Password Policy Simulation and Analysis

Richard J. K. Shay
Purdue University
West Lafayette, IN
rshay@cs.purdue.edu

Abhilasha Bhargav-Spantzel
CERIAS
Purdue University
West Lafayette, IN
bhargav@cs.purdue.edu

Elisa Bertino
Purdue University
West Lafayette, IN
bertino@cs.purdue.edu

ABSTRACT

Passwords are an ubiquitous and critical component of many security systems. As the information and access guarded by passwords become more necessary, we become ever more dependent upon the security passwords provide. The creation and management of passwords is crucial, and for this we must develop and deploy password policies. This paper focuses on defining and modeling password policies for the entire password policy lifecycle. The paper first discusses a language for specifying password policies. Then, a simulation model is presented with a comprehensive set of variables and the algorithm for simulating a password policy and its impact. Finally, the paper presents several simulation results using the password policy simulation tool.

Categories and Subject Descriptors: K.6.5 Computing Milieux-Management of Computing and Information Systems[Security and protection]

General Terms: Management, Security, Standardization

Keywords: Password, Policy, Management, Modeling, Simulation

1. INTRODUCTION

The ubiquity of passwords is a fact of the present age. Authentication is usually executed by using the combination of a user name and password [12]. Thus, passwords are frequently the sole barrier between a potential attacker and a victim's information. Knowing a person's password allows an attacker to impersonate that person in an online setting, or access sensitive data intended only for that person. As society becomes increasingly dependent on passwords for security, it also becomes vulnerable to those passwords becoming compromised.

Previous work has shown that users, given the option, tend to create simple passwords [13, 8, 3]. However, simple passwords are especially vulnerable to attackers [13, 12]. Therefore, many groups and organizations develop password policies which impose restrictions on the passwords a user may make, and how those passwords are used. A well-written policy may increase an organization's security [5, 8, 12, 2].

DIM'07, November 2, 2007, Fairfax, Virginia, USA.

A large portion of password policies is usually related to the creation of passwords. For example, a password creation policy may require that passwords be at least six characters long and contain at least one numeric character. There are, however, several other facets in a password lifecycle for which password policies are relevant.

In the design of a password policy, it is crucial that human factors be considered in addition to technical factors. While a password policy may specify the encryption to be used on the password, an overly complex password may be written down on paper by its users because they fail to memorize it [8]. Likewise, many password policies specify with whom a user may share passwords, and under what circumstances an administrator is to be contacted.

Password policy creation today appears to be more guesswork and heuristic than science or calculation. There are many organizations using passwords for security, but no widely-accepted unified context under which all of those password policies may be understood and compared. Such a unified context would enable both the creation of better password policies and a better understanding of password policies. The goal of this paper is to introduce such a context through a password policy language and model, and to present experimental results derived from a simulation of that model.

The primary contributions of this paper are a password policy lifecycle, a password policy language, a simulation model for password policies, and a set of experimental results from that simulation model. The password lifecycle discussion in this paper provides a framework in which password policies may be better understood; by studying the situations in which password policy use arises, we can better understand the vital role password policies play in the security of an organization. The availability of a language for password policies may help in more precisely considering, describing, and differentiating password policies. The password policy simulation model, and the password policy simulation implementation, offer a tool to assist in a more rigorous approach to password policy creation. An administrator wishing to create a new policy may use the simulation to compare different potential password policies before actually issuing a policy. Because the simulator uses a large number of factors, including both technical and human factors, the simulation tool is able to fit many different organizational structures with differing assumptions about users. Beyond being used to assist in the creation of specific password policies, this simulator also provides a utility for comparing how different factors impact the effectiveness of passwords. analyze and implement password policies.

The rest of this paper is organized as follows. Section 2 presents the password lifecycle and a comprehensive language to describe password policies. Section 3 then presents the data used in the password policy simulation model. The simulation algorithm is

elaborated upon in Section 4. In Section 5, we describe an implementation of the simulation model and report some experimental results. This is followed by a discussion on the limitations of the simulation model in Section 6. In Section 7 the related works are discussed. Finally, in Section 8 we present some conclusions.

2. PASSWORD LIFECYCLE AND POLICIES

In this section we first introduce a reference password lifecycle and then formally define a language to express the password policies relevant at every stage of this lifecycle.

2.1 Password Lifecycle

The lifecycle of a password is organized into four stages: *creation, storage and memorization, usage,* and *deletion*. One or more of these stages may be influenced by a password policy. A password policy is a set of rules regarding the creation, use, and deletion of a password. In what follows we describe each stage of the password lifecycle and examples of password policies which may be relevant to that stage.

Password Creation. A password is created in accordance with the stipulations of a password policy. Such policy often ensures that the password is complex enough to present an obstacle to potential attackers. The policy of the University of Massachusetts at Amherst, for example, requires that a password have certain numbers of letters, numbers, and non-alphanumeric symbols [11]. The policy also sets the minimum password length to six characters, and the maximum length to eight.

Password Storage and Memorization. Once a password has been created, it must be memorized or otherwise stored by the user. The computer system must also retain the password to be able to authenticate the user. As an example, a password policy could require that all Unix passwords stored on a server be hashed in a particular type of database. The Boston University password policy is explicit about passwords needing to be stored encrypted. "Passwords must never be contained in a non-encrypted form on the system, even in a protected file ... Whenever possible, encrypted passwords should be kept in a protected file [9]". There could also be policies which restrict the user from storing passwords in a particular manner. For example, the Brown University password policy explicitly prohibits users from writing down their passwords [10].

Password Use. A password policy may provide users with instructions regarding the handling of a password which has already been created and memorized. For example, the Brown University password policy asks users to email their passwords only with administrative permission [10]. Brown University policy also requires that the users' passwords be encrypted when being transmitted. Another related policy relevant to many institutions is a restriction on the sharing of passwords.

Password Deletion. The final stage in the password lifecycle is deletion; this stage may also be influenced by password policy. Deletion results in the password becoming unusable. Passwords may be deleted for one of two reasons – expiration or revocation. Many password policies limit the temporal duration of a password. A password which has exceeded this limit is said to have *expired*. The password policy of Pennsylvania State University, for example, makes passwords expire after one year. Passwords may be *revoked* when the user or administrator believes the password has been compromised. For example, the Brown University password policy asks users to request new passwords if they suspect their passwords are compromised [10].

2.2 Password Policy Language

In this section we discuss the formal language for the specification of password policies. We derive the password policy language from a generic authentication policy language [7]. We begin by introducing the notation and symbols to be used for policy specification and then illustrate the syntax of the language.

2.2.1 Preliminary Notion

Constant Symbols.

