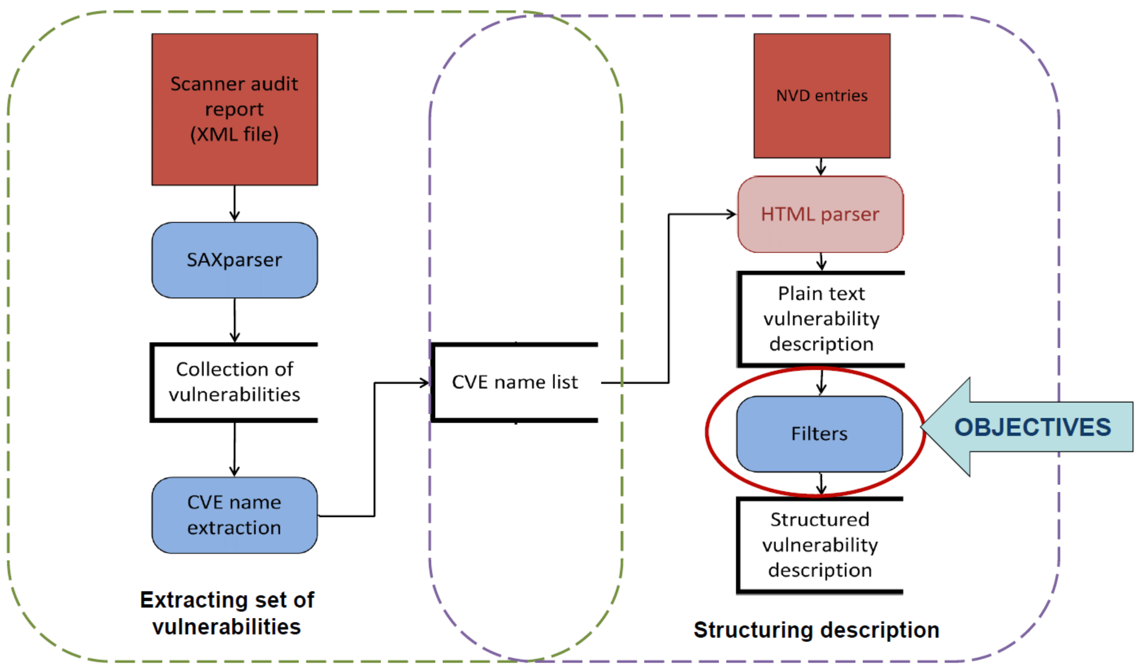
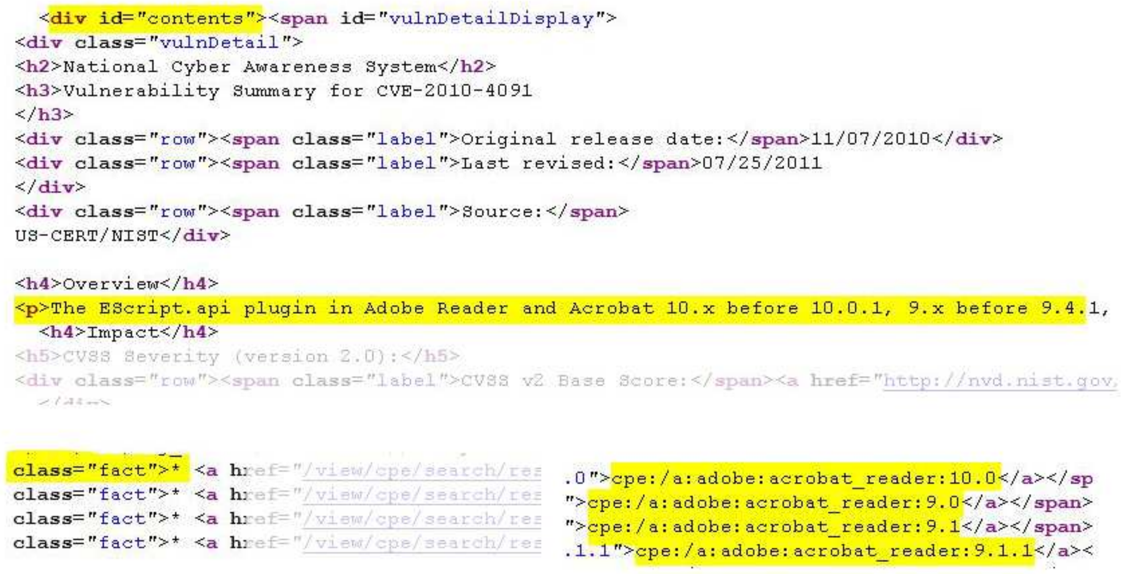

Scanner audit report (XML file)
SAXparser
Collection of vulnerabilities
CVE name extraction
Extracting set of vulnerabilities
CVE name list
NVD entries
HTML parser
Plain text vulnerability description
Filters
OBJECTIVES
Structured vulnerability description
Structuring description

Figure 5.2: Information flow; red squares denote outside system information, blue rounded rectangles are processes, and unclosed rectangles are data storage.

We would like to mention that different vulnerability scanners can be used to scan the single host. However, in order to get the CVE numbers list, the scanner report parser may need to be customized depending on report's structure.

5.2.2 HTML Parser

Each CVE number is submitted to the NVD search engine. Using the Jsoup library, the vulnerability description is extracted from NVD website as HTML code and converted to plain text. Algorithm 4 shows the extraction process. From HTML (Figure 5.3 shows example source code for CVE-2010-4091) the vulnerability description is identified (lines 5 and 6). Next, using the *class="fact"*, the information about infected software is recognized (line 7). Finally, the vulnerability description and infected software are convert from HTML to a plain text (line 8).



```
  <div id="contents"><span id="vulnDetailDisplay">
<div class="vulnDetail">
<h2>National Cyber Awareness System</h2>
<h3>Vulnerability Summary for CVE-2010-4091
</h3>
<div class="row"><span class="label">Original release date:</span>11/07/2010</div>
<div class="row"><span class="label">Last revised:</span>07/25/2011
</div>
<div class="row"><span class="label">Source:</span>
US-CERT/NIST</div>

<h4>Overview</h4>
<p>The EScript.api plugin in Adobe Reader and Acrobat 10.x before 10.0.1, 9.x before 9.4.1,
  <h4>Impact</h4>
<h5>CVSS Severity (version 2.0):</h5>
<div class="row"><span class="label">CVSS v2 Base Score:</span><a href="http://nvd.nist.gov,

class="fact">* <a href="/view/cpe/search/res   .0">cpe:/a:adobe:acrobat_reader:10.0</a></sp
class="fact">* <a href="/view/cpe/search/res   ">cpe:/a:adobe:acrobat_reader:9.0</a></span>
class="fact">* <a href="/view/cpe/search/res   ">cpe:/a:adobe:acrobat_reader:9.1</a></span>
class="fact">* <a href="/view/cpe/search/res   .1.1">cpe:/a:adobe:acrobat_reader:9.1.1</a><
```

Figure 5.3: Example input for HTML parser.

