Automatically Designing Consistent Deception Plans in Defense of Computer Systems

Neil C. Rowe
Cebrowski Institute, U.S. Naval Postgraduate School
Code CS/Rp, 833 Dyer Road, NPGS, Monterey CA 93943
ncrowe@nps.edu

1. Introduction

Access controls including encryption are insufficient to protect computer systems from malicious attack today, as is witnessed by the many attacks that can circumvent them. It is important to examine secondary lines of defense once access controls have been breached as well as for insider attacks. One fruitful secondary line of defense is deliberate deception to fool attackers. Deception could fool attackers into going away because they think their attack cannot succeed, or into thinking they have succeeded in their goals when they have not. Deception is particularly useful for high-priority systems like those of the military and national infrastructure that must be kept running; then delaying a new kind of attack by deceptions may suffice to give time to figure it out and thwart it in real time. While there are ethical concerns about deception, most ethical theories permit deception to prevent a significantly greater harm, and destruction of the software of a computer system is a serious harm.

As an example, suppose a user logs in to a system, establishes an Internet connection, and transfers an executable file to their directory on the system. This is not suspicious. However, suppose they next try to transfer a big executable in which the signature of a Trojan horse is apparent. Rather than refuse this to them, which will just encourage them to find another route to get the file onto the system, we could deceive them. We could give a false excuse that the network is down, we could pretend to download the file but not actually do so, we could download it but insert random characters into it to make it useless, or we could download it and delete it later. (Rowe, 2004) developed a theory of "generic excuses" for not doing something or doing it incorrectly, most of which concerned resource availability like "the network is down". But the network-down deception will not be convincing if the user successfully downloaded a previous file. While the network could have stopped working between the two commands, it is unlikely and suspicious, and deceptions are ineffective against a suspicious user. On the other hand, we could give the false excuse that second file is too large to transfer, since that excuse would be logically consistent in an unchanging environment. We would like to find such logically consistent deceptions in a systematic way, so that we can choose a convincing deception when a user starts appearing malicious as per an intrusion-detection system (Proctor, 2001). Often more than one deception will be possible, so we also need a way to rank possible deceptions.

2. Previous work

Deception is a common social phenomenon (Ford, 1996). Many people assume that deception is bad and should never be done except under dire circumstances (Bok, 1978). However, in truth people use deception all the time for worthwhile ends (Scheibe, 1980; Nyberg, 1993), and it would be impossible for societies to function without it in many areas such as law, politics, business, entertainment, and psychology (Miller & Stiff, 1993). Attackers of computer systems use deception themselves in regard to their identities and software tools, so it would seem fair to use deception in defending our systems too. Deception has the advantage of being a generally surprising defense for computers, and one that can be difficult for attackers to perceive.

Deception has been used effectively for “honeypots”, computer systems intended solely for collection of data about attacks (The Honeynet Project, 2004). For defenders to collect useful data, the attacker must think that the system is a legitimate one. To this end, it helpful to create fake files and data on the honeypot to suggest that real users have been using it in normal ways, thereby “leading on” attackers (Cohen and Koike, 2003). Such fake data can include passwords to other machines, credit-card numbers, and other things which we can confirm are being used. Other work on computerized deception has addressed the creation of fake information for counterintelligence purposes, data that spies are encouraged to use so we can track them more easily (Gerwehr et al, 2000).

A variety of models of deception have been constructed, including those in philosophy (Chisholm & Feehan, 1977), psychology (Heuer, 1982), counterintelligence (Whaley, 1982), and stage magic (Nelms, 1969). Recent years have seen computational models including the descriptive models of (Cohen, 1999), the linguistic speech-act model of (DeRosis et al, 2003), agent-based deception-planning models (Christian & Young, 2004), reputation-systems assessment of deceptiveness (Barber & Kim, 2001; Yu & Singh, 2003), and a simple probabilistic model of (Park & Levine, 2001). Our previous work built probabilistic models of belief in generic excuses for why the attacker could not achieve their goals, plus the belief of the attacker in the hypothesis "I am being deceived" which we desire to minimize (Rowe, 2004); excuses are a versatile strategy for refusing to do things (Snyder, Higgins, & Stucky, 1980). A problem with tracking the degree of belief, however, is that it is very numeric. It is true that suspicion can be quantitative, in that people can easily assess degrees of suspiciousness in activities and people. But people also seem to reason logically (yes/no) as to whether they are being deceived. Thus it would seem useful to develop a theory of deception with a major component of logical reasoning to check logical inconsistencies in behavior.

3. Logical modeling of the attacker

3.1 Resources, facets, and parameters

In what follows we assume a computer system is under attack by a reasonably alert attacker who knows what they are doing and is trying to achieve particular goals. These are reasonable assumptions for insider attacks as well as state-sponsored or organization-sponsored outsider attacks, and even for some sophisticated hackers. The assumptions do not apply to script-based attacks where deceptions would be wasted on an attacker who does not notice or understand them. However, many good deception techniques we recommend, like false error messages about unavailability of resources, will deny access in suspicious circumstances and would easily stop scripted attacks.

We thus postulate an attacker of a computer system remembers the resources they used, directly or indirectly, in their interaction with it. For instance, when they successfully download a file from the network, this logically implies that the network, the local file system, and the remote file systems are working, at least for that particular kind of file. If later they try to download a similar file and are told the network is down, this is a logical inconsistency if the state of the system has not changed, and would make them likely suspicious of the normality of our computer system. This could lead to behavior (including retaliation) that is difficult to predict, hard for us to handle, and thus undesirable. Of course, the network could have gone down between the two actions, but that is unlikely in such a short time interval. So our first approach to consistent deception is to make:

The Statelessness Assumption: The resources of a computer system remain constant in their availability status through a session of a user.