All users ($\mathcal{U}$) is the universal set of all possible users.
Users ($U \subseteq \mathcal{U}$) is the set of users registered to the system.
Passwords ($\mathcal{P}$) denotes the set of passwords in the system. Each password is associated with a user.
Storage ($\mathcal{S}$) is the set of types of storage available in the system to store passwords. The type of storage determines how stored passwords are encoded.
Verifier ($\mathcal{V}$) is the set of verifiers whose responsibilities may include issuing, establishing, and verifying passwords in the system.
Time ($\mathbb{T}$) is the discrete time in the system.
Natural Numbers ($\mathbb{N}$).

Variable and Term Symbols. Variable symbols corresponding to the above constant symbols are denoted by $V_{\mathcal{U}}, V_U\ V_{\mathcal{P}}, V_{\mathcal{S}}$, $V_{\mathcal{V}}, V_{\mathbb{T}}$, and $V_{\mathbb{N}}$ respectively. The variable symbols are placeholders in policy statements for their corresponding types of constant symbols.

The set of user terms, denoted by UT, is equal to the set $V_U \cup U \cup V_{\mathcal{U}} \cup \mathcal{U}$. Similarly, we have password terms $\mathcal{P}T$, storage terms $\mathcal{S}T$, verifying parties terms $\mathcal{V}T$, terms denoting time $\mathbb{T}T$, and number terms $\mathbb{N}T$.

Assertion Symbols. The assertions we provide are divided into three distinct sets. *System state* assertions ($\mathcal{SA}$) determine the system configuration factors involved in password use. Specifications of *password characteristics* ($\mathcal{PC}$) qualify the passwords in the system. Specifications of *user characteristics* ($\mathcal{UC}$) qualify the user behavior in the system.

2.2.2 Formal Definitions

Password policies are key elements in decisions made at each step of the password lifecycle. The specification of password policies rely on the notion of *password policy factors*. Password policy factors define the conditions satisfied at each stage of the password lifecycle. Password policy factors are described in terms of *descriptors* [7] which are instantiations of various assertions relevant to password management.

DEFINITION 2.1. **(Descriptor)** *A descriptor* d *is a predicate of the form* $p(x, \mathbf{t})$*, in which* x *is a variable, and* $\mathbf{t}$ *is a vector of one or more terms.*

Properties of a specific password policy may be described in multiple ways. In our language, we represent them through a finite set of descriptors to enable the specification of expressive password policies[1]. The arguments of the descriptors are typically simple terms from $\mathcal{P}T$, $\mathcal{S}T$, UT, $\mathbb{N}T$, and $\mathbb{T}T$. Relevant descriptors necessary to express articulated policy conditions are listed in Table 1. Password policy factors are specified through a Boolean conjunction of descriptors. A formal definition is given as follows.

DEFINITION 2.2. **(Password policy factor)** *A password policy factor is a Boolean conjunction of descriptors* $d_1, \ldots, d_k$*, each of the form* $d = p(x, \mathbf{t})$*, satisfying the following condition: the same*

[1]We limit ourselves to the listed descriptors because we identify them as significant for expressing policy language without an overly-complex grammar. Additional descriptors may be envisioned in an extended version of the current language.

Type	Descriptor	Arity	Argument Types	Meaning
$\mathcal{SA}$	sys_NumUsr	2	$\mathcal{PT}, \mathbb{NT}$	If $sys\_NumUsr(p, n)$ is true, then the system where password p is stored has n users
$\mathcal{SA}$	sys_Harm	3	$\mathcal{PT}, \mathbb{NT}, \mathbb{TT}$	If $sys\_Harm(p, x, t)$ is true, then at time t for the system where password p is stored, the system harm is x
$\mathcal{SA}$	$sys\_NumAttacks$	2	$\mathcal{PT}, \mathbb{NT}$	If $sys\_NumAttacks(p, y)$ is true, then the system where password p is stored has on an average y number of attacks per discrete time unit.
$\mathcal{PC}$	$pass\_Entropy$	2	$\mathcal{PT}, \mathbb{NT}$	If $pass\_Entropy(p, e)$ is true, then the average per-character entropy of the given password p is e.
$\mathcal{PC}$	$pass\_expTime$	2	$\mathcal{PT}, \mathbb{TT}$	If $pass\_expTime(p, t)$ is true, then password p expires after t discrete time units.
$\mathcal{PC}$	$pass\_creaTime$	2	$\mathcal{PT}, \mathbb{TT}$	If $pass\_creaTime(p, t)$ is true, then time when password p was created is t; t is measured in discrete time units.
$\mathcal{PC}$	$pass\_length$	2	$\mathcal{PT}, \mathbb{NT}$	If $pass\_length(p, n)$ is true, then the given password p has a length of n characters.
$\mathcal{PC}$	$pass\_store$	2	$\mathcal{PT}, \mathcal{ST}$	If $pass\_store(p, s)$ is true, then the given password p is stored in storage type s.
$\mathcal{UC}$	$user\_forgetfulness$	2	$UT, \mathbb{NT}$	If $user\_forgetfulness(u, m)$ is true, then user u has *forgetfulness* level m. m is a real-value probability representing how likely user u is to be unable to remember a seven-digit number when first given that number.
$\mathcal{UC}$	$user\_isMalicious$	2	$UT, \mathbb{NT}$	If $user\_isMalicious(u, mal)$ is true, then the given user u is assumed to be malicious if and only if mal has a value of *true*.
$\mathcal{UC}$	$user\_isCompromised$	2	$UT, \mathbb{NT}$	If $user\_isCompromised(u, com)$ is true, then user u is compromised if and only if com has a value of *true*. A user u is compromised when there is a malicious agent who knows the password of u.

Table 1: Password Policy Descriptors

factor variable x appears in every descriptor $d_m = p(x, \mathbf{t}_m)\ \forall m \in [1, k]$.

As from Definition 2.2, password policy factors can be defined using any possible combination of descriptors.

EXAMPLE 1. *Examples of password policy factors are the following:*
1) $pass\_Entropy(p, .6) \wedge pass\_length(p, 6) \wedge pass\_expTime(p, \text{“30 days”})$,
2) $sys\_NumUsr(p, 42) \wedge pass\_store(p, s(\text{“written”}, \text{“cleartext”}))$

The password policy factors, as defined, are independent in that the specification of one single factor is not related to that of any other factor. We formalize the concept of password policy with respect to the password lifecycle as follows.

DEFINITION 2.3. **(Password Policy)** *A password policy* p *is a tuple of the form* $\langle [f_1, \ldots, f_k], Ts \rangle, k \geq 1$, *where:*

- $[f_1, \ldots, f_k]$ *is a list of password policy factors, such that* $f_j \neq f_m$ *if* $j \neq m$
- *Ts denotes the number of mandatory password policy factors to be verified; thus* $1 \leq Ts \leq k$.