Algorithm 4 Extraction of Vulnerability Description from the HTML Code.

```
Require: $list$={$list\ of\ items\ with\ CVEnumbers$ }
    $NVDaddress$= {$http://web.nvd.nist.gov/view/vuln/detail?vulnId=$}
Ensure: $[text, sofListText]$={$plain\ text\ vulnerability\ description\ as\ a\ String$}
 1: for $item \in list$ do
 2:   if $item$ from $list$ is $CVEnumber$ then
 3:     $url$= $NVDaddress$ + $CVEnumber$
 4:     $html$=connect($url$)
 5:     $vulDetail$=$html$.getElementByID("contents")
 6:     $overview$=$vulDetail$.getElementByTag("p")
 7:     $softwareList$=$html$.getElementByClass("fact").select("*")
 8:     [$text$,$sofListText$]=convertToText($overview$,$softwareList$)
 9:   else
10:     print "no CVE number"
11:   end if
12: end for
```

5.2.3 Filters

To better understand how the extraction process works, as well as its complexities, we use the CVE-2010-4091 from Figure 5.1 as an example. Five filter groups are responsible for identifying distinct information from a vulnerability description. These are 1) infected software, 2) pre-condition attacker, 3) pre-condition user, 4) pre-condition system, and 5) post-condition of attack. The final step is collating the strings returned from filters into a text file which forms the new structured VD. The output file for our example is shown in Figure 5.4b.

5.2.3.1 Infected Software Filters

These filters identify the software (by name and version) that is involved in a particular vulnerability. They are implemented as a Java class. The infected software from Figure 5.1 are: *adobe:acrobat_reader:10.0*, *adobe:acrobat_reader:9.0*, and *adobe:acrobat_reader:9.1*.

Algorithm 5 shows how filters work. First, *sofListText* (textual software list from HTML parser) is tokenized according to whitespace (line 1) e.g., token: "*cpe:/a:adobe:acrobat _reader:10.0*". If the token contains *cpe*, it is further analyzed (line 3). The token is then searched for *keyWords* which provide "start" and "stop" conditions for the extraction (lines 4 and 5). According to the previous example *keyWord* "start" is *cpe:/a:*, and the end of extraction phrase is indicated by a whitespace. The process is repeated until all tokens are checked.

Algorithm 5 Extracting Infected Software.

```
Require: sofListText={software list from HTML parser}
    keyWord={words which support extraction}
Ensure: InfSofList={Infected software list}
 1: tokens=tokenize(sofListText)
 2: for token ∈ tokens do
 3:     if token.contain("cpe") then
 4:         indexToStart=token.find(keyWord)
 5:         indexToStop=token.length()
 6:     else if ... then
 7:         ...
 8:     else
 9:         ...
10:     end if
11: end for
```

5.2.3.2 Attacker Action Pre-condition Filters

These filters are responsible for extracting information about the attacker's involvement in exploiting a vulnerability (as shown in algorithm 6). These filters are implemented as two different Java classes. They are based on two main keywords: "via" and "by." From Figure 5.1, the phrase: "*a crafted PDF document that triggers memory corruption*" captures an attacker pre-condition.

Algorithm 6 presents how filters works. First, the *keyWordsStart* is identified, in our example it is "via" (line 2). Next in line 4, the sub-phrase of the *text* starting from *keyWordsStart* is extracted. Subsequently, the sub-phrase is searched for appropriate *keyWordsStop*. In this example it is a comma symbol. In the last step line 8, the final *phrase* is extracted.

However, in some cases, parsing this description is not straightforward. Consider a phrase from CVE-2010-0483: "*by referencing a (1) local pathname, (2) UNC share pathname, or (3) WebDAV server with a crafted .hlp file in the fourth argument (aka helpfile argument) to the MsgBox function*"– the *keyWordsStart* is "by" but we cannot use the first comma for a terminating condition because we would lose the rest of the information in the list. Our program currently does not support this kind of extraction. For such a vulnerability description our program returns only "referencing a (1) local pathname." However, it can be handled by extending the filters conditions of the new regular expressions, for example, by adding a condition to consider the numbers or parentheses.

Algorithm 6 Extracting Attacker Action Pre-conditions.

```
Require: $text$={$vulnerability\ description\ as\ a\ String$}
    $keyWordStart, keyWordStop$={$word\ or\ phrase\ which\ support\ extraction$}
Ensure: $phrase$={$phrase\ of\ Attacker\ Action\ Pre\text{-}conditions$}
 1: scan $text$ from left to right
 2: $indexToStart = text.\text{find}(keyWordStart)$
 3: if $indexToStart \geq 0$ then
 4:     $stringToLook = text.\text{substring}(indexToStart)$
 5:     $indexToStop = stringToLook.\text{find}(keyWordStop)$
 6:     if $indexToStop \geq 0$ then
 7:         $phrase$=substring($keyWordStart, keyWordStop$)
 8:         return $phrase$
 9:     end if
10: else
11:     return null
12: end if
```

5.2.3.3 System Pre-condition Filters

System pre-condition filters identify the system conditions needed for a successful attack, e.g., the phrase "*The EScript.api plugin*". They are implemented as two Java classes. They are based on two main keywords: "do not properly" and "in." These filters work similar to Algorithm 6 with a different set of keywords adapted to this specific extraction. As with the previous filters, the lack of consistency in vulnerability descriptions leads to added complexity. For instance, in (CVE-2008-3107) "*Unspecified vulnerability in the Virtual Machine in Sun Java Runtime Environment (JRE)*", the keyword "in" is not useful as a cue; therefore, we have to keep checking contiguous word-tokens until the name of the software is found. Then the extraction rule stops and returns the whole string. However, system information is often provided in different places in vulnerability description. Hence, it is very hard to model the regular expression rules which address these differences.

5.2.3.4 User Action Pre-condition Filters

These filters identify the conditions necessary for the user's involvement in exploiting a vulnerability. They are implemented as eight different Java classes; each class is customized for a specific keyword. The classes are checked in sequence and the suitable one for a particular vulnerability description is used. In most cases, the descriptions do not include the conditions necessary for the user's involvement in exploiting a vulnerability explicitly. So, our system has to reason from what is there, i.e., check a set of conditions and infer (as shown in algorithm 7). From our example, the phrase "*open pdf document*" will be returned.