We need to enumerate the resources associated with each possible command issued to an operating system, as these are the "material" used in attacks (Templeton & Levitt, 2000). The main categories are:

- The directories and files of the computer;
- Peripheral devices to which the computer is attached;
- Networks to which the computer is attached;
- Other sites accessible by the networks; and
- The executables for the commands run by an operating system.

For each of these resources, we have at least five facets of availability:
- Existence: Whether the resource exists (predicate: exists(X)).
- Status: Whether the resource is currently working (predicate: working(X)).
- Authorization: Whether the user is authorized to use the resource (as by passwords and access control) (predicate: authorized(X)).
- Initialization: Whether the preparations and initialization are sufficient for the associated action to be done (predicate: initialized(X,A)).
- Compatibility: Whether the associated action is compatible with the resource (as a text editor trying to open an image) (predicate: compatible(X,A)).

Resources also have associated ranges of suitable parameters and these ranges can be inferred. For instance, a successful download of a file of a size of one megabyte from site "foobar" implies that any file less than or equal to one megabyte could be transferred from foobar. But it does not necessarily imply that a ten-megabyte file could be transferred because bandwidth or buffer-size limitations could be exceeded. In general, each resource has associated parameter maxima:
- Files of the computer: Size in bytes, authorization level
- Peripheral devices to which the computer is attached: Bandwidth in bits per second, size of data in bytes
- The networks the computer is attached to: Size of data in bytes

As an example consider the successful action by user Bob of downloading a file "foobar.doc" of size 50,000 bytes from "site1" to "site2" via the FTP file-transfer utility at his local network "localnet". The resources necessary to accomplish this are file systems on "site1" and "site2", the network that connects them, and the executable for "ftp". So successful completion of the actions says that:
- File systems on site1 and site2 exist.
- File systems on site1 and site2 are working.
- File systems on site1 and site2 are authorized access by Bob.
- The network localnet is working.
- The network localnet is authorized use by Bob.
- The executable ftp on site1 is working.
- Bob is authorized to use the executable ftp on site1.
- The executable ftp on site1 is initialized
- The executable ftp on site1 is compatible with retrieving files from site2.
- Localnet can transfer files of 50,000 bytes or less from site2 to site1.

3.2 Using inference rules

Rules can be used to infer the status of a resource based on the status of another resource. This reduces the number of separate facts that need be specified for each action. Our approach is analogous to that in automated debugging of software by finding causal relationships (Zeller, 2002), but we assume an attacker will not want to systematically vary parameters to track down precise reasons for failures since they have more important things to do. It has more in common with debugging by looking for

inconsistencies in source code (Engler et al, 2001) as inferred from the actions attempted, as for instance when a program attempts to dereference a variable that has never been initialized.

When a user issues a command and it is successfully completed, that means that each input resource and all its facets are working. It also implies some things about the output resources. We can formalize this as (where "done" means the action was completed successfully without errors):

$$\forall X \forall A[(exists(X) \wedge working(X) \wedge authorized(X) \wedge$$
$$initialized(X,A) \wedge compatible(X,A)) \leftarrow (done(A) \wedge input(A,X))]$$
$$\forall X \forall A[(exists(X) \wedge authorized(X)) \leftarrow (done(A) \wedge output(A,X))]$$

Note that outputs of a successful command are not necessarily working, initialized, or compatible since other actions may need to be done first to use them, e.g. a downloaded compressed file.

If the command fails, usually a particular facet is indicated for failure by the message from the operating system. If more than one facet is negated by the failure, usually only one the highest-priority facet is indicated. For instance, Unix will first tell you if a resource does not exist, and then if it does, it will tell you if it is working, and then whether you are authorized to use that resource. So under Unix, if your attempt to download fails with the message “Site not available”, we can infer that the site exists and that it is not working, but we cannot infer whether the user is authorized to access it. In general, operating systems implement the rules:

$$\forall X[exists(X) \leftarrow working(X)]$$
$$\forall X[working(X) \leftarrow authorized(X)]$$
$$\forall X \forall A[authorized(X) \leftarrow initialized(X,A)]$$
$$\forall X \forall A[initialized(X,A) \leftarrow compatible(X,A)]$$

Note the contrapositive of these says that if X does not exist, none of the other facets are true for it.

We also need rules for accessing indirect resources like remote network sites. Such sites require additional resources like a network connection to access. In general:

$$\forall A \forall X \exists Y[input(A,X) \leftarrow (input(A,Y) \wedge needs(Y,X))]$$