A password policy is in part specified by a combination of factors to be evaluated. The verification of all the factors may or may not all be mandatory, as specified by threshold value, denoted by *Ts*.

EXAMPLE 2. *The following is an example of a password policy:*
$p = \langle [f_1, f_2], 2 \rangle$ *states that the password policy should enforce both factors f_1 and f_2. Here, f_1 and f_2 correspond to the factors in Example 1.*

The specification of the mandatory number of factors enhance the flexibility and expressive power of the policy language.

3. SIMULATION MODEL DATA

This section defines a simulation model which can be expressed based on the above-defined password policy language. This model simulates a computer system which can have multiple password policies. The simulation model, and the simulation itself, serve two purposes. First, they enable different password policies to be examined, as is done in Section 5. Second, the simulation provides a utility for network administrators and others responsible for password policy creation. The simulator enables new password policies to be tested by simulation before being deployed in practice.

The simulation model is divided into two parts: the model variables and the model algorithm. This is illustrated in Figure 1. This section describes and explains the model variables. The next section in this paper, Section 4, describes the algorithm in which those variables are used. Most variables in the password policy language are mapped to configuration variables in the simulation model. The simulation models a single system, its users, and the password policies.

Model variables are divided into four types: system variables, password variables, user description variables, and simulation termination variables. Some of the variables are set by the entity running the simulation, to whom we refer as the *administrator*; while the others are derived by the *simulation algorithm* specified in Section 4. The administrator specifies the password variables and the simulation termination variables. These do not change during the simulation. The password variables specify details of the password policies and the simulation termination variables specify the conditions under which the simulation should terminate.

The simulation derives both the individual user and system variables from its policies. An example of a user variable is a Boolean variable indicating whether one particular user is compromised or not. Similarly, the amount of harm done to the system by malicious users is represented as a system variable. These values may change during the simulation. Table 2 describes all of the variables in the simulation, their type, and the password policy language assertions from which they are derived.

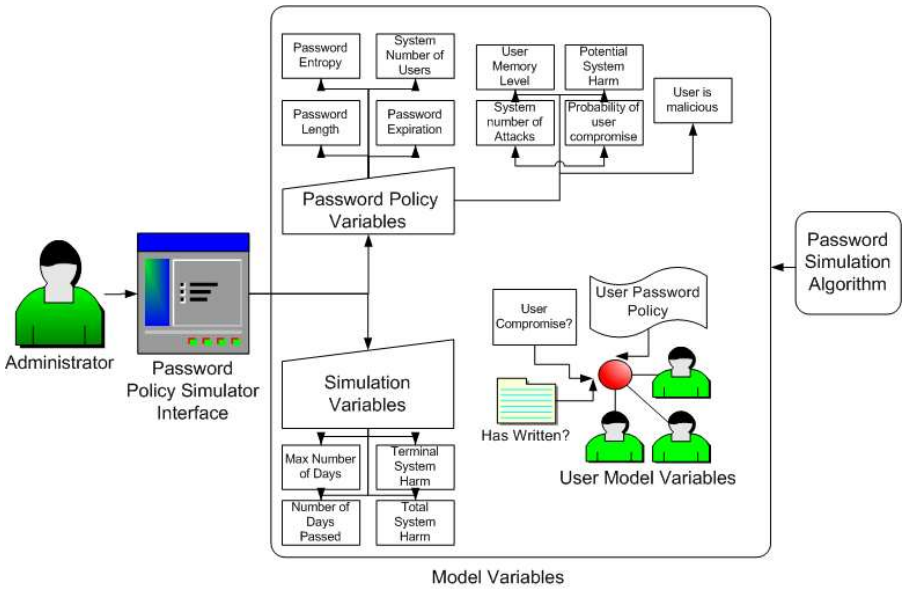

Figure 1: The password policy model and data

3.1 Password Policy Variables

A password policy in the system can be described by the password policy language presented in Section 2.2. Each of these variables must be set by the administrator prior to the start of the simulation.

A system policy is defined as follows: $\Pi = \{forget, potHarm, isMalicious, numAttacks, compWritten, numUsr, passExpire, passLength, passEntropy\}$

Multiple password policies may coexist within a system.

3.1.1 Password Creation and Deletion

$numUsr$ is the number of individual computer system users who follow password policy $User_{Policy}$. $passExpire$ is the number of days after which a password is to be changed; after a password has existed for $passExpire$ days, it expires. $passLength$ is the minimum length of a password, in characters. $passEntropy$ is the minimum per-character entropy required of a password.

Section 7 indicates that increased complexity in passwords can lead to better security for those passwords. Following the work of Shannon [6], this model uses the concept of entropy to represent complexity in text. More specifically, $passEntropy$ is a decimal number representing the average number of bits required to represent each character in a password of policy $User_{Policy}$.

3.1.2 Human Memory and Password Storage

Limitations of human memory can lead users to select simple passwords over more secure options [2]. The model deals with a very specific case of the storage types that can be described with the $pass\_store$ password policy descriptor. The model represents limited human memory capacity in how a password is stored: a user who has not yet memorized his or her password is assumed to write down that password near his or her computer.

Some users may be better at memorizing passwords than others. A user's ability to memorize a password is a function of three variables: the password length $passLength$, the password per-letter entropy $passEntropy$, and the user's forgetfulness $forget$. $forget$ is a decimal value between 0.0 and 1.0 which is the probability that a user is unable to memorize a password composed of seven digits. Having the administrator enter a user's memory in this way has two advantages. First, it lets a user's ability to memorize a password vary with the complexity of that password. Second, it gives the administrator an intuitive way to think about user memory; seven digits is a local telephone number, so the value may be considered how likely a user is to forget a local phone number when first hearing that number.

3.1.3 System Harm and Compromised Users

This model represents the harm done to the computer system by all sources as a single integral value Σ_{harm}. While the above-mentioned variables exist for each user or for each policy, a single instance of the Σ_{harm} variable exists for the entire simulation. This representation of total system harm as an integer is a general solution which makes no underlying assumptions about the computer system itself. This concept facilitates quick and clear comparison between two different password policies.

Individual password policy variables, set by the person using the simulation, contribute to the final value of Σ_{harm}. In any system some users being compromised may be more harmful to the system than others. This is represented by the policy variable $potHarm$; the higher the value of $potHarm$, the greater the damage dealt to the system by a compromised user of that policy. If an individual user U ends a day compromised then the value of $potHarm$ specified by that user's password policy $User_{Policy}$ is added to the present value of Σ_{harm}.