First, the *text* is tokenized (line 1). Next, the program searches for a match between keywords from *helpFile* and the *tokens* (line 2-4). If a match is found it returns the appropriate *userAction.*

5.2.3.5 Post-condition Filters

The post-condition filters extract the consequences of exploiting a particular vulnerability. They are implemented as seven different Java classes with specific keywords. The classes are

Algorithm 7 Extracting User Action Pre-conditions.

```
Require: $text$ {$vulnerability\ description\ as\ a\ String$}
    $keyWord$ {$word\ or\ phrase\ which\ support\ extraction$}
    $helpFile$ {file with $keyWords$}
Ensure: $userAction$ {$user\ actions\ associated\ with\ exploit$}
 1: $token \leftarrow$ tokenize($text$)
 2: while $text$ hasNext $token$ do
 3:     get($token$)
 4:     $keyWordToCheck$=$helpFile$.take($keyWord$)
 5:     if $token$=$keyWordToCheck$ then
 6:         return $userAction$
 7:         break
 8:     else
 9:         getNext($token$)
10:     end if
11: end while
```

checked in sequence and the suitable one for a particular vulnerability description is used. In our example, filters will find the phrase "*execute arbitrary code or cause a denial of service (application crash)*" which describes the two ways in which exploiting this vulnerability affects the system.

Algorithm 8 presents how filters works. First, the *keyWordsStart* is identified; in our example, it is "allows remote attackers to" (line 2). Next, the sub-phrase of the *text* starting from *keyWordsStart* is extracted (line 4). Subsequently, the sub-phrase is searched for appropriate *keyWordsStop*, in the studied example it is a "via" word. In the last step, the final *phrase* is extracted (line 8).

5.3 Evaluation

We evaluate our program using the evaluation procedure applied in the Message Understanding Conference (MCU) [Chi92]. According to Nadeau et al. [NS07], the extraction system is ranked according to two criteria: the ability to find the correct type and the ability to find the exact text. We will use four measures:

- Correct - elements extracted correctly
- Missing - elements not extracted that should have been

Algorithm 8 Extracting Post-conditions of Exploit.

```
Require: $text$={$vulnerability\ description\ as\ a\ String$}
    $keyWordStart, keyWordStop$={$word\ or\ phrase\ which\ support\ extraction$}
Ensure: $phrase$={$phrase\ of\ post\text{-}conditions\ of\ exploit$}
 1: scan $text$ from left to right
 2: $indexToStart = text.\mathrm{find}(keyWordStart)$
 3: if $indexToStart \geq 0$ then
 4:     $stringToLook = text.\mathrm{substring}(indexToStart)$
 5:     $indexToStop = stringToLook.\mathrm{find}(keyWordStop)$
 6:     if $indexToStop \geq 0$ then
 7:         $phrase$=substring($keyWordStart, keyWordStop$)
 8:         return $phrase$
 9:     end if
10: else
11:     return null
12: end if
```

- Spurious - elements extracted wrongly
- Partial - only the part of the information which should be extracted

Using the above measures the following criteria are defined [Chi92, NS07]:

- Precision - How many of the identified elements are correct?

$$Precision = \frac{Correct + 0.5 * Partial}{Correct + Spurious + Partial}$$

- Recall - How many of the elements that exist are identified?

$$Recall = \frac{Correct + 0.5 * Partial}{Correct + Missing + Partial}$$

- $$F\text{-}measure = \frac{(\beta^2 + 1) * Precision * Recall}{(\beta^2 * Precision) + Recall}$$

  The *F-measure* is calculate for β=1 and from this point it is called $F1$-*measure*.

We used 240 vulnerability descriptions identified on a computer running Microsoft Windows XP Professional SP3 with standard configuration. Before collecting the data, the system was secured and updated. Subsequently, the machine was disconnected from the Internet, and automatic updates were disabled. After a few months, the machine was plugged into the Internet and scanned, 244 vulnerabilities were identified. For the purposes of our analysis, we have dropped four of them because their descriptions greatly differ from the rest: 1) they are very short, 2) they provide information about hardware rather than software, and 3) they were published in the late ’90s.

We validate the extracted information produced by our system by comparing it manually with the original descriptions. The results of the comparison are expressed using four measures described above (correct, missing, spurious, partial). Next, we have calculated the precision, recall and F1-measure for each type of extracted information (infected software, precondition attacker, precondition user, precondition system, and postcondition of attack).

5.3.1 Description Accuracy

For each of the 240 vulnerability descriptions, a file with structured vulnerability description is created. The example output from our system is presented in Figure 5.4b, and the original description is shown in Figure 5.4a. From the original description we construct the Gold Standard (ideal output) - Table 5.1. We have compared elementwise the output from our system (structured description) with its Gold Standard. The results of comparison are retained for each type of extracted information. Table 5.2 presents results for CVE-2010-4091. Similarly, we compare each structured vulnerability description with its Gold Standard.

Table 5.2: Results of the comparison between structured description and Gold Standard for CVE-2010-4091.

	Correct	Missing	Spurious	Partial
Infected Software	55	0	0	0
Pre-condition Attacker	1	0	0	0
Pre-condition User	1	0	0	0
Pre-condition System	1	1	0	0
Post-condition of Attack	2	0	0	0

The EScript.api plugin in Adobe Reader and Acrobat 10.x before 10.0.1, 9.x before 9.4.1, and 8.x before 8.2.6 on Windows and Mac OS X allows remote attackers to execute arbitrary code or cause a denial of service (application crash) via a crafted PDF document that triggers memory corruption, involving the printSeps function. NOTE: some of these details are obtained from third party information.
$*cpe : /a : adobe : acrobat\_reader : 10.0 * cpe : /a : adobe : acrobat\_reader : 9.0 * cpe : /a : adobe : acrobat\_reader : 9.1*$

(a) Original vulnerability description.

Infected Software:
    adobe:acrobat_reader:10.0
    adobe:acrobat_reader:9.0
    adobe:acrobat_reader:9.1
    ...
Pre-condition:
    attackerAtributes
        a crafted PDF document that triggers memory corruption
    userAtributes
        user opens pdf file
    systemAtributes
        The EScript.api plugin
Post-condition:
    execute arbitrary code
    cause a denial of service (application crash)

(b) Structured vulnerability description produced by our system.

Figure 5.4: Vulnerability description for CVE-2010-4091 in non-structured (a) and structured form (b) after extraction.

The performance measurements for each type of extracted information are shown in Table 5.3a. The precision of extracted information is very high for almost all of the information. It means that identified information is in the majority correct. The "Infected Software" is extracted with perfect accuracy. This is because this information is provided in a very structured way.

The "Post-condition of Attack" has slightly worse results with precision being 0.985. However, not all the "Post-condition of Attacks" are recognized by our system (recall 0.796). Increasing the number of filters that address specific vulnerability descriptions that were not previously considered can improve the recall.

A similar situation is for the "Pre-condition User" where the precision is very high 0.991 and recall is 0.513. Such poor results are caused by the fact that this information depends on the context of the description. The accuracy of the system strictly depends on the quality of the rules for extraction and their ability to infer what is implied. The results can be improved by extending the set of keywords which support extraction.

The main problem of our system is extracting the “Pre-condition System.” This is caused by a large number of partial identification. Solving this issue is not easy because building new filters does not guarantee precision and recall improvement. This is because the ways that the “Pre-condition system” are provided in vulnerability descriptions are challenging to address by the regular expressions.