Inheritance rules also apply to resource facets and the part-whole relationship. An example is inheritance of "working": If we know that the file-transfer buffer is not working, and it is necessary for file transfers, then file transfers are not working by upward inheritance. Analogously, if we know that file transfers are working, then the file-transfer buffer must be working by downward inheritance of the opposite property. But both these inferences require a strict interpretation of the part-whole relationship to mean a *necessary* part of the whole, as contrasted with an optional part of a whole like debugging features in a file-transfer process. The other facets of resources also lend themselves to similar inference rules, with the exception of "existence" since failure of a software object to exist in the usability sense doesn't mean its components don't exist. Unfortunately, the type-subtype relationship, which is usually a fruitful source of inheritance rules, is not helpful here because all objects observed by a user are ground instances at the same level of abstraction. Formally, we can write:

$$\forall X \forall Y[working(X) \leftarrow (part_of(X,Y) \wedge working(Y))]$$
$$\forall X \exists Y[\neg working(X) \leftarrow (part_of(Y,X) \wedge \neg working(Y))]$$
$$\forall X \forall Y[authorized(X) \leftarrow (part_of(X,Y) \wedge authorized(Y))]$$
$$\forall X \exists Y[\neg authorized(X) \leftarrow (part_of(Y,X) \wedge \neg authorized(Y))$$
$$\forall X \forall A \forall Y[initalized(X,A) \leftarrow (part_of(X,Y) \wedge initialized(Y,A))]$$
$$\forall X \forall A \exists Y[\neg initialized(X,A) \leftarrow (part_of(Y,X) \wedge \neg initialized(Y,A))]$$
$$\forall X \forall A \forall Y[compatible(X,A) \leftarrow (part_of(X,Y) \wedge compatible(Y,A))]$$
$$\forall X \forall A \exists Y[\neg compatible(X,A) \leftarrow (part_of(Y,X) \wedge \neg compatible(Y,A))]$$

Unfortunately the negative-conclusion rules are not especially helpful in the direction they specify since usually we know about the failure of a whole and want to infer the failure of a part – this is what troubleshooting is about. However, a common additional assumption is that a single "atomic" cause is responsible for all observed manifestations since coincidental failures of two or more objects are rare. We can express this by:

$$\forall P \forall Q \exists X \exists Y[P = Q \leftarrow (\neg working(P) \wedge \neg working(Q) \wedge atomic(P) \wedge atomic(Y)$$
$$\wedge part_of(P,X) \wedge part_of(P,Y) \wedge part_of(Q,X) \wedge part_of(Q,Y)$$
$$\wedge \neg working(X) \wedge \neg working(Y))]$$

Attacker knowledge can also be expressed as rules. For instance, if the file-transfer utility always seems to "break" when the attacker tries to transfer a file of more than one million bytes, the attacker would learn a rule:

$$\forall X \forall A[\neg exists(X) \leftarrow$$
$$(output(X,A) \wedge a_kind_of(A, file_transfer) \wedge size(A,S) \wedge (S > 1000000))]$$

This is a classic instance of concept learning where the attacker tries to generalize from a finite set of examples to rules that can handle a infinity of cases. Concept learning, like all learning, cannot be guaranteed to be correct: One must take chances in guessing what the proper generalization of the data must be. Nonetheless, we can be quite aggressive in postulating inference rules here because users will tend to seek logical explanations of unusual events as they expect computer systems to be consistent.

3.3 Proposing deceptions

Our strategy in defending a computer system is to deceive when the probability of user maliciousness exceeds a threshold, and to consistently maintain that deception through the deception. For maximum effectiveness, the deception should also be backwards consistent (consistent with the resource facets reported so far). It should also be chosen so as to appear to prevent completion of the user's inferred attack plan. For instance, if we wish to prevent a rootkit download, we pretend the network is down; if we wish to prevent installation of a Trojaned executable, we pretend it is too large to install. This requires some reasoning about attack plans, but ultimate attacker goals are not highly varied (install a rootkit, make a site unusable, steal information, etc.) and many attacks are well documented (Chirillo, 2002).

For each resource and facet, we have several tactics as to how to deceive, building on the general principles discussed in (Rowe & Rothstein, 2004):

- Tell the user the deception and do not do the requested action, as with an error message. For instance, say "network down" on an attempt at a file transfer from another computer. (Applicable to existence, working, authorization, and compatibility facets.)

- Cause abnormal symptoms to occur and do not do the request action. For instance, on an attempt at a file transfer from another computer, generate strange characters on the screen and terminate early without the transfer being made. (Applicable to working, initialization, and compatibility facets.)
- Don't do the action, but don't tell the user. For instance, if the resource is "network1" and the facet is "working", then do not transfer any files when asked to download them. (Applicable to the working facet.)
- Do the action, but do it in a way that makes it worthless. For instance, make random changes to files when transferring them so the user cannot use them. (Applicable to working, initialization, and compatibility facets).
- Obey the user's orders, but undo the action later. For instance, transfer files when asked but delete them later when user will not notice it as easily. (Applicable to the authorization facet.)
- Do the action, but refuse further similar actions by alleging a side effect of the first action. For instance, allow a download of one file, but refuse further downloads with an error message that suggests the user "broke" the file-transfer utility. (Applicable to the compatibility facet.)

The first method involves a phenomenon (error messages) that is more commonly observed on computer systems than the others. Thus we will assume it in the rest of this paper. However, it may be more convincing in the long run to use some of the other methods with small probabilities to better model the usual distribution of bugs of a computer system.

3.4 Implementing the finding of consistent deceptions

We have written a program that computes such resource implications for sequences of commands issued to an operating system. It returns a list of the resources they imply, taking into account the range restrictions. For a demonstration, commands were drawn from the 110 in the rootkit-installation model of (Rowe, 2004). Default resources for every command are the agent issuing the command and the computer system from which the command is issued; beyond this, resources usually appear as arguments to commands. This program can be used to plan deceptions; those resources and facets not identified in the analysis output for a sequence of commands are suitable objects of deception.

For example, consider the sequence of actions by a user:

1) Login in to system "patsy".
2) Make an FTP connection to site "hackerhome" using local-area network "homelan".
3) Download file "secureport.exe" from "hackerhome" across "homelan".
4) Download file "rootkit.exe" from "hackerhome" across "homelan".
5) Close the FTP connection.

