2003-06

Counterplanning Deceptions to Foil Cyber-Attack Plans

Rowe, Neil C.

Monterey, California. Naval Postgraduate School

Proceedings of the 2003 IEEE Workshop in Information Assurance, West Point, NY, June 2003

Counterplanning Deceptions to Foil Cyber-Attack Plans

Neil C. Rowe

Cebrowski Institute for Information Innovation and Information Superiority
and Computer Science Department
Code CS/Rp, 833 Dyer Road, Naval Postgraduate School, Monterey, CA 93943
ncrowe at nps.navy.mil

Abstract ? Tactics involving deception are important in military strategies. We have been exploring deliberate deception in defensive tactics by information systems under cyber-attack as during information warfare. We have developed a tool to systematically "counterplan" or find ways to foil a particular attack plan. Our approach is to first find all possible atomic "ploys" that can interfere with the plan. Ploys are simple deceits the operating system can do such as lying about the status of a file. We analyze ploys as to the degree of difficulty they cause to the plan wherever they can be applied. We then formulate a "counterplan" by selecting the most cost-effective set of ploys and assign appropriate presentation methods for them, taking into account the likelihood that, if we are not careful, the attacker will realize they are being deceived and will terminate our game with them. The counterplan can be effected by a modified operating system. We have implemented our counterplanner in a tool MECOUNTER that uses multi-agent planning coupled with some novel inference methods to efficiently find a best counterplan. We apply the tool to an example of a rootkit-installation plan and discuss the results. [1]

Index terms ? deception, information warfare, planning, counterplanning, rootkits, intelligent agents, stochastic models, decision theory.

I. INTRODUCTION

Military organizations today heavily depend on their information systems. Unfortunately, many vulnerabilities in software are known and new vulnerabilities are constantly being discovered [1]. This means it is unreasonable for a military organization -- and particularly the U.S. military, which uses much vulnerable commercial software -- to think their systems cannot be compromised by determined adversaries with serious motivations (as opposed to "hackers" and "script kiddies"). A second line of defense beyond access controls is critical for military information systems.

Deceptive tactics could provide a second line of defense, but they have not been much investigated. "Honeypots" and "honeynets" [2], computer systems designed to entrap attackers and collect information about them, are a simple "decoy" deception that is increasingly popular. But honeypots are a relatively passive deception that responds in substantially the normal way to an attack. Honeypots also violate a key principle of effective deception, that it should be integrated with operations [3]: They are easy to recognize via network-traffic "sniffers" and inspection of their file systems. A real computer system that serves real needs is more likely to fool an attacker. Such a system could be part of "active network defense", defense that impedes an attacker in more sophisticated ways.

Active network defense can exploit the classic deception methods of conventional warfare [4]: concealment, camouflage, ruses (using enemy equipment and procedures), demonstrations (of capabilities), feints, false and planted information (disinformation), lies, displays ("techniques to make the enemy see what isn't there"); and insight ("deceiving the opponent by outthinking him"). Of the nine, concealment and camouflage of operations (as opposed to data) are difficult in cyberspace since so much of it is black and white: Either a file exists or not. Then domain name servers are quite willing to tell adversaries about what resources exist. Ruses are not helpful in cyberwarfare because identity theft is easy and lacks surprise. Demonstrations are hard to make convincing and likely counterproductive since shows of strength encourage attacks which cannot be defended. Feints are not helpful because it is so difficult to localize an enemy for a counterattack in cyberspace. False and planted information such as posting fake attack methods on hacker bulletin boards is certainly possible, but it is generally easy to confirm most statements about cyberspace with sufficient time.

This leaves lies, displays, and insights as potential defensive tactics. And they are powerful tactics just beginning to be explored [5]. Outright lies by an information system can be effective because users are so accustomed to truth from their systems. The system could lie about completing a download of a suspicious file or give a false excuse as to why it cannot. Or lies could be integrated into an

overall "display" for the attacker. An example would be simulating an infection by a virus while actually destroying the infection. "Insights" would be the next step, where a set of lies and displays could be integrated into an overall defensive strategy designed to cause the attacker the maximum amount of trouble, using methods from artificial intelligence. The rest of this paper will present a mechanism to do this. Note in the limited space here we must ignore equally important issues we are currently exploring such as to how we recognize an attack and how to embed deception in software [5].