Table 5.3: Summary of performance.

(a) Results for five type of extracted information.

	Precision	Recall	F1-measure
Infected Software	1	1	1
Pre-condition Attacker	0.984	0.948	0.966
Pre-condition User	0.991	0.513	0.676
Pre-condition System	0.656	0.445	0.53
Post-condition of Attack	0.985	0.796	0.881

(b) Overall system performance.

Precision	Recall	F1-measure
0.93	0.701	0.8

5.4 Discussions

The vulnerability descriptions available from current VDs are not in a form that expedites security analyses, especially when the system being analyzed is a single computer. The descriptions do not readily provide information about the user’s or the attacker’s role in the vulnerability exploitation. Moreover, in order to construct PAG the causal relationships between user activities and preferences, attacker strategies, and system activities that lead to an exploit need to be inferred from the textual description. What complicates matter is that vulnerabilities are not described in a uniform and consistent manner across different databases. As a result, it becomes extremely difficult to extract relevant information from vulnerability databases and synthesize them into a structured description that is suitable for analysis.

We have demonstrated a method for automatically extracting vulnerability information from plain-text. We have evaluated that method on 240 vulnerabilities. The current version of our program is able to construct descriptions with precision 0.93, recall 0.7 which gives the F1-measure 0.8. The results are satisfactory. However, when we consider each type of extracted information separately, we can recognize where our system has issues and where it performs very well.

Thus, given the variability in vulnerability descriptions, the program's accuracy is likely to decrease as new vulnerability descriptions are added. New patterns will need to be added to the filters to accommodate for the lack of consistency in the prose for the vulnerability descriptions and the lack of explicitly included information. Therefore, correct extraction requires an expansion in the number of rules which grows quickly with the number of cases. Therefore, we should generate the extraction patterns automatically.

In our future work, we plan to address most of the problems stated above. We will also reduce the human factors that can cause errors in the whole process of extraction. To accomplish this, we are going to use one of the existing information extraction systems with a machine learning algorithm to generate extraction patterns. We will build a new IE system that will use both hand-coded method and the machine learning algorithm.

Chapter 6

Conclusions and Future Work

Automated security analysis of the home computer can be expressed as a graph structure called Personalized Attack Graph (PAG). The PAG requires the extensive information about: 1) vulnerabilities present in the home computer, 2) information about home user behavior, 3) attacker strategies, and 4) home computer activities that all together lead to exploiting the security threats.

In this thesis, we have elaborated on modeling the user behavior and obtaining the most comprehensive vulnerability descriptions. We have also presented the methodology of transforming vulnerability information into a format useful in automatic construction of the PAG.

6.1 Conclusions

We have modeled the home user to obtain information about her/his behavior which provides information to complement the PAG's components. We have analyzed the existing conceptual user models in order to understand the relationships between user characteristics. This helps us to create our conceptual home user model which addresses home computer user activities. We have measured the likelihood of these activities by conveying a conceptual model into a Bayesian network - Bayesian User Action (BUA). We have examined how different user profiles influence the probability of compromise of the computer system. We have shown that, depending on user personality, the probability of threats differs. Preventing the user from taking some actions can improve the security. Furthermore, we have analyzed the accuracy of our user model. We were interested in identifying how sensitive to changes our model is. We have demonstrated that minor changes in user information do not significantly impact the probability of user's action.

We have analyzed the Vulnerability Databases (VDs) in order to choose the source of vulnerability information. Constructing the PAG requires specific information about user actions, attacker strategies, and system activities which cannot always be obtained from a single source. We have compared collected information and measured the similarity between vulnerability descriptions from different VDs. We have demonstrated that calculating semantic similarity can be helpful in identifying the same vulnerability across different VDs. We have failed in demonstrating that syntactic similarity can help with identifying any additional significant information about vulnerability. However, we believe that the issue is not in the concept of using syntactic similarity measurement, but in the method that was used.

We have described a methodology of structuring the textual vulnerability descriptions in terms of pre- and post-conditions of exploiting vulnerability. The structured vulnerability descriptions are used in the automatic construction of the PAG. We have implemented a hand-coded Information Extraction method. Our extraction system is based on manually created rules, and it structures vulnerability descriptions taken from the National Vulnerability Database. We have constructed five filter groups which are responsible for identifying distinct information from a vulnerability description (Infected Software, Precondition Attacker, Precondition User, Precondition System, and Postcondition of Attack). We have measured the performance of our system on 240 vulnerability descriptions. We have shown that our hand-coded system works well for the majority of vulnerability descriptions. We have identified two issues 1) the need for creating new patterns in order to compensate for the lack of consistency in the prose for the vulnerability descriptions, and 2) the lack of explicitly included information in vulnerability descriptions. The correct extraction requires an expansion in the number of rules which grows quickly with the number of cases. Therefore, we should generate the extraction patterns automatically.

6.2 Future Work

In future work we are going to analyze more deeply each of presented objectives along with addressing identified issues. We are going to redefine our BUA to include more details about home user personality. We are planning to address it by applying further psychological study on home computer users. Also, we will automate the mapping of the *Bayesian User Profile* (BUP) into the PAG. This should be addressed to streamline the process of modeling the user and obtaining probability values which are used in the PAG.

Furthermore, in future work we will check different methods of similarity measurements in order to get additional information about vulnerability. We believe that using keywords or field association [AFMA03] can greatly improve this similarity measure.

We are going to extend our IE system to address the problems with extraction of some vulnerability information. We believe that this can be achieved by applying machine learning algorithm. The machine learning methods are dependent on corpora annotations used to train machine learning models of extraction [Sar08]. Combining these two methods can result in a diligent and infallible IE system.

References

[AC03] K. Aytes and T. Conolly. A Research Model for Investigating Human Behavior Related to Computer Security. In *Proceedings of the 9th Americas Conference on Information Systems*, Tampa, FL, USA, Aug. 2003.

[AC04] K. Aytes and T. Connolly. Computer Security and Risky Computing Practices: A Rational Choice Perspective. *Journal of Organizational and End User Computing*, 16(3):22–40, July-Sept. 2004.

[AFMA03] E. Atlam, M. Fuketa, K. Morita, and J. Aoe. Documents Similarity Measurement Using Field Association Terms. *Information Processing & Management*, 39(6):809 – 824, Nov. 2003.

[APRS05] P. Ammann, J. Pamula, R. Ritchey, and J. Street. A Host-Based Approach to Network Attack Chaining Analysis. In *Proceedings of the 21st Annual Computer Security Applications Conference*, Tucson, AZ, USA, Dec. 2005.

[AWK02] P. Ammann, D. Wijesekera, and S. Kaushik. Scalable, Graph-Based Network Vulnerability Analysis. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, Washington, DC, USA, Nov. 2002.

[Ban77] A. Bandura. Self-efficacy: Toward a Unifying Theory of Behavioral Change. *Psychological Review*, 84(2):191–215, Mar. 1977.