Suppose the following a priori probabilities:

- 0.01: A site is not up.
- 0.02: A network is not working.
- 0.005: An executable is not working.
- 0.2: A user is not authorized to use a site.
- 0.1: A user is not authorized to use a network.
- 0.02: A user is not authorized to use the executable.
- 0.05: A user is exceeding magnitude bounds with a command
- 0.01: All other failure situations.

We further assume that the value of a deception is a monotonically increasing product of the above number and $1+\ln(K)$ where K is the number of occasions on which the deception can be used, to reward deceptions that are repeated as being more obvious to the user (human perception of quantity is logarithmic). Then the best logically-consistent deceptions assuming statelessness as computed by our program are:

- 0.52: The user is not authorized to use the site "patsy", in response to step 1.
- 0.48: The user is not authorized to use the site "hackerhome", in response to step 2.
- 0.26: Site "patsy" is not working, in response to step 1.
- 0.24: The user is not authorized to use the network "homelan", in response to step 2.
- 0.24: Site "hackerhome" is not working, in response to step 2.
- 0.17: The user is not authorized to log in on site "hackerhome", in response to step 2.
- 0.12: The file "secureport" is too large to transfer, in response to step 3.

4. Choosing if and when to deceive

4.1 Incorporating probabilities and costs

We can initiate deceptions that are consistent with resource inferences at many times in a sequence of commands. However, some opportunities are better than others because (a) some deceptions are more convincing, and (b) some deceptions cause more potential harm to legitimate users who just happen to be doing something accidentally suspicious. We will use a decision-theory model since this has been effective in understanding deception in sociobiology (Lachmann & Bergstrom, 2003).

The simplest possible model is shown in Figure 1. We assume that the user is malicious with probability p_m, a malicious user will continue working after the deception with probability p_{mr}, and a legitimate user will continue working after the deception with probability p_{lr}. The first will be obtained from the data provided by an intrusion-detection system (Proctor, 2001; Monteiro, 2003). The latter will be personality-dependent and task-dependent, since some users are more determined than others for reasons that are not immediately obvious (Lydon & Zanna, 1990). Assume also the cost of allowing an attack to succeed is c_m, and the cost of the deception in terms of preventing a legitimate user from achieving their goals is c_l.

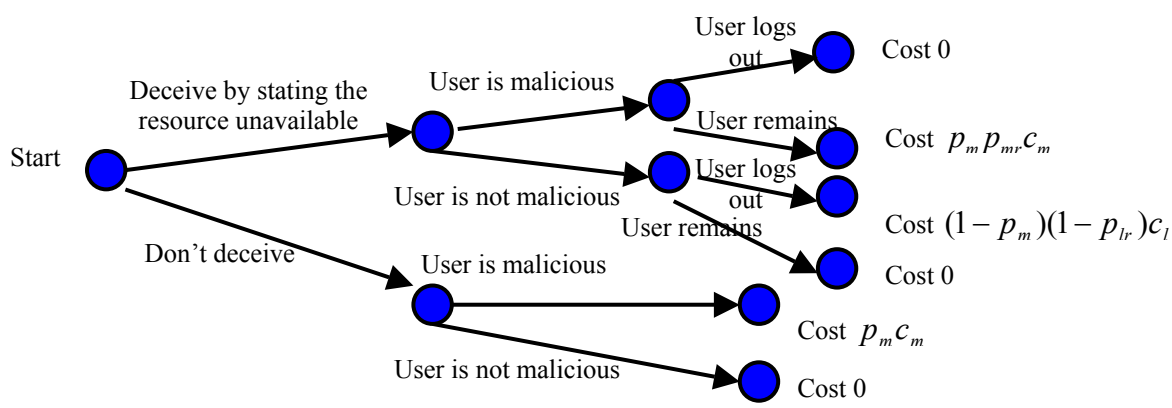

Figure 1: The simplest model of the decision to deceive at some point in a command sequence.

Then, following Figure 1, the expected cost of deceiving will be less than the cost of not deceiving if $p_m p_{mr} c_m + (1-p_m)(1-p_{lr})c_l < p_m c_m$, or when $p_m > c_l(1-p_{lr})/[c_l(1-p_{lr}) + c_m(1-p_{mr})]$. This can be written as $p_m > g[c_l(1-p_{lr})/c_m(1-p_{mr})]$ where $g(x) = x/(1+x)$. This g is the same function used in converting odds to probabilities, so the expression is brackets is analogous to an odds measure.

Note this inequality implies that when the deception is completely ineffective in encouraging the malicious user to go away, it is never desirable; if it completely ineffective in encouraging the legitimate user to go away, it is always desirable; and if the deception no more affects the attacker's goals than the legitimate user's goals, the inequality reduces to $p_m > c_l/(c_l + c_m)$ so even then the deception may be desirable.

For example, suppose our deception is to pretend that the network is down. This will prevent most rootkit installations, and the cost of one of these is maybe 2 hours of system-administrator work to reinstall the operating system and close backdoors. The cost to a legitimate user of the network being down is probably 5 minutes of work since in most cases they can copy a portable storage medium to do file transfers or they can find another machine to use the Web. Since a network transfer is essential to most rootkit installations, a malicious user will have a low probability of continuing after this deception, say 0.1. On the other hand, legitimate users probably have other tasks to accomplish on the system that do not involve the network, such as editing files, so their probability of continuing is something like 0.5. Hence our criterion formula becomes $p_m > g[(5*0.5)/(120*0.9)]$ or $p_m > 0.023$.