There are three ways in which a user may become compromised: the user may be malicious and therefore always be considered compromised, the user's password may be guessed in an attack, or the user may be compromised because his or her password is written down. A compromised user is a user whose password is known by a malicious agent. Therefore, a malicious user is always considered compromised. A user may become compromised as the result of an attack against that user's password; each user is subject to $numAttacks$ attacks per day of the simulation, as described in Section 4. Should a user not remember his or her password, that user writes the password down, leading to an additional daily probability of becoming compromised, as set by $compWritten$. A

Variable Name	Type	Admin-Specified?	Matching Language Assertion	Description
Π	Policy	Yes	NONE	A Password Policy
$forget$	Policy	Yes	user_forgetfulness	User forgetfulness
$potHarm$	Policy	Yes	sys_Harm	Potential system harm
$isMalicious$	Policy	Yes	user_isMalicious	Is the user malicious?
$numAttacks$	Policy	Yes	sys_NumAttacks	Daily number of attacks against user
$compWritten$	Policy	Yes	user_isCompromised	Daily probability user compromised, if password written
$numUsr$	Policy	Yes	sys_NumUsr	Number of group members
$passExpire$	Policy	Yes	pass_expTime	Password Expiration
$passLength$	Policy	Yes	pass_length	Password length
$passEntropy$	Policy	Yes	pass_Entropy	Per-character password entropy
Ω_{days}	Termination	Yes	NONE	Simulation ends after this many days
Ω_{harm}	Termination	Yes	NONE	Simulation ends after this much harm inflicted
Σ_{harm}	System	No	NONE	Total system harm in the simulation
Σ_{days}	System	No	$\mathbb{T}$	Number of days passed in the simulation
$User_{isWritten}$	User	No	pass_store	Is the user's password written down?
$User_{isCompromised}$	User	No	user_isCompromised	Is the user compromised?
$User_{Policy}$	User	No	NONE	User's Password Policy, derived from the policies

Table 2: Model Variables — note that a separate instance of the Policy variables exists for each different password policy represnted in the system

nonmalicious user who is compromised becomes uncompromised when assigned a new password.

3.2 Simulation Variables

The current day of the simulation is Σ_{days}. The administrator sets at least one of two possible simulation termination variables which set the conditions for the simulation to terminate. At least one of these two variables, Ω_{days} and Ω_{harm}, must be nonzero. If Ω_{days} is nonzero, the simulation terminates once Ω_{days} days have passed. If Ω_{harm} is nonzero, the simulation terminates after the day in which the total amount of system harm Σ_{harm} reaches or exceeds the value of Ω_{harm}.

3.3 Modeling System Users

Individual users are not specified directly by the administrator, but instead are created and maintained by the simulation algorithm. An individual user U is described as a set of three user attributes.
$U = \{User_{isWritten}, User_{isCompromised}, User_{Policy}\}$
Each user has exactly one password policy to which it is bound. Each user also has two Boolean variables. $User_{isCompromised}$ is a dynamic value which is either $false$ or $true$; if $true$, then the user is compromised. $User_{isWritten}$ is a dynamic value which is either $false$ or $true$; if $true$, then the user has his or her password written unencrypted.

4. SIMULATION MODEL ALGORITHM

The simulation model algorithm simulates the actions and events of the computer system described by the variables listed in Section 3. Each day in the simulation is divided into five sections. First, users who need new passwords are assigned new passwords and try to memorize them. Then, the users undergo a number of attacks and may become compromised. Next, users who have not yet memorized their passwords try to memorize them again. Fourth, the simulation calculates the harm done to the system by compromised users. Finally, the algorithm checks whether to continue the simulation or not.

4.1 Memorizing a Password and Cracking a Password

We assume that a user creates a password as long as the minimum required length, and with a per-character entropy equal to the minimum required per-character entropy. Previous work, discussed in Section 7, indicates that users are often not vigilant about password security [3, 8].

Each password in this model has a specific complexity derived from the password policy under which it is created, specifically by its length $passLength$and its per-character entropy $passEntropy$. A password created under a policy $User_{Policy}$ is uniquely determined by ($passLength \times passEntropy$) bits. This value is used in the model algorithm both in determining if a user memorizes a password and in determining if a user is compromised by a brute force attack.

$forget$ is the probability that a given user of password policy $User_{Policy}$ is unable to memorize a seven-digit number. The entropy of a seven digit number is $log_2 10^7$. The model assumes that the ability of a user to memorize a set of characters grows and shrinks proportionately with the total entropy of those characters. This gives us the following equation:
$\frac{Probability\ of\ failure\ to\ memorize\ 7-digit\ number}{Complexity\ of\ 7-digit\ number} = \frac{Probability\ of\ failure\ to\ memorize\ password}{Complexity\ of\ password}$
This, in turn, gives us the following, in which P is the probability of the user failing to memorize his or her password on the first try.
$P = \frac{forget * passLength * passEntropy}{7 * log_2 10}$
Based on the equation, if either the value of $forget$ for a user increases, or the complexity of the user's password increases, that user becomes less likely to memorize a password on the first try.

A password is either memorized by its user or written down on a piece of paper. Once a user has memorized a password, that user is assumed never to forget it, and it is never written down. However, if the user has not yet memorized his or her password, then that user tries to memorize it again each day. The model assumes that a user becomes more likely to memorize a password the longer he or she uses it. Each user with a written password is subjected to an additional daily probability of becoming compromised, as set by $compWritten$. This represents the added danger a user encounters when his or her password is written down on a piece of paper or stored in some other insecure manner.

Increased password complexity makes it more difficult for an attacker to guess a password through a brute force attack [8]. Each password, as described above, is uniquely determined by ($passLength \times passEntropy$) bits. Our model assumes that a potential attacker is aware of the number of bits needed to determine a partic-

ular password, but is unaware of what those bits are. Therefore, the probability of a single brute force attack, or guess, by an attacker succeeding is given by the following expression.
$2^{-1 * passLength * passEntropy}$
Each user is subject to a daily number of such attacks determined by that user's password policy, $numAttacks$.

4.2 The Model Algorithm

The password policy simulation algorithm is specified in Algorithm 1.