[BEJS10] H. Birkholz, S. Edelkamp, F. Junge, and K. Sohr. Efficient Automated Generation of Attack Trees from Vulnerability Databases. In *Working Notes for the 2010 AAAI Workshop on Intelligent Security*, pages 47–56, Atlanta, GA, USA, July 2010.

[BFP08] P. Bryant, S.M. Furnell, and A.D. Phippen. Improving Protection and Security Awareness Amongst Home Users. In *Advances in Networks, Computing and Communications 4*. University of Plymouth, Apr. 2008.

[BFS05] K.C. Baumgartner, S. Ferrari, and C.G. Salfati. Bayesian Network Modeling of Offender Behavior for Criminal Profiling. In *Proceedings of the Conference of Decision and Control*, Seville, Spain, Dec. 2005.

[BM01] E. Bingham and H. Mannila. Random Projection in Dimensionality Reduction: Applications to Image and Text Data. In *Proceedings of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, San Francisco, CA, USA, Aug. 2001.

[BWL$^+$12] S. Byrne, Z., J. M. Weidert, J. Liff, M. Horvath, C. Smith, A. Howe, and I. Ray. Perceptions of Internet Threats: Behavioral Intent to Click Again. In *Proceedings of the Society for Industrial and Organizational Psychology Conference*, San Diego, CA, USA, Apr. 2012.

[CEN11] US CENSUS. Computer and Internet Use in the United States: 2011. Available from `http://www.census.gov/hhes/computer/publications/2011.html`.

[Chi92] N. Chinchor. MUC-4 Evaluation Metrics. In *Proceedings of the 4th Conference on Message Understanding*, McLean, VA, USA, June 1992.

[CJ10] C. Claar and J. Johnson. Analyzing the Adoption of Computer Security Utilizing the Health Belief Model. *Issues in Information Systems*, 11(1):286–291, 2010.

[Cla11] C.L. Claar. *The Adoption of Computer Security: An Analysis of Home Personal Computer User Behavior Using the Health Belief Model.* PhD thesis, Utah State University, May 2011.

[Con06] W. A. Conklin. *Computer Security Behaviors of Home PC Users: a Diffusion of Innovation Approach.* PhD thesis, The University of Texas at San Antonio, 2006.

[CVE13] Common Vulnerabilities and Exposure. The Dictionary of Common Names (CVE Identifiers). http://cve.mitre.org/, May 2013.

[CVSScall13] National Vulnerability Database. Common Vulnerability Scoring System Version 2 Calculator. Available from `http://nvd.nist.gov/cvss.cfm?calculator\&version=2`, July 2013.

[DCH02] J. Dawkins, C. Campbell, and J. Hale. Modeling Network Attacks: Extending the Attack Tree Paradigm. In *Proceedings of the Workshop on Statistical Machine Learning Techniques in Computer Intrusion Detection*, Baltimore, MD, USA, June 2002.

[DKC09] R. Dantu, P. Kolan, and J. W. Cangussu. Network Risk Management Using Attacker Profiling. *Security and Communication Networks*, 2(1):83–96, 2009.

[DPRW07] R. Dewri, N. Poolsappasit, I. Ray, and D. Whitley. Optimal Security Hardening Using Multi-Objective Optimization on Attack Tree Models of Networks. In *Proceedings of the 14th ACM Conference on Computer and Communications Security*, Alexandria, VA, USA, Oct. 2007.

[DSL13] Decision Systems Laboratory (DSL), School of Information Sciences, University of Pittsburgh. SMILE (Structural Modeling, Inference, and Learning Engine) and User Interface GeNIe. Available from `http://genie.sis.pitt.edu`, July 2013.

[FBP07] S.M. Furnell, P. Bryant, and A.D. Phippen. Assessing the Security Perceptions of Personal Internet Users. *Computers & Security*, 26(5):410–417, 2007.

[FHH$^+$02] B. Friedman, D. Hurley, D.C. Howe, H. Nissenbaum, and E. Felten. Users' Conceptions of Risks and Harms on the Web: A Comparative Study. In *Extended Abstracts of the Conference on Human Factors in Computing Systems*, Minneapolis, MN, USA, Apr. 2002.

[Fis22] R. A. Fisher. On the Interpretation of $\chi 2$ from Contingency Tables, and the Calculation of P. *Journal of the Royal Statistical Society*, 85(1):87–94, 1922.

[FWSJ08] M. Frigault, L. Wang, A. Singhal, and S. Jajodia. Measuring Network Security Using Dynamic Bayesian Network. In *Proceedings of the 4th ACM Workshop on Quality of Protection*, Alexandria, VA, USA, Oct. 2008.

[GDG$^+$05] N. Good, R. Dhamija, J. Grossklags, D. Thaw, S. Aronowitz, D. Mulligan, and J. Konstan. Stopping Spyware at the Gate: a User Study of Privacy, Notice and Spyware. In *Proceedings of the 2005 Symposium on Usable Privacy and Security*, Pittsburgh, PA, USA, July 2005.

[HMFW12] L. Huang, D. Milne, E. Frank, and I. H. Witten. Learning a Concept-based Document Similarity Measure. *Journal of the American Society for Information Science and Technology*, 63(8):1593–1608, 2012.

[HRR$^+$12] A. E. Howe, I. Ray, M. Roberts, M. Urbanska, and Z. Byrne. The Psychology of Security for the Home Computer User. In *Proceedings of the 2012 IEEE Symposium on Security and Privacy*, San Francisco, CA, USA, May 2012.

[HTZJ10] S. Huang, H. Tang, M. Zhang, and J.Tian. Text Clustering on National Vulnerability Database. In *Proceedings on the 2nd International Conference on Computer Engineering and Applications*, Bali Island, Indonesia, Mar. 2010.

[HY10] L. Huigang and X. Yajiong. Understanding Security Behaviors in Personal Computer Usage: A Threat Avoidance Perspective. *Journal of the Association for Information Systems*, 11(7):394–413, July 2010.

[JB84] N. K. Janz and M. H. Becker. The Health Belief Model: A Decade Later. *Health Education & Behavior*, 11(1):1–47, 1984.

[JNO05] S. Jajodia, S. Noel, and B. O'Berry. *Topological Analysis of Network Attack Vulnerability*, volume 5 of *Massive Computing*, chapter 9, pages 247–266. Springer US, 2005.

[Jso12] Jsoup. Java HTML Parser. Available from `http://jsoup.org/`, Sept. 2012.

[JSW02] S. Jha, O. Sheyner, and J. Wing. Two Formal Analyses of Attack Graphs. In *Proceedings of the 15th IEEE Workshop on Computer Security Foundations*, Cape Breton, Nova Scotia, Canada, June 2002.

[KN11] K. B. Korb and A. E. Nicholson. *Bayesian Artificial Intelligence.* CRC Press, 2011.

[Kri01] M. L. Krieg. *A Tutorial on Bayesian Belief Networks.* DTIC Document, 2001.