Appendix A shows the results of a survey to estimate some probabilities about an operating system and potential deceptions on it. The survey was administered to seven students and five faculty members at our school. Geometric means (the antilog of the mean of the logarithms) are displayed since the quantities reported varied widely in magnitude but could not be 0. Geometric means of the response values for each subject ranged from 0.187 to 0.000002, but *ratios* of responses between questions were much less variable, indicating that conditional probabilities were more consistent across subjects than a priori probabilities (that may reflect personality). It can be seen (questions 2a-2f) that deception is considered a reasonably possible explanation for events that are rare on a computer system, with conditional probabilities of 0.32, 0.54, and 0.09 for deception with the reported conditions "network down", "local network down", and "network messing up files". So subjects understand that just because deception is rare a priori does not mean it is conditionally rare. It can also be seen that our subjects were persistent in trying the same thing again even when it failed three times, so p_{mr} and p_{lr} should be close to 1; thus for these subjects, we can use the approximation $p_m > c_l/(c_l + c_m)$.

4.2 Refining the deception model

The Statelessness Assumption becomes less tenable the longer the period of time over which it applies. One problem with the simple model above is that it implies we should wait as long as possible to deceive an attacker. That is because the longer we wait, the more clear it should be that they are malicious if they are; the larger this p_m is, the more likely the inequality holds, and the larger the expected cost benefit to us. However, it may be dangerous to wait too long to deceive. (Josang, 2001) suggests that trust and distrust are fundamentally different, and distrust grows much more easily than trust; waiting gives the attacker time to become distrustful. (Rowe, 2004) proposes that the believability of a deception in response to attacker commands is a product of three factors: the likelihood of the deception event, the likelihood that the system engages in deception, and the intrinsic suspiciousness of what the attacker has done previously. So attackers who know they are being suspicious are more inclined to be suspicious of events that thwart them, and this increases as a session progresses. We will postulate that attackers that think they are being deceived will not log out because they figure they can find a way around the deception, whereas unsuspicious attackers will log out because a deception should be chosen to prevent accomplishment of most attacker plans.

This gives us enough constraints to create a Bayesian model (Korb & Nicholson, 2004) of p_{mr}, the probability the malicious user remains logged in after a deception that would imply that they cannot achieve their goals. Using Bayes' rule where d is the occurrence of a deception action and c is the condition that the attacker believes they are being deceived, $p(c \mid d) = p(d \mid c)p(c) / p_d$ and $p(\neg c \mid d) = p(d \mid \neg c)(1 - p(c)) / p_d$, so after dividing the equations and solving for $p(c \mid d)$ which is equal by our assumptions to p_{mr}, we get $p_{mr} = p(c \mid d) = g[p(c)p(d \mid c)/(1 - p(c))p(d \mid \neg c)]$. So for the deception "network down" and the probabilities $p(c) = 0.01$, $p(d \mid c) = 0.1$, and $p(d \mid \neg c) = 0.01$, we estimate $p_{mr} = 0.9$. On the other hand, we will assume p_{lr} is a fixed constant independent of circumstances, because deceptions will usually be in regard to features that are not essential to a legitimate user.

The estimate of p_{mr} can also depend on previous events. For example, consider the condition "network down", which on the computer systems the author uses is approximately 0.02. But if the network was down one hour ago, that increases the probability it is down now. A down period for the author's network has an expected duration of one day. In general, we can use a Poisson model much like that for M/M/1 queuing models, where λ_D is the number of times that the deceptive condition D would occur sometime during a day (like an arrival rate for a queue), and λ_U is the number of times that D would be remedied during a day (like the service rate for a queue). Then if we reported to the user that D was true at some time, the probability that D is still true at a time t units later is $p_U = e^{-\lambda_U t}$; if we reported to the user that D was false at some time, the probability that D is still false at a time t units later is $p_D = e^{-\lambda_D t}$. This permits us to assign probabilities to inconsistent reports at different times.

Another issue to consider is that as the number of possible deceptions decreases during a session as resource availabilities are confirmed, the likelihood of being able to choose a convincing deception decreases. For instance, problems with accessing a Web site are less convincing excuses when the user has already had trouble with the network and file systems on several unrelated machines. This is similar to factor of "mobility" in the game of chess, where of two positions with equal piece count and danger, the one with more possible moves is preferred because it is less likely to lead to a forced bad move.

4.3 Anticipatory deception

We can use this approach to analyze the idea of anticipatory deception, where we deceive when we are only mildly suspicious to enable a more consistent and convincing deception later if we need to. For instance, we can deceive a user by saying the network is down in response to an attempt to transfer an innocuous file, in anticipation of a second deception critical to preventing an attack, as when a user attempts to transfer a rootkit. Then the first deception provides consistency of the "network down" hypothesis and reduces attacker suspicion. We assume, if we deceive more than once, that we use the same deception both times since deceptions will be rare events and two unrelated deceptions will be even rarer and thus particularly suspicious. Thus it will be important to use "generic excuses" like "network down" in deceptions because a broad such excuse can explain failure of many separate things. (Cohen and Koike, 2003) and (Rowe, 2003) proposed detailed decision graphs for particular known attack plans, but we would like here to provide methods more independent of the kind of attack.