Algorithm 1 Password Policy Simulation Algorithm

Require: Static model configuration variables set
1: $\Sigma_{days} \leftarrow 0$
2: $\Sigma_{harm} \leftarrow 0$
3: **for all** user U **do**
4: **if** (Σ_{days}= 0) OR (Σ_{days}% $passExpire$= 0) **then**
5: **if** $isMalicious$= 1 **then**
6: $User_{isCompromised} \leftarrow 1$
7: **else**
8: $User_{isCompromised} \leftarrow 0$
9: **end if**
10: UATTEMPTS TO MEMORIZE PASSWORD
11: **if** U FAILS TO MEMORIZE **then**
12: $User_{isWritten} \leftarrow 1$
13: **else**
14: $User_{isWritten} \leftarrow 0$
15: **end if**
16: **end if**
17: **for** $numAttacks$ **do**
18: **if** BRUTE FORCE ATTACK SUCCEEDS **then**
19: $User_{isCompromised} \leftarrow 1$
20: **end if**
21: **end for**
22: **if** $User_{isWritten}$ **then**
23: WITH PROBABILITY $compWritten$: $User_{isCompromised} \leftarrow 1$
24: **end if**
25: **if** $User_{isWritten}$ **then**
26: **for** NUMBER OF DAYS PASSWORD HAS ALREADY BEEN WRITTEN DOWN **do**
27: **if** U IS ABLE TO MEMORIZE PASSWORD **then**
28: $User_{isWritten} \leftarrow 0$
29: **end if**
30: **end for**
31: **end if**
32: **if** $User_{isCompromised}$ **then**
33: $\Sigma_{harm} \leftarrow \Sigma_{harm} + potHarm$
34: **end if**
35: **end for**
36: $\Sigma_{days} \leftarrow \Sigma_{days} + 1$
37: **if** ($\Sigma_{days} = \Omega_{days}$ AND $\Omega_{days} \neq 0$) OR ($\Sigma_{harm} \geq \Omega_{harm}$ AND $\Omega_{harm} \neq 0$)) **then**
38: (STOP)
39: **else**
40: GOTO 3
41: **end if**

While the simulation is running, a number of metrics may be recorded as its output. The duration of the simulation in days (final value of Σ_{days}) and the total damage to the system (final value of Σ_{harm}) may be recorded. In addition, a number of metrics may be recorded on a per-policy basis, as shown in Table 3.

Per-Policy Simulation Results
Number of passwords assigned
Number of times a user wrote down a password
How many times a user ended a day with written password
How many times a user ended a day compromised
System harm (Σ_{harm}) done by users
Average number of new passwords assigned to a user
Average number of times a user wrote down a password
Average number of days a user ended with written password
Average number of days a user finished compromised
Average amount of system harm (Σ_{harm}) done by a user
Percent of total system harm for which users are responsible

Table 3: Per-Policy Simulation Results

Password Policy Variable	**User Memory**	**Password Entropy**	**Password Frequency**
$passLength_i$	5	8	5
$forget_i$	*variable*	.1	0.1
$passEntropy_i$	3.0	*variable*	1.0
$passExpire_i$	60	20	*variable*
$potHarm_i$	1	1	1
$isMalicious_i$	0	0	0
$numAttacks_i$	10	10	10
$compWritten_i$	0.10	0.20	0.01
$numUsr_i$	100	100	1000

Table 4: Simulation Experiments' Data Values

5. EXPERIMENTS AND RESULTS

In this section we first provide some information about the implementation of the model; we then report results from three experiments carried out using the simulator. The first experiment explores user memory, the second analyzes password entropy, and the third studies password change frequency. The input to each experiment may be found in Table 4. In order to elucidate the model and its components, the first experiment is explained in rigorous detail. The model is designed to be open-ended; the examples given here are only three of many possible experiments.

5.1 Implementation

The password policy simulation model is implemented as a program in Java version 1.5.0_06. The Java random number function is used, with a random number seed provided by the user. A graphical user interface allows the user to enter data and assists the user in understanding the simulation results. The implementation runs quickly. On a Windows XP machine with a 2.80GHz Pentium CPU and 1.0 GB of RAM, all nine simulations used in the first experiment took a total of only 3048 milliseconds to execute. Images from the implementation may be found in the Appendix.

5.2 User Memory Experiment

This experiment deals with user memory and its impact on system harm. It includes a careful explanation of the results to give the reader a better understanding of the model. The simulation is run nine different times for this experiment, each time with one policy in the simulation. Let us call the nine different different policies simulated π_1 through π_9. These nine password policies are identical except that they have different levels of forgetfulness.

Information which is the same for all π_i may be found in Table 4 under "User Memory." The nine different password policies differ in their forgetfulness $forget$. For each π_i, $forget_i = \frac{i}{10}$. So, for example, π_3 has user forgetfulness of 0.3. In each of the nine instances, the simulation is set to terminate after 365 days have passed.

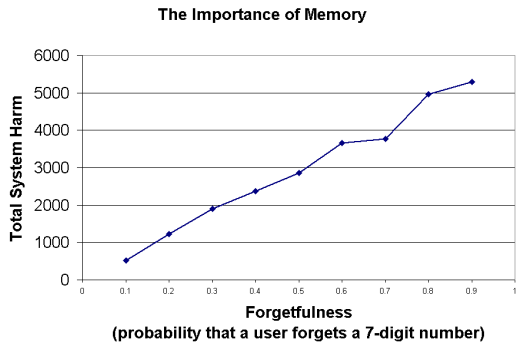

Figure 2: Comparison of Σ_{harm} versus $forget$.

The results for this experiment are found in Figure 2. In the figure, each of the nine simulations is represented by a point on the graph. The $x - axis$ represents the forgetfulness of the users of that particular policy; the further to the right the policy, the worse its users memory. Likewise, the $y - axis$ represents the amount of harm done to the system by the users of the different password policies; the higher a policy, the more system harm it inflicted. It is clear that everything else being equal, users with a better memory tend to inflict less harm to the computer system they use. This is because a user with a better memory is less likely to write down his or her password, and writing down a password can lead to a user being compromised.

5.2.1 Model Explanation Variables

In order to demonstrate the algorithm employed by this model, we focus on a specific simulation from the above-described experiment. In particular, we consider one specific password policy, π_3. We consider a specific user in this group, to whom we refer as *Plato* and describe his life in the simulation. Because *Plato* follows the password policy π_3, we know that *Plato* has certain values. These are values derived from his password policy, and they never change while the simulation is running. π_3 has a total of 100 members, so there are 99 other users who share this policy and its static values with *Plato*.

Table 4 under "User Memory" gives us a number of facts about *Plato*. Each password *Plato* creates has a length of 5 characters. Each character in a password of *Plato* has an average entropy of 3. *Plato* must change his password every 60 days. At the end of each day that *Plato* is compromised, he increases the total value of system harm by 1. *Plato* is not a malicious user. *Plato* is subjected to 10 random brute force attacks each day. Each day that *Plato* ends with his password written down, there is an extra 10% chance that he becomes compromised. We also know that *Plato* has a 70% chance of memorizing a seven digit number when he first sees it, and a 30% chance of failing to do so.

In addition to these static variables, *Plato* has two dynamic variables associated with him: $User_{isWritten}$ and $User_{isCompromised}$. $User_{isWritten}$ is $true$ when *Plato* has written down his password and not disposed of it; for example, if he has his password written on a scrap of paper by his computer. $User_{isCompromised}$ is $true$ when and only when *Plato* is compromised. *Plato* is compromised only when his current password has fallen into the hands of some malicious person.