II. OBSTRUCTIVE COUNTERPLANNING

Obstructive (or adversarial) counterplanning is planning to interfere with or frustrate an existing plan [6], and it has important applications to military mission planning. It differs from game theory in focusing on complex strategies. Not much research relates to this topic. The work [6] proposes a top-down approach to counterplanning intended for understanding stories. While helpful as a general explanation for counterplanning tactics, its heuristics proposed are difficult to use to generate counterplans. One military project [7] provided a rather general approach to counterplanning using heuristic search. Some work in management has explored how organizations can work effectively, and this work can be inverted to provide suggestions for counterplanning against enemy organizations [8]. Some interesting work [9] used decision-tree attack models for information systems to study ways in which attacks could be detected. A cause-and-effect network can model attack processes in a similar way [10]. Decision trees and graphs, however, are equal in power to propositional calculus, and the more powerful resources of predicate calculus including variables are necessary to accurately model the ability of attackers to generalize and learn from experience. Also, [9] is preliminary work; [10] is a high-level model that, while quite comprehensive, is insufficiently detailed to generate attack plans. Furthermore, the approaches of [9] and [10] require extensive human expertise to implement since they are "expert systems" that do not reason from general principles.

We have built a general tool for doing obstructive counterplanning using predicate calculus, MECOUNTER, based on our earlier research on tools for planning [11, 12, 13]. A good such tool needs an intelligent planner to anticipate what an enemy would do, what we could do to obstruct it, and what the enemy could do to respond to our obstructions. Hierarchical planning is desirable with sufficient lead time to do a thorough job, and when we need to model goal priorities and complex kinds of human behavior; both conditions apply to system-attack modeling. Our planner handles abstraction with variables, multi-agent phenomena [14], and stochastic effects. Plans are decomposed into precondition and postcondition subtrees using formal definitions of actions and their priorities. Goals can be full predicate-calculus expressions with negations and disjunctions as well as conjunctions. Unexpected events can be handled by replanning, which is made efficient by caching plan starts for subproblems. Stochastic effects can be situationally dependent.

Each action has a priority list of agents qualified to accomplish it. Agents plan independently to achieve their goals, assuming cooperation as necessary from other agents. They can have skill levels, resource limitations, and can recognize and abort actions when other agents invalidate preconditions. Agent communication is modeled by an order-report paradigm: One agent (a "client") submits an order to another (a "proxy") to accomplish particular goals, the proxy constructs and executes a plan to achieve the goals, and reports to the client. For cyber-attacks, the attackers are the clients and the computer system is the proxy.

MECOUNTER makes it straightforward to define attack models with mostly-declarative definitions of actions through our tools. For instance, we defined the "decompress" action on a file for a demonstration as follows (as paraphrased from the Prolog program):

- *If the system does not want a file to be compressed, it should decompress it.*
- *Decompressing a file on a system requires that it is known to be there, it is compressed, it is known to be compressed, the actor is logged in to its system, and the actor is not currently running any other program there.*
- *Normally when a file is decompressed, this deletes the fact of its compression and adds no new facts.*
- *10% of the time decompression fails with the error message "Wrong format".*
- *Decompression requires a mean time of five seconds and has a Poisson distribution.*

There are also definitions of order and report actions. Their preconditions are especially interesting. The proxy (the computer system) cannot normally report something unless it is true and it has been ordered. The client cannot order the proxy to do something unless it is possible to do it immediately, since this kind of proxy does not make plans. Some orders need additional specialized preconditions; for instance, it does not make sense to order the computer to log you out until every goal there is reported to have been achieved.

For a significant test of the planner and subsequent counterplanner, we built a model for attacks on computer systems that use buffer overflows to install rootkits. This model is of a sufficiently high level to ignore details that change frequently between attacks such as

the particular buffer-overflow method, but detailed enough to capture most details of situation-independent behavior. It models 19 action types which can be instantiated to 93 distinct actions. The specifications cover a possible 115 distinct facts (and their 115 negations) in states, preconditions, and goals, and allow for 13 categories of random occurrences with actions. We defined for it 3072 test starting states for the goals of installing a rootkit on a system, installing gimmicked port-management software, and logging out. When no complicating random events occur, 48 steps are sufficient to achieve the goal from the most complex start state.