To analyze this problem we refine our decision model of Figure 1. Figure 2 shows a situation where a user can be deceived twice. Here we distinguish p_{m1}, the probability the user is malicious at the first

deception, from p_{m2}, the probability at the second deception. We similarly distinguish p_{mr1}, the probability a malicious user will remain after the first deception, from p_{mr2}, the probability they will remain after the second deception. We also distinguish p_{lr1} from p_{lr2}, and both from p_{mr2d} and p_{lr2d}, the probabilities of the malicious or legitimate user remaining when the first deception is not done but the second deception is done. As before, the cost of allowing a malicious attack to succeed is c_m, and the cost of preventing a legitimate user from achieving their goals is c_l. We can generally make the simplifying assumption that $p_{lr1} = p_{lr2} = p_{lr2d} = p_{lr}$ since deceptions are designed with attackers in mind, not legitimate users, so each occurrence of the same deception is equally likely to make a legitimate user give up and go away. We also can generally assume that $p_{mr2d} > p_{mr2}$ and $p_{mr2d} > p_{mr1}$, since the "mr2d" case most obviously suggests deception, and apparent deception should make a malicious user want to remain logged in.

We can then calculate expected costs for three strategies for a particular deception D in the course of an assumed short period of time shown in Figure 2 (over a longer period of time, use the exponential decay formulas of the last section):

- S1: No deception: $p_{m2}c_m$
- S2: Deceive only at the second opportunity: $p_{m2}p_{mr2d}c_m + (1-p_{m2})(1-p_{lr})c_l$
- S3: Deceive at both opportunities:
  $p_{m2}p_{mr1}p_{mr2}c_m + (1-p_{m1})(1-p_{lr})c_l + (1-p_{m2})(1-p_{lr})p_{lr}c_l$

Note that it is a poor strategy to deceive only at the first opportunity and not at the second, as this does not foil an action necessary to the attack if it is anticipatory, creates a suspicious inconsistency, and risks antagonizing a legitimate user, so we ignore this strategy.

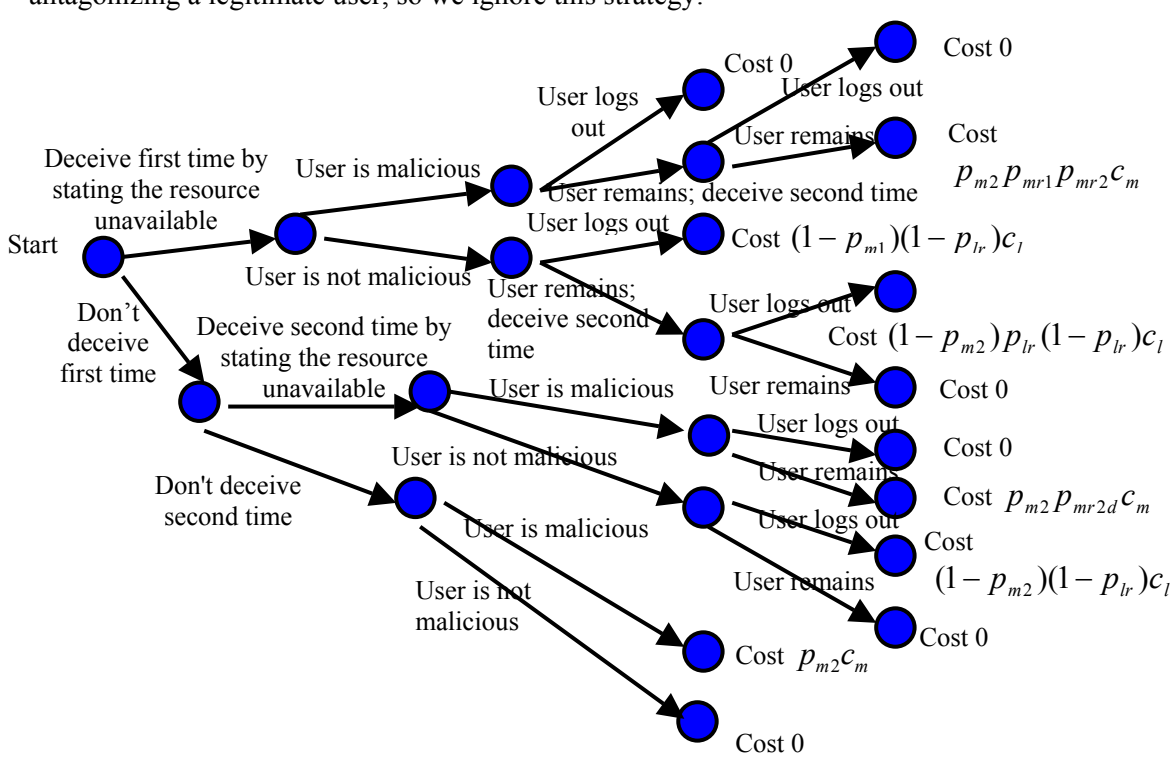

Figure 2: Analysis of the problem of anticipatory deception.

Unfortunately there is no general way to simplify the mathematics for comparing the strategies, so one must just calculate the formula for each possible deceptive situation. But we can address some special cases. As before, S2 and S3 will be suboptimal if the malicious user is never fooled, since then $p_{mr2d} = 1$, $p_{mr1} = 1$, and $p_{mr2} = 1$, hence the first terms of the expressions for S2 and S3 are the entire expression for S1. Usually, however, p_{mr2d} will be close to 1 because an attacker will find it hard to believe if they are paying attention, and will be equal to 1 if the attacker is not paying attention, so S2 will be usually undesirable compared S1.