5.2.2 Getting a New Password

Plato has a $passExpire$ value of 60, meaning that he is given a new password every 60 days. In addition, he gets a new password on the first day of the simulation. When *Plato* is given a new password, the first thing that happens is that he becomes not compromised if he were compromised before; $User_{isCompromised}$ takes on a value of $false$. This is because any malicious agent who used to know his old password does not know his new one.

When *Plato* gets his new password, the first thing that he does is to try to memorize it. His password has a length of 5 characters, and each character has an entropy of 3.0. This means that the new password of *Plato* has a total entropy, or complexity, of $5 * 3.0 = 15.0$. We know that the probability of *Plato* failing to memorize a password of seven digits is equal to his $forget$ value, or 0.3. We also know that a password of seven digits has an entropy equal to $7 * log_2 10$, or 23.2535. Knowing both the complexity of his password (15.0) and the probability that *Plato* fail to memorize a seven-digit password (30%), we can calculate how likely *Plato* is to fail to memorize his new password on the first attempt.
$P = \frac{(0.3)*(15.0)}{23.2535} = 0.193519$
Therefore, *Plato* has around a 19% chance of failing to memorize his new password. If *Plato* memorizes his password, his value for $User_{isWritten}$ becomes $false$; otherwise, his value for $User_{isWritten}$ becomes $true$.

5.2.3 Attacks

After checking to see if he needs a new password, *Plato* is subjected to a number of brute-force attacks. *Plato* is subjected to 10 brute-force attacks per day. Determining whether any of the 10 attacks succeeds is function of his password's total entropy, which is 15.0, the product of $passEntropy$ and $passLength$. This means that his password is uniquely determined by 15 bits. We assume that an attacker knows that 15 bits determine the password, but does not know which bits those are. We know that 15 bits have a total of 2^{15} possible different combinations; therefore each of the ten brute force attacks has a probability to succeed of $\frac{1}{2^{15}}$. If any of them do succeed, then *Plato* becomes compromised. *Plato* has a $compWritten$ value of 0.1. This means that each day *Plato* has not yet memorized his password, he is subjected to an additional 10% chance of becoming compromised.

5.2.4 Trying Again to Memorize a Password

We have shown above that *Plato* has an 81% chance to memorize his password on the first try, the day he receives it. If he does not memorize it the first day, then every day after that *Plato* gets a number of chances to memorize his password equal to the number of days for which he has already had the password. So, suppose that *Plato* gets a new password on Monday; he has an 81% chance to memorize the password that day. Tuesday, *Plato* has another 81% chance to memorize his password. Failing to do so, his password remains written down and on the following day, Wednesday, *Plato* has two chances to memorize his password, each with an 81% chance of success. If *Plato* memorizes his password in either of the attempts, $User_{isWritten}$ becomes $false$.

5.2.5 Ending the Day

In the case of *Plato* and his 99 fellow users of policy π_3, the value of $potHarm$ is equal to 1. This means that each day which ends with *Plato* being compromised increases the total system harm value Σ_{harm} by exactly 1. Σ_{harm} starts as 0 and never decreases. The final value of Σ_{harm} for π_3 is 1906. This means that there are 1,906 instances of users of policy π_3 ending the day compromised; on average, a user of policy π_3 ends 19 days of the simulated year compromised.

The last step in the simulated day is checking whether the simulation continues or not. In this simulation, Ω_{harm} is set to zero, so the simulation does not end based on the value of Σ_{harm}. However, the simulation is set to end after 365 days. If 365 days have not passed, then *Plato* begins his day again.

5.3 Password Entropy Experiment

5.3.1 Data and Results

This experiment involves ten different instances of a given password policy, $\pi_{1.5}$, π_2, $\pi_{2.5}$, π_3, $\pi_{3.5}$, π_4, $\pi_{4.5}$, π_5, $\pi_{5.5}$, and π_6. These ten instances of password policies are identical except for their different values for per-character password entropy $passEntropy$. Information which is the same for all π_i is found in Table 4 under "Password Entropy." The ten different password policies differ in their per-letter entropy $passEntropy$. Specifically, for each π_i, $passEntropy_i = i$.

For this experiment, the simulation is run ten times, once per policy. As described in the algorithm, the simulation is probabilistic; the same random number seed is used in each of the ten simulation runs. In each run, the simulation is set to terminate after 365 days have passed; that is, $\Omega_{days} = 365$.

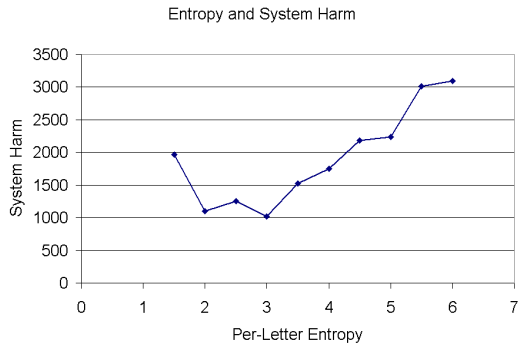

Figure 3: Comparison of Σ_{harm} versus $passEntropy$.

5.3.2 Discussion

The results of the experiment are reported in Figure 3. The $x - axis$ corresponds to the per-character entropy $passEntropy$ of the policies. Because the ten policies are otherwise identical, a policy with a higher value for $passEntropy$ requires a more complex password. The $y - axis$ shows the total system harm Σ_{harm} caused by the users of the different password policies. The graph illustrates a certain tension in password policy design as noted by [1, 2, 8, 12]. If password policy does not require a sufficiently complex password, users are more vulnerable to their passwords being cracked. However, if a password policy requires an overly complex password, users may have difficulty remembering their passwords consistently and may therefore write their passwords down. The lowest system harm in the results is when the required complexity is neither too little nor too great.

5.4 Password Change Frequency Experiment

5.4.1 Data and Results

This experiment involves examining nine different password policies. Each policy is the same except for the frequency with which its passwords expire. Let these different password policies be π_1 through π_9. Each password policy π_i has a value of $passExpire_i$ such that its users must change passwords i times in the course of a year. Thus π_1 has $passExpire_1$ equal to 365, π_2 has $passExpire_2$ equal to 182, and π_9 has $passExpire_9$ equal to 40.

Other than $passExpire$, all of the password policies share the same values, as show in Table 4 under under "Password Frequency." The simulations are set to end after 365 days pass. The same random number seed is used in each of the nine simulations.

Note that each user is subjected to only a 1% chance per day of becoming compromised because of his or her password being written down. Whereas, each day a user is subjected to ten attacks to guess his or her password, each of which have a probability to succeed of $\frac{1}{2^5}$, or around 3%. Thus, in this environment, brute-force attacks are more concerning than the threat of being compromised by writing down one's password.