Figure 1 shows a simplified-example problem-decomposition tree created by our planner for a "root compromise" of a computer system. Inorder traversal of the tree (left branch, root, and then right branch) gives the order of actions. "FTP" is a file-transfer utility. Orders and reports have been omitted as well as several other details of our full model.

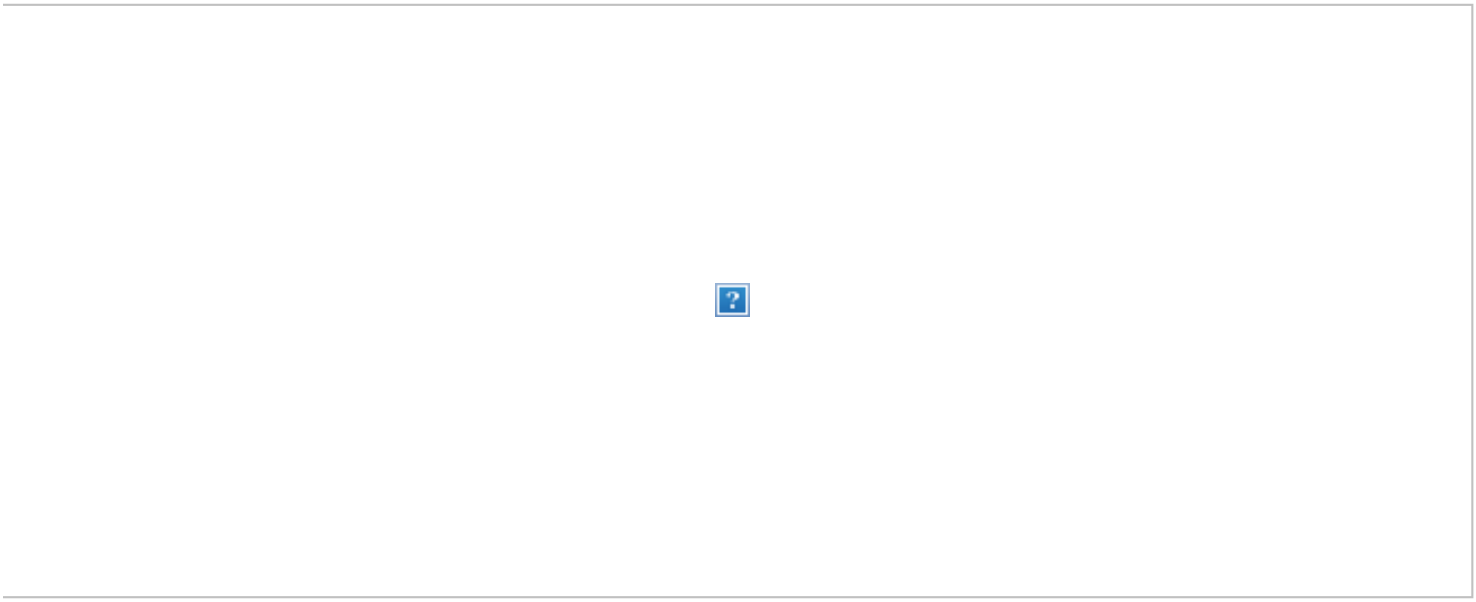


Figure 1: Example problem-decomposition tree for part of the plan for installation of a rootkit on a computer system; left branches satisfy preconditions and right branches handle postconditions of actions.

III. IMPLEMENTING COUNTERPLANNING

A. Ploys for Counterplans

A simple way to obstruct a plan is to delay execution of its steps. For instance, every order the computer system gets from a suspicious user could be delayed to take twice as much time as normal. We are exploring this in other research, but this requires some psychological modeling to predict the effect. So here we will focus on the less ambiguous discrete counteractions we can do impede an attack, for which our planner will be helpful.