[KSO$^+$01] A. Kuenzer, Ch. Schlick, F. Ohmann, L. Schmidt, and H. Luczak. An Empirical Study of Dynamic Bayesian Networks For User Modeling. In *Proceedings of the Workshop on Machine Learning for User Modeling*, Sonthofen, Germany, July 2001.

[Lea03] J. Leach. Improving User Security Behaviour. *Computers & Security*, 22(8):685–692, 2003.

[LJL13] Y. Lin, J. Jiang, and S. Lee. A Similarity Measure for Text Classification and Clustering. *IEEE Transactions on Knowledge and Data Engineering*, to appear, 2013.

[LL11] H.T. Le and P.K.K. Loh. Using Natural Language Tool to Assist VPRG Automated Extraction from Textual Vulnerability Description. In *Proceedings of IEEE Workshops of International Conference on Advanced Information Networking and Applications*, Biopolis, Singapore, Mar. 2011.

[LM05] Y. Liu and H. Man. Network Vulnerability Assessment Using Bayesian Networks. In *Proceedings of the SPIE - Data Mining, Intrusion Detection, Information Assurance, and Data Networks Security*, volume 5812, pages 61–71. Mar. 2005.

[LSHL09] H.T. Le, D. Subramanian, Wen Jing Hsu, and P.K.K. Loh. An Empirical Property-Based Model for Vulnerability Analysis and Evaluation. In *Proceedings of the 4th IEEE Asia-Pacific Services Computing Conference*, Singapore, Dec. 2009.

[Luc13] The Apache LuceneTM project. Apache Lucene Core. Available from `http://lucene.apache.org/core`, May 2013.

[Math13] The Apache Software Foundation. The Apache Commons Mathematics Library. Available from `http://commons.apache.org/proper/commons-math/`, Jan. 2013.

[Meg13] D. Megginson. Simple API for XML. Available from `http://www.saxproject.org/`, Sept. 2013.

[MLC09] G. R. Milne, L. I. Labrecque, and C. Cromer. Toward an Understanding of the Online Consumer's Risky Behavior and Protection Practices. *Journal of Consumer Affairs*, 43(3):449–473, 2009.

[MLJ$^+$11] V. Mulwad, W. Li, A. Joshi, T. Finin, and K. Viswanathan. Extracting Information about Security Vulnerabilities from Web Text. In *Proceedings of the 2011 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology - Workshops*, Lyon, France, Aug. 2011.

[Mos13a] J. Moss. Computer Security Conference. Available from `https://www.blackhat.com`, Oct. 2013.

[Mos13b] J. Moss. DEFCON-Annual Hacker Conventions. Available from `http://www.defcon.org`, Oct. 2013.

[MRS08] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA, 2008.

[MSR07] P. Mell, K. Scarfone, and S. Romanosky. *CVSS: A Complete Guide to the Common Vulnerability Scoring System Version 2.0.* FIRST: Forum of Incident Response and Security Teams, June 2007.

[NJOJ03] S. Noel, S. Jajodia, B. O'Berry, and M. Jacobs. Efficient Minimum-Cost Network Hardening via Exploit Dependency Graphs. In *Proceedings of the 19th Annual Computer Security Applications Conference*, Las Vegas, NV, USA, Dec. 2003.

[NKX09] B. Ng, A. Kankanhalli, and Y. Xu. Studying Users' Computer Security Behavior: A Health Belief Perspective. *Decision Support Systems*, 46(4):815–825, 2009.

[NS07] D. Nadeau and S. Sekine. A Survey of Named Entity Recognition and Classification. *Lingvisticae Investigationes*, 30(1):3–26, 2007.

[NVD13] National Vulnerability Database. NVD XML Feed Documentation. Available from `http://nvd.nist.gov`, May 2013.

[Ope12] The Open Vulnerability Assessment System (OpenVAS). Community Project Carried Out by Volunteers. Available from `http://www.openvas.org/`, Sept. 2012.

[OSVDB13] Open Source Vulnerability Database. OSVDB Documentation. Available from `http://osvdb.org`, May 2013.

[PB05] A. Pandya and P. Bhattacharyay. Text Similarity Measurement Using Concept Representation of Texts. In *Proceedings of the 1st International Conference on Pattern Recognition and Machine Intelligence*, Kolkata, India, Dec. 2005.

[PDR12] N. Poolsappasit, R. Dewri, and I. Ray. Dynamic Security Risk Management Using Bayesian Attack Graphs. *IEEE Transactions on Dependable and Secure Computing*, 9(1):61–74, Jan. 2012.

[Pew11] Pew Internet. What Internet Users Do Online — Pew Research Center's Internet & American Life Project. http://www.pewinternet.org/Static-Pages/Trend-Data/Online-Activites-Total.aspx, Oct. 2011.

[Pol05] S. Polepeddi. Aggregating Vulnerability Information from Current White-Hat Databases. *SANS Institute - GlobalInformation Assurance Certification Paper*, Jan. 2005.

[RHR$^+$11] M. Roberts, A. Howe, I. Ray, M. Urbanska, Z.S. Byrne, and J.M. Weidert. Personalized Vulnerability Analysis Through Automated Planning. In *Working Notes of Workshop on Security and Artificial Intelligence*, Barcelona, Catalonia, Spain, July 2011.

[Ros66] I. M. Rosenstock. Why People Use Health Services. *The Milbank Memorial Fund Quarterly*, 44(3):94–127, 1966.

[RP05] I. Ray and N. Poolsapassit. Using Attack Trees to Identify Malicious Attacks from Authorized Insiders. In *Proceedings of the 10th European Symposium on Research in Computer Security*, Milan, Italy, Sept. 2005.

[RSB88] I.M. Rosenstock, V.J. Strecher, and M.H. Becker. Social Learning Theory and the Health Belief Model. *Health Education Quarterly*, 15(2):175–183, 1988.

[Sah05] M. Sahlgren. An Introduction to Random Indexing. In *Proceedings of the Methods and Applications of Semantic Indexing Workshop at the 7th International Conference on Terminology and Knowledge Engineering*, Copenhagen, Denmark, Aug. 2005.

[Sar08] S. Sarawagi. Information Extraction. *Foundations and Trends in Databases*, 1(3):261–377, 2008.

[SBW01] M. A. Sasse, S. Brostoff, and D. Weirich. Transforming the 'Weakest Link' a Human/Computer Interaction Approach to Usable and Effective Security. *BT Technology Journal*, 19:122–131, 2001.

[Sec13] SecurityFocus. BugTraq. Available from `http://www.securityfocus.com`, May 2013.

[SF09] P. Szewczyk and S. Furnell. Assessing the Online Security Awareness of Australian Internet Users. In *Proceedings of the 8th Annual Security Conference Discourses in Security Assurance & Privacy*, Las Vegas, NV, USA, Apr. 2009.