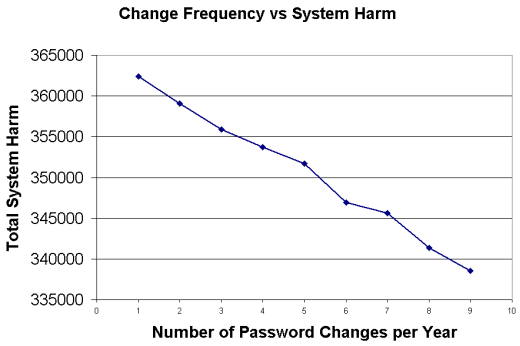

Figure 4: Comparison of Σ_{harm} versus $passExpire$.

5.4.2 Discussion

The results of the experiment are reported in Figure 4. As with the above experiment, the $y - axis$ shows the total system harm Σ_{harm} caused by the users of the different password policies. In this experiment, however, the $x - axis$ shows how many times each year the password is changed. One can observe that in the environment described by the variables, more frequent password changes lead to a more secure computer system.

This experiment illustrates how the password policy simulation model may be used to describe a specific environment, and is therefore adaptable to many different situations and organizations. The environment presented here is one in which the primary threat to a user being compromised arises from brute-force attacks on the user, and not from the user writing down his or her password. In other environments, the results may be different.

6. MODEL LIMITATIONS AND FUTURE WORK

Any simulation model entails some simplifications and assumptions, and the model presented in this paper is no exception. Despite being a fairly flexible model, there are some password policies which may not be easily represented directly in the model. For example, the model assumes that all users have a single password and a single account. The model may be extended in the future to include a single user with multiple accounts but one password for all of them, as is found in a system with single sign-on capability. It could also include a single user with multiple passwords.

The model presented here also makes assumptions about user behavior. It assumes that users always use the least complex password allowed by the policy; this assumption is supported by the

findings of [3] and [1]. It assumes that no two users share their passwords with one another intentionally, and that non-malicious users never divulge their password on purpose. This model assumes that compromised, non-malicious users never discover when they are compromised. In practice, many password policies such as that of Brown University require that users who believe themselves to be compromised contact an administrator [10]. In the model, a password is only changed when it expires.

The memory of a user is represented by a single probability – that of failing to memorize a seven digit number. The difficulty of memorizing a password increases directly with the complexity of that password. The model also assumes that a user becomes increasingly capable of memorizing a given password over time. Further, in our model, a user never contacts an administrator to ask what his or her password is, because that user is assumed never to forget a password once memorized. In many cases, there are tools to automatically provide users with the password again, so the administration costs will be very low. If however there are no such tools or the users has to go in person - as in the case of very critical accounts, the administrative costs may be very high. Therefore in future versions of this model we will incorporate the administrative costs of a user asking the administrator for a lost password.

Our model does not take into account administrative cost or how burdensome a password policy may be for its users. A policy which requires more frequent password changes can have a beneficial effect on system security, as shown in Figure 4. However, such frequently issued passwords may be costly to the system administrators. Moreover, even if users are able to memorize the new passwords, those users may still find the more frequently assigned passwords onerous and have more trouble memorizing them. Future work can extend the model to consider these factors.

This model simplifies all requirements a password policy may make on the composition of a password into two variables, the minimum per-character entropy of the password and its length. The length requirement is found directly in many actual policies, such as those cited above. However, actual password policies do not mandate a certain per-character complexity directly. Instead, they tend to accomplish this by requiring certain configurations of different character types, such as requiring that a password contain both an upper-case and lower-case letter. All of these sorts of requirements are cast into a single metric in this model, the per-character entropy $passEntropy$. This assumption allows us easily to compare the requirements of one password policy with another. Compared to an arbitrary list of text requirements, a decimal number is easier for a person using the simulation to enter into the computer, and easier for the model to utilize directly in its algorithm.

Finally, it would be insightful to measure how useful this model and simulation are in practice. Future work may include giving the simulation program to actual system administrators and seeing how helpful it is to them. Further future work may involve comparing the simulated results to those found in actual surveys among users.

7. RELATED WORK

Vu et al. conduct a study on various password policies [12]. They measure the ability of users to generate passwords under different policies and the effectiveness of these policies for creating more secure passwords. Password restrictions the authors recommend include a minimum length for the password, the use of special characters to increase complexity, and avoiding simple and predictable text patterns. The authors note that although a simple password is easy to guess, randomly generated character strings are difficult to remember and therefore more likely to be written down. While Vu et al. conduct a study on actual users of password policy, our paper presents a language and model through which password policies may be better understood.

Summers et al. present an analysis of passwords and password policies [8]. Passwords are frequently chosen poorly and password policies often permit this. A password without sufficient complexity is easy to crack, but a password which is too difficult to remember leads to a user writing it down. The authors recommend that passwords be changed regularly, but point out potential problems with a user being forced to change passwords too frequently. Similarly, our paper is concerned with the analysis of password policy. We incorporated the relation of the complexity of the password with the memorability of the users into the model to determine the optimal parameters in a password policy. Our experiments regarding the frequency change of passwords also favor frequent changes. However, as elaborated in Section 6, there exist additional concerns like administrative costs that may lead to other difficulties when passwords are changed too frequently.

Leyden conducts a survey on password use among office workers [3]. This survey shows that users are often not cautious with their password selection. 12% of the workers surveyed use the word "password" as their password; 16% use their own name; 11% use a sports team; and 8% use their birthday. Two-thirds of workers surveyed share their password with a co-worker. This indicates that users are unlikely to take the initiative to create secure passwords on their own. These findings justify our model assumption that users select the most simple passwords possible. However, our simulated users, unlike the actual users studied by Leyden, did not share passwords with one another. We may investigate how to incorporate the shared passwords in the simulation as a part of future work.

Adams and Sasse conduct a study on how users behave and what they believe regarding password policies [4]. Through interviews, they identify factors leading to more effective password usage. Many users have difficulty memorizing multiple passwords at once. Other users complain that requiring constant password changes leads to writing down passwords and using more simple passwords. These results suggest directions for future work in password policy modeling.

A survey conducted by SafeNet examines password utilization [1]. Half of the survey participants admit to writing down their passwords. The survey advises that three factors can improve the security of password policies: requiring longer passwords, increasing the required complexity of passwords, and requiring more frequent password changes. However, the survey also indicates that each of these steps may have the undesired consequence of making users more likely to write down a password. Surveys such as this gather information about the present state of password use, while our language and model provide a context in which policies may be explored. The SafeNet survey indicates that there are flaws in the present state of password use; where as our simulation model provides administrators with a tool to improve password policy.