4.4 Multi-session reasoning

So far we have assumed that user analysis is done within a session. However, an attacker could log in multiple times to accomplish their attack, as in "low and slow" attacks. We can make a small change to the above model to handle this. Each pair of sessions can have an associated probability that they are due to the same user. If we are trusting of our identity management, we could take this as 1 when a user logs in under the same name and 0 otherwise. However, attackers may very well masquerade as legitimate users, and we need to allow for intermediate probabilities of identity when a user starts doing something anomalous; atypicality can be measured by anomaly-focused intrusion-detection systems.

Again we can use the exponential model. A user logging in again after time t after logging out under a different name is likely to be the same user with probability $p_S = p_S e^{-\lambda_{Sm} t}$ if they are malicious and zero if they are nonmalicious, where λ_{Sm} is the malicious-user "decay" rate and p_S is the probability that a malicious user logging out will immediately log in again. If R users logged out recently, each with probability of maliciousness p_{mi} and at time t_i before the present, the probability a new user is now malicious is $p_{m0} + \sum_{i=1}^{R} p_S p_{mi} e^{-\lambda_{Sm} t_i}$ where p_{m0} is the a priori probability a new user would be malicious. But this is only an initial value; once the new session begins, this probability of being malicious will change with user actions as per the intrusion-detection system. Note we exclude users that are still logged in. Some of these could be conceivably opening separate login windows, but unless commands to the operating system are being generated by a software script, which can be detected by unusually fast commands, a user cannot do more than one thing at a time.

For instance, suppose user "tom" logged in at 10PM and used the file system and the network connection to site "foobar" and transferred what looks to be a rootkit, logged out at 10:30PM, and then another username logged in at 10:35PM with no one else on the system. Suppose $\lambda_{Sm} = 0.1$ for times in minutes, $p_S = 0.4$, $p_{m1} = 0.8$, and $p_{m0} = 0.001$. The probability the new user is malicious is thus $0.001 + 0.4 * 0.8 * e^{-0.5} = 0.195$. Suppose the cost of a successful malicious attack is 20, the cost of hurting legitimate user was 1, the probability of an attacker remaining after the particular deception "network not authorized" is 0.5, and the cost of a legitimate user remaining after this deception is 0.2. By our simple decision model, 0.195 > g((1*0.8)/(20*.5)) = g(0.08) = 0.08 so this deception is justified.

5. Conclusions

We have examined a new approach to defending computer systems, that of having them deliberately deceive attackers to waste their time or make them go away. Many opportunities for effective deception are available, so it is valuable to rank them. Counterintelligence training emphasizes the importance of finding discrepancies to consistency (Whaley & Busby, 2002); if we are deceive attackers of a computer system, we need to minimize their number. This requires keeping track of the logical inferences that

attackers can make to avoid obvious inconsistencies. But complete consistency is impossible, and we must introduce probabilities at some point and solve a decision-theoretic optimization problem.

Acknowledgements

This work was supported by the National Science Foundation under the Cyber Trust Program. The views expressed are those of the author and do not represent policy of the U.S. Government.

References

Barber, R. S., and Kim. J., Belief revision process based on trust: agents evaluating reputation of information sources. In Falcone, R., Singh, M., and Tan, Y.-H. (eds.), *Trust in Cyber-Societies*, LNAI 2246, Berlin: Springer-Verlag, 73-82, 2001.
Bok, S., *Lying: Moral Choice in Public and Private Life*. New York: Pantheon, 1978.
Chirillo, J., *Hack attacks revealed*. New York: Wiley, 2002.
Chisholm, R. M., & Feehan, T. D., The intent to deceive. *Journal of Philosophy, 73* (3), 143-159, March 1977.
Christian, D., and Young, R. C., Strategic deception in agents. Proc. 3rd Intl. Joint Conference on Autonomous Agents and Multiagent Systems, New York, NY, 218-226, 2004.
Cohen, F., Simulating cyber attacks, defenses, and consequences. http://all.net/journal/ntb/simulate/simulate.html, May 1999.
Cohen, F., and Koike, D., Leading attackers through attack graphs with deceptions. *Computers and Security*, Vol. 22, no. 5, pp. 402-411, 2003.
De Rosis, F., Castelfranchi, C., Carofiglio, V., & Grassano, R., Can computers deliberately deceive? A simulation tool and its application to Turing's imitation game. *Computational Intelligence*, Vol. 19, No. 3, 2003, 235-263.
Engler, D., Chen, D., Hallem, S., Chou, A., & Chelf, B., Bugs as deviant behavior: a general approach to inferring errors in systems code. Proc. 18th ACM Symposium on Operating System Principles, Banff, Alberta, Canada, 57-72, 2001.
Ford, C. V., *Lies! Lies!! Lies!!! The Psychology of Deceit*. Washington, DC: American Psychiatric Press, 1996.
Gerwehr, S., Weissler, R., Medby, J. J., Anderson, R. H., & Rothenberg, J., Employing deception in information systems to thwart adversary reconnaissance-phase activities. Project Memorandum, National Defense Research Institute, Rand Corp., PM-1124-NSA, November 2000.
Hardin, R., *Trust and trustworthiness*. New York: Russell Sage Foundation, 2002.
Heuer, R. J., Cognitive factors in deception and counterdeception. In *Strategic Military Deception*, ed. Daniel, D. C., & Herbig, K. L., New York: Pergamon, 1982, pp. 31-69.
The Honeynet Project, *Know Your Enemy, 2nd Edition*. Boston: Addison-Wesley, 2004.
Josang, A., A logic for uncertain probabilities. *Int. Journal of Uncertainty, Fuzziness, and Knowledge-Based Systems*, Vol. 9, No. 3 (June), 2001, pp. 279-311.
Korb, K., & Nicholson, A., *Bayesian artificial intelligence*. Boca Raton, FL: Chapman and Hall/CRC, 2004.
Lachmann, M., & Bergstrom, C., Deception and the value of information. University of Washington Working Paper, 2003.
Lydon, J., & Zanna, M., Commitment in the face of adversity: A value-affirmation approach, *Journal of Personality & Social Psychology*, Vol. 58, No. 6, June 1990, pp. 1040-1047.
Miller, G. R., & Stiff, J. B., *Deceptive Communications*, Newbury Park, UK: Sage Publications, 1993.
Monteiro, V., How intrusion detection can improve software decoy applications. M.S. thesis, U.S. Naval Postgraduate School, March 2003, www.cs.nps.navy.mil/people/faculty/rowe/oldstudents/monteiro_thesis.htm.
Nelms, H., *Magic and showmanship: A handbook for conjurers*. Mineola, NY: Dover, 1969.