[SHHB00] M. Schumacher, C. Haul, M. Hurler, and A. Buchmann. Data Mining in Vulnerability Databases. *Department of Computer Science, Darmstadt University of Techonology*, 22, 2000.

[SHJ$^+$02] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J.M. Wing. Automated Generation and Analysis of Attack Graphs. In *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, CA, USA, May 2002.

[SM86] G. Salton and M. J. McGill. *Introduction to Modern Information Retrieval.* McGraw-Hill, Inc., 1986.

[SWY75] G. Salton, A. Wong, and C. S. Yang. A Vector Space Model for Automatic Indexing. *Communications of the ACM*, 18(11):613–620, 1975.

[Sym07] Symantec. Symantec Internet Security Threat Report Trends for January–June 07, Sept. 2007. Volume XII.

[Tie05] S. Tierney. *Knowledge Discovery in Cyber Vulnerability Databases.* MS thesis, University of Washington, 2005.

[Uni13] University of Pittsburgh. Semantic Vectors Package. Available from `https://code.google.com/p/semanticvectors`, Aug. 2013.

[URHR12] M. Urbanska, I. Ray, A. Howe, and M. Roberts. Structuring a Vulnerability Description for Comprehensive Single System Security Analysis. In *Rocky Mountain Celebration of Women in Computing*, Fort Collins, CO, USA, Nov. 2012.

[URR$^+$13] M. Urbanska, M. Roberts, I. Ray, A. Howe, and Z. Byrne. Accepting the Inevitable: Factoring the User into Home Computer Security. In *Proceedings of the 3rd ACM Conference on Data and Application Security and Privacy*, San Antonio, TX, USA, Feb. 2013.

[Was10] R. Wash. Folk Models of Home Computer Security. In *Proceedings of the 6th Symposium on Usable Privacy and Security*, Redmond, WA, USA, 2010.

[WC10] D. Widdows and T. Cohen. The Semantic Vectors Package: New Algorithms and Public Tools for Distributional Semantics. In *Proceedings of the 4th IEEE International Conference on Semantic Computing*, Pittsburgh, PA, USA, Sept. 2010.

[WF08] D. Widdows and K. Ferraro. Semantic Vectors: a Scalable Open Source Package and Online Technology Management Application. In *Proceedings of the 6th International Conference on Language Resources and Evaluation*, Marrakech, Morocco, May 2008.

[WIZD04] S.M. Weiss, N. Indurkhya, T. Zhang, and F. Damerau. *Text Mining: Predictive Methods for Analyzing Unstructured Information.* Springer, 2004.

[WLI07] L. Williams, R. Lippmann, and K. Ingols. An Interactive Attack Graph Cascade and Reachability Display. In *Proceedings of the Workshop on Visualization for Computer Security*, Sacramento, CA, USA, Oct. 2007.

[WNJ06] L. Wang, S. Noel, and S. Jajodia. Minimum-Cost Network Hardening Using Attack Graphs. *Computer Communications*, 29(18):3812–3824, 2006.

[ZA01] I. Zukerman and D. W. Albrecht. Predictive Statistical Models for User Modeling. *User Modeling and User-Adapted Interaction*, 11(1–2):5–18, 2001.

Appendix A

Input to the BUA for Three Hypothetical Users' Profiles

Table A.1: Input to the BUA for UserA actions

	user actions					
User Information	ClickOn LinkIn Email	OpenFlash File	Download Applet	Execute Applet	RunJava WebStar-tApp	ReadEmails
RiskTolerance	Low	Low	Low	Low	Low	Low
PerceivedSeverity	Low	Low	Low	Low	Medium	Medium
PBen	High	Medium	Medium	Low	Low	High
PBar	High	High	Medium	Low	High	Medium
SEficacy	Low	Low	Low	Medium	Low	Medium
CtoA	Medium	Medium	Medium	Medium	Low	High
Gender	F	F	F	F	F	F
Age	age50	age50	age50	age50	age50	age50
EduLevel	High	High	High	High	High	High
SocEcon	High	High	High	High	High	High
ExpGood	Low	Low	Low	Medium	Low	Medium
ExpBad	Medium	Low	High	Medium	Low	Low

Table A.2: Input to the BUA for UserB actions.

	user actions							
User Information	ClickOn LinkIn Email	Open Flash File	Download Applet	Execute Applet	LDAP conection	Open PDF	RunJava WebStartApp	Read Emails
Risk Tolerance	Medium	High	High	High	Medium	High	Medium	Low
Perceived Severity	High	Low	High	Medium	Low	Low	Low	Medium
PBen	Medium	Medium	High	Medium	Medium	High	Low	High
PBar	Low	Low	Low	Low	Low	Low	Low	Low
SEficacy	High	High	High	High	High	High	High	High
CtoA	Medium	Medium	Medium	Low	High	Medium	Low	High
Gender	F	F	F	F	F	F		
Age	age2029	age2029	age2029	age2029	age2029	age2029	age2029	age2029
EduLevel	Medium	Medium	Medium	Medium	Medium	Medium	Medium	Medium
SocEcon	Low	Low	Low	Low	Low	Low	Low	Low
ExpGood	Medium	Medium	Low	Medium	Low	Medium	Low	Medium
ExpBad	Medium	Medium	Medium	Medium	Low	Low	Low	Medium

Table A.3: Input to the BUA for UserC actions

	user actions					
User Information	ClickOn LinkIn Email	Open Flash File	Download Applet	Execute Applet	OpenPDF	Read Emails
RiskTolerance	Low	Low	Medium	Low	Medium	Low
PerceivedSeverity	High	High	High	High	Medium	High
PBen	Medium	Medium	Medium	Medium	High	High
PBar	Low	Low	Low	Low	Low	Low
SEficacy	High	High	High	High	High	High
CtoA	Medium	Medium	Medium	Low	Medium	High
Gender	F	F	F	F	F	F
Age	age2029	age2029	age2029	age2029	age2029	age2029
EduLevel	Medium	Medium	Medium	Medium	Medium	Medium
SocEcon	Low	Low	Low	Low	Low	Low
ExpGood	Medium	Medium	Medium	Medium	Medium	High
ExpBad	Medium	Low	Medium	Medium	Low	Medium

Appendix B

CPTs for Instantiated PAG for UserA

Table B.1: System configuration nodes.

(a)

Adobe_Flash 6_0_88_0	
T	0.5
F	0.5

(b)

Sun_JRE 1_4_0_02	
T	0.5
F	0.5

(c)

Acrobat Reader_9_4_1	
T	0.5
F	0.5

(d)

Microsoft Windows XP Professional SP3	
T	0.5
F	0.5

Table B.2: User action nodes.

(a)

UserOpens FlashFile	
T	0.928524
F	0.071476

(b)

User Starts JavaWebStartApp	
T	0.867202
F	0.132798

(c)

User Using Ldap Connection	
T	0.015
F	0.985

(d)

User Loads PdfDocument	
T	0.015
F	0.985

(e)

User read emails	
T	0.967239
F	0.032761

(f)

User click on link	
T	0.918919
F	0.081081

(g)

User download an untrusted applet or application	
T	0.913726
F	0.086274

(h)

User executed a specially crafted applet	
T	0.949352
F	0.050648

Table B.3: Attacker action nodes.