Kuo et al. explore generating complex passwords which remain easy for the user to recall [2]. A good password, according to this paper, is both memorable and difficult to guess. More complex passwords can decrease the effectiveness of automatic password cracking schemes. Dictionary attacks can be thwarted by using a password which is not a commonly known word. Brute force password attacks rely on assumptions about which patterns of letters are found together. They therefore may be countered by deviating from the patterns the attacker expects. Our model notes the difficulty found in creating a password both easy to remember and difficult to crack. However, in our model, a brute force attack is a random guess which does not assume any patterns in the password.

8. CONCLUSION

The literature on password policy is in agreement on two statements: that a well-crafted password policy is vital to the security of an organization, and that an optimal policy is difficult to create. Recognizing that password policy creation is an important yet difficult problem, this paper offers three related contributions which will be useful in the creation and analysis of password policy. It presents an extensible language to describe comprehensive password policies. It then presents a simulation model, consisting of a set of variables and an algorithm, to simulate password policies being used on a computer system. Further, the paper presents an actual password policy simulation program providing insight on the effectiveness of such modeling.

The impact various factors have on password policy are studied through experimentation. It is shown that user memory impacts the security of computer systems. Moreover, it is shown that a good password policy must find a balance between an overly simple password and a password that is too complex. Above all, the open-ended password policy simulation model is shown to allow administrators to define many different kinds password policies and subsequently assess the effectiveness of those policies.

9. ACKNOWLEDGEMENT

The work reported in this paper has been partially supported by the United States Department of Education through the Graduate Assistance in Areas of National Need program and by the National Science Foundation under the ITR Grant No. 0428554 “The Design and Use of Digital Identities."

10. REFERENCES

[1] 2004 annual password survey results. *SafeNet*, 2005.

[2] Cynthia Kuo, Sasha Romanosky, and Lorrie Faith Cranor. Human selection of mnemonic phrase-based passwords. In *SOUPS '06: Proceedings of the second symposium on Usable privacy and security*, pages 67–78, New York, NY, USA, 2006. ACM Press.

[3] John Leyden. Office workers give away passwords for a cheap pen. *The Register*, 2003.

[4] Anne Adams and Martina Angela Sasse. Users are not the enemy. *Commun. ACM*, 42(12):40–46, 1999.

[5] III Robert M. Polstra. A case study on how to manage the theft of information. In *InfoSecCD '05: Proceedings of the 2nd annual conference on Information security curriculum development*, pages 135–138, New York, NY, USA, 2005. ACM Press.

[6] C. E. Shannon. Prediction and entropy of printed english. *Bell Systems Technical Journal*, 30:50–64, 1951.

[7] Anna Squicciarini, Abhilasha Bhargav-Spantzel, Alexei Czeskis, and Elisa Bertino. AuthSL: A System for the Specification and Enforcement of Quality based Authentication Policies. In *CERIAS Technical Report*.

[8] Wayne C. Summers and Edward Bosworth. Password policy: the good, the bad, and the ugly. In *WISICT '04: Proceedings of the winter international synposium on Information and communication technologies*, pages 1–6. Trinity College Dublin, 2004.

[9] Boston University. Information security management guidelines. http://www.bu.edu/computing/policies/infomanagement.html.

[10] Brown University. Brown university password policy. http://www.brown.edu/Facilities/CIS/policy/password.html.

[11] Amherst University of Massachusetts. Netid and passwords. http://www.oit.umass.edu/accounts/passwords.html.

[12] Kim-Phuong L. Vu, Robert W. Proctor, Abhilasha Bhargav-Spantzel, Bik-Lam (Belin) Tai, and Joshua Cook. Improving password security and memorability to protect personal and organizational information. *International Journal of Human-Computer Studies*, 2007.

[13] Jianxin Jeff Yan. A note on proactive password checking. In *NSPW '01: Proceedings of the 2001 workshop on New security paradigms*, pages 127–135, New York, NY, USA, 2001. ACM Press.

APPENDIX

A. SIMULATION INTERFACE

The graphical user interface simplifies running the simulation and guides the administrator through each step of the process. The model itself is described in Section 3 and Section 4.

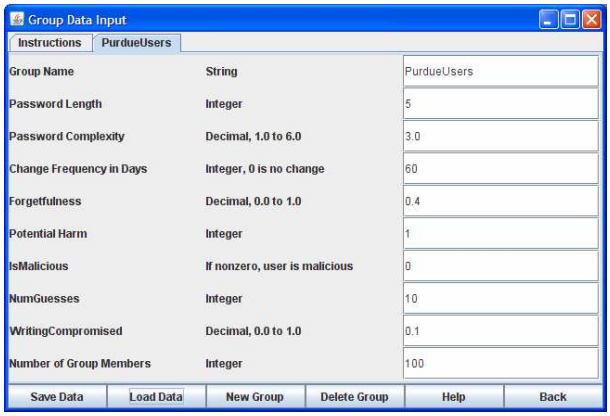

Figure 5: Policy specification interface

The administrator enters information about one or more password policies for the simulated computer system, as seen in Figure 5. After one or more password policies are entered, the administrator saves this data to disk for future use.

After the password policy data are entered, the simulation runs. Then, the administrator may use the graphical interface to explore the results. Section 4 describes a number of metrics which may be considered the output of the simulation. These are displayed for the administrator in the interface, as depicted in Figure 6. either the global results of the simulation or the data for policy The simulation results are saved to disk, and may be called up to be displayed by the interface any time in the future.

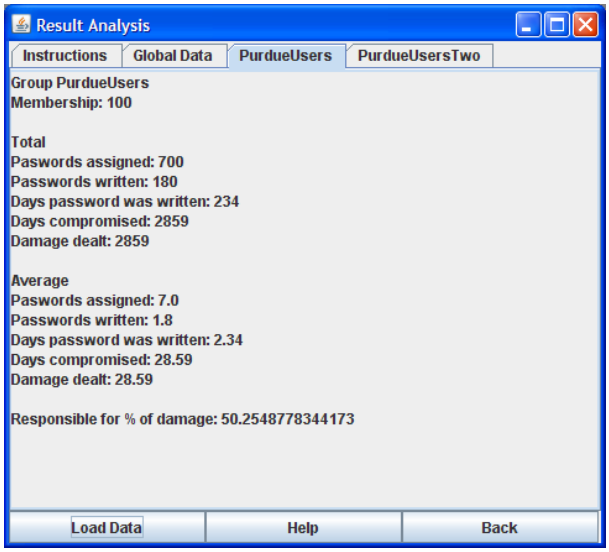

Figure 6: Simulation results interface

The graphical interface facilitates using the model. It includes a *help* tool which guides its user through each step, enabling the model to be a practical utility for administrators seeking to create or improve a password policy.