Nyberg, D., *The varnished truth: truth telling and deceiving in ordinary life*. Chicago: University of Chicago Press, 1993.
Park, H. S., and Levine, T. R., A probability model of accuracy in deception detection experiments. *Communication Monographs*, Vol. 68, No. 2 (June), 2001, 201-210.
Proctor, P. E., *Practical intrusion detection handbook*. Upper Saddle River, NJ: Prentice-Hall PTR, 2001.
Rowe, N. C., Counterplanning deceptions to foil cyber-attack plans. Proc. 2003 IEEE-SMC Workshop on Information Assurance, West Point, NY, June 2003, 203-211.
Rowe, N., Designing good deceptions in defense of information systems. *Computer Security Applications Conference*, Tucson, AZ, 418-427, December 2004.
Rowe, N., & Rothstein, H., Two taxonomies of deception for attacks on information systems. *Journal of Information Warfare, 3* (2), 27-39, July 2004.
Scheibe, K., In defense of lying: on the moral neutrality of misrepresentation. *Berkshire Review, 15*, 15-24, 1980.
Snyder, C. R., Higgins, R. L., and Stucky, R. J., *Excuses: masquerades in search of grace*. New York: Wiley, 1983.
Templeton, S., & Levitt, K., A requires/provides model for computer attacks. Proc. of the New Security Paradigms Workshop, Cork, Ireland, September 2000.
Whaley, B., Towards a general theory of deception. *Journal of Strategic Studies*, Vol. 5, No. 1, pp. 179-193, March 1982.
Whaley, B., & Busby, J., Detecting deception: practice, practitioners, and theory. In Godson, R., & Wirtz, J. (Eds.), *Strategic Denial and Deception* (New Brunswick: Transaction Publishers, 2002), pp. 181-221.
Yu, B., & Singh, M., Detecting deception in reputation management. Proc. Conf. Multi-Agent Systems (AAMAS), Melbourne, AU, 73-80, July 2003.
Zeller, A., Isolating cause-effect chains from computer programs. Proc. 10th ACM SIGSOFT Symposium on Foundations of Software Engineering, Charleston, SC, 1-10, 2002.

Appendix: Questionnaire and geometric mean of the responses

Instructions: The following questions refer to using computer systems at NPS. Please estimate the probability for each question based on your experience and intuition, using a scale between 0 and 1. Because nothing can be absolutely certain, you shouldn't give an answer of 0 or 1 for any of these questions.

1a. What is the probability of finding a random computer's Internet connection down on a random day? .0416

1b. What is the probability of finding a random computer's local-area network down on a random day? .0422

1c. What is the probability that communications defaults are messed up so that binary file transfers across the Internet make the files unopenable? .0079

1d. What is the probability that a random computer will say that you are not an authorized user when you give it a correct password? .0057

1e. What is the probability that a random computer will deliberately deceive you? .0043

2a. What is the probability that a random system is wrong when it says the Internet is down? .0165

2b. What is the probability that a random system is wrong when it says its local-area network is down? .0086

2c. What is the probability that a random system will inadvertently mess up your files in binary transfers across the Internet? .0113

2d. What is the probability that a random system will lie to you when it says the Internet is down? .0072

2e. What is the probability that a random system will lie to you when it says its local-area network is down? .0032

2f. What is the probability that a random system will deliberately mess up your files in binary transfers across the Internet? .009

3a. Supposing you need to download a file as the major part of your job today, what is the probability you will log out immediately when the system says the Internet is down or it appears that the Internet is down? .0196

3b. Supposing you need to download a file as the major part of your job today, what is the probability you will log out immediately when you have tried three times to transfer files and the system says the Internet is down or it appears that the Internet is down each time? .1012

3c. Supposing you need to download a file as the major part of your job today, what is the probability you will log out immediately when the system says the local area network is down or it appears that the Internet is down? .0566

3d. Supposing you need to download a file as the major part of your job today, what is the probability you will log out immediately when you have tried three times to transfer files and the system says the local area network is down or it appears that the Internet is down each time? .0702

3e. Supposing you need to download a file as the major part of your job today, what is the probability you log out immediately when the files you transfer are messed up? .0052

3f. Supposing you need to download a file as the major part of your job today, what is the probability you log out immediately when you have tried three times to transfer files and the files you transfer are messed up each time? .0225