(a)

AttackAction FlashFileCompromised	
T	0.86
F	0.14

(b)

Attacker JavaAppWith Long Vm Argument	
T	0.99
F	0.01

(c)

Attacker Action	
T	0.99
F	0.01

(d)

Attacker PdfCompromised	
T	0.86
F	0.14

(e)

Attacker has send phisheing email	
T	0.86
F	0.14

(f)

Attacker Action2	
T	0.99
F	0.01

(g)

Attacker Action3	
T	0.99
F	0.01

Table B.4: Vulnerability present in user's computer nodes.

(a) CVE-2010-0187 AdobeFlash

CVE-2010-0187 AdobeFlash	Adobe_Flash 6_0_88_0	
	T	F
T	0.85888	0
F	0.14112	1

(b) CVE-2008-3111 SunJavaMultiple_10

CVE-2008-3111 SunJava Multiple_10	Sun_JRE 1_4_0_02	
	T	F
T	0.99968	0
F	0.00032	1

(c) CVE-2009-1094 JavaCPU

CVE-2009-1094 JavaCPU	Sun_JRE 1_4_0_02	
	T	F
T	0.99968	0
F	0.00032	1

(d) CVE-2010-4091 OS

CVE-2010-4091 OS	AcrobatReader 9_4_1	
	T	F
T	0.85888	0
F	0.14112	1

(e) Cumulative Security Update of ActiveX Kill Bits CVE-2010-0811

Cumulative Security Update of ActiveX Kill Bits CVE-2010-0811	Microsoft Windows XP Professional SP3	
	T	F
T	0.85888	0
F	0.14112	1

(f) JRE Virtual Machine Vulnerabilities CVE-2008-3107

JRE Virtual Machine Vulnerabilities CVE-2008-3107	Sun_JRE 1_4_0_02	
	T	F
T	0.99968	0
F	0.00032	1

Table B.4: Contd.

(g) JRE Font Processing Vulnerability CVE-2008-3108

JRE Font Processing Vulnerability CVE-2008-3108	Sun_JRE 1_4_0_02	
	T	F
T	0.99968	0
F	0.00032	1

Table B.5: Vulnerability exploited nodes.

(a) CVE-2010-0187 Exploited

	CVE-2010-0187 AdobeFlash	T				F			
	AttackAction FlashFileCompromised	T		F		T		F	
	UserOpensFlashFile	T	F	T	F	T	F	T	F
CVE-2010-0187 Exploited	T	0.43	0	0	0	0	0	0	0
	F	0.57	1	1	1	1	1	1	1

(b) CVE-2008-3111 Exploited

	CVE-2008-3111 SunJavaMultiple_10	T				F			
	Attacker JavaAppWithLongVmArgument	T		F		T		F	
	User Starts JavaWebStartApp	T	F	T	F	T	F	T	F
CVE-2008-3111 Exploited	T	0.999	0	0	0	0	0	0	0
	F	0.001	1	1	1	1	1	1	1

(c) CVE-2009-1094 Exploited

	CVE-2009-1094 JavaCPU	T				F			
	User Using Ldap Connection	T		F		T		F	
	AttackerAction	T	F	T	F	T	F	T	F
CVE-2009-1094 Exploited	T	0.99	0	0	0	0	0	0	0
	F	0.01	1	1	1	1	1	1	1

(d) CVE-2010-4091 Exploited

	CVE-2010-4091 OS	T				F			
	Attacker PdfCompromised	T		F		T		F	
	User Loads PdfDocument	T	F	T	F	T	F	T	F
CVE-2010-4091 Exploited	T	0.93	0	0	0	0	0	0	0
	F	0.07	1	1	1	1	1	1	1

Table B.5: Contd.

(e) CVE-2010-0811 Exploited

	Cumulative Security Update of ActiveX Kill Bits CVE-2010-0811	T				F			
	Link to crafted website is presented	T		F		T		F	
	user click on link	T	F	T	F	T	F	T	F
CVE-2010-0811 Exploited	T	0.93	0	0	0	0	0	0	0
	F	0.07	1	1	1	1	1	1	1

(f) CVE-2008-3107 Exploited

	User download an untrusted applet or application	T				F			
	JRE Virtual Machine Vulnerabilities CVE-2008-3107	T		F		T		F	
	Attacker Action2	T	F	T	F	T	F	T	F
CVE-2008-3107 Exploited	T	0.99	0	0	0	0	0	0	0
	F	0.01	1	1	1	1	1	1	1

(g) CVE-2008-3108 Exploited

	User executed a specially crafted applet	T				F			
	JRE Font Processing Vulnerability CVE-2008-3108	T		F		T		F	
	AttackerAction3	T	F	T	F	T	F	T	F
CVE-2008-3108 Exploited	T	0.99	0	0	0	0	0	0	0
	F	0.01	1	1	1	1	1	1	1

Table B.6: Exploit nodes.

(a) DenialOfService

	Parent	CVE-2010-0187 Exploited	CVE-2008-3111 Exploited	CVE-2009-1094 Exploited	CVE-2010-4091 Exploited	LEAK
	State	T	T	T	T	
Denial Of Service	T	0.29	0.99	0.99	0.99	0.5
	F	0.71	0.01	0.01	0.01	0.5

(b) Root Access Privilege

	CVE-2010-0811 Exploited	T	F
Root Access Privilege	T	0.99	0
	F	0.01	1

(c) Authentication Bypass

	CVE-2009-1094 Exploited	T	F
Authentication Bypass	T	0.99	0
	F	0.01	1

Table B.6: Contd.

(d) User Access Privilege

	Parent	CVE-2008-3107 Exploited	CVE-2008-3108 Exploited	LEAK
	State	T	T	
User Access Privilege	T	0.99	0.99	0.5
	F	0.01	0.01	0.5

(e) Arbitrary code execution

	Parent	CVE-2010-0811 Exploited	CVE-2008-3107 Exploited	CVE-2008-3108 Exploited	LEAK
	State	T	T	T	
Arbitrary code execution	T	0.99	0.99	0.99	0.5
	F	0.01	0.01	0.01	0.5

(f) Unauthorized Modification

	Parent	CVE-2008-3107 Exploited	CVE-2008-3108 Exploited	LEAK
	State	T	T	
Unauthorized Modification	T	0.99	0.99	0.5
	F	0.01	0.01	0.5

Table B.7: System compromised node.

	Parent	Denial Of Service	Authentication Bypass	Root Access Privilege	User Access Privilege	Arbitrary code execution	Unauthorized Modification	LEAK
	State	T	T	T	T	T	T	
System Compromised	T	1	1	1	1	1	1	0.5
	F	0	0	0	0	0	0	0.5