We need units out of which we can build counterplans. "Ploys" can be identified as atomic changes to states in the plan model that invalidate preconditions of actions there (since these prevent the actions), invalidate overall goals (since these drive planning), or accomplish defined random changes (since these can surprise the planner and impede planning). (Postconditions on actions are inferior ploys since they do not necessarily affect execution of a plan.) Ploys can involve deletion of facts, addition of facts, or changes of facts to their opposites. We assume here that the facts involved are mentioned somewhere in the plan-action specifications. It is certainly possible to obstruct with new preconditions on actions, such as "downloading file F requires an additional password" in our rootkit-installation model. But the number of them is usually unbounded; they often depend considerably on the domain; and most have the same effect as explicit preconditions already in the model. So they are not very practical and we ignore them here.

Ploys may necessitate changes to more than one fact simultaneously. For instance, if we add a fact that a file F on site S is compressed, we must add the fact that F is on S if that fact is not asserted already. Similarly, attackers generally work sequentially so a file-transfer connection to one site implies no simultaneous connection to another. So for any proposed deletion or addition of a fact to a state, we must determine all implied facts and implied negations of facts. We can infer that A implies B if whenever A occurs in a state, B also occurs. We can infer that A implies not(B) if whenever A occurs in a state, B does not occur. Such implications obey transitivity (if A implies B and B implies C, A implies C) and contrapositivity (if A implies B, then not(B) implies not(A)) which reduces their required number. As an aid to designers of deception strategies, we have a tool program that examines the states resulting from a set of simulation runs of the planner and suggests implications using these ideas. To be reasonable, it only considers for implications those facts that occur in at least one state and are missing from at least one state. Although a finite number of runs of a stochastic model may miss rare phenomena and the implications derived are not guaranteed, its results can be quickly checked by the designer, saving them time.

Several principles can prune possible ploys. It makes no sense to apply a ploy to a state if the ploy is a normal plan action for that state, one that helps accomplish goals there. Facts added by a ploy that are already true in a state, or facts deleted by a ploy that are already false in a state, can be ignored to simplify the ploy (or eliminate it if all changes are removed). It is not generally possible to delete mental consequences of actions (such as human knowledge of particular vulnerability), though strenuous efforts at disinformation can sometimes work. It makes no sense to delete, add, or change an order with a ploy since computers are supposed to be precise agents of human needs and this kind of deception is easily apparent. Similarly, it makes no sense to delete a report since a human is expecting that report. But false reports are useful ploys, either events that never occurred (like reporting that the network crashed) or events that occurred differently (like reporting that the network is slow when it is not); false reports eventually cause precondition violations and failure during execution of actions. We assume a false report must relate to a precondition or postcondition of the preceding action. For our rootkit-installation example after pruning, we found 588 implications, 16 fact-deletion ploys, 28 fact-addition ploys, and 16 ploys with both fact deletion and addition.

Further decisions must be made in implementing a ploy ("performing" it). Good deception is like stage magic in skillfully directing the viewer's attention [15], and there are several "ploy presentation strategies":

1) Honesty: We apply the ploy and tell the attacker we are doing it because we want to preserve system security. For instance, when asked to download a file we say we won't because the filename sounds suspicious.
2) Stealth: We apply the ploy but do not say we did. The attacker can find out later when an expected precondition is violated. For instance, after a file is downloaded and the user is doing something else, we delete it. This is only possible when the ploy is *not* related to the previous action.
3) Excuse: We apply the ploy and give a false reason why. This works best with fact-deletion ploys. For instance, when asked to download a file, we say we can't get permission.
4) Equivocation: This is a true but deliberately misleading excuse. For instance, when asked to download a file, we say its system has been refusing connections, knowing full well that it isn't now.
5) Lying: We apply the ploy but claim falsely the result without the ploy that the attacker expects. For instance, when asked to download a file, we don't but say we did.

6) Overplay: We apply the ploy ostentatiously to cover some other ploy. For instance, we could complain vigorously when asked to download a file with a suspicious name, but not at all after the user renames it and tries to download again. However, we could recognize the file from its signature, refuse to download it, and pretend subsequently that the file was corrupted during network transfer. This could fool attackers because they think they have outfoxed you, and exploits the tendency of people to see patterns too easily [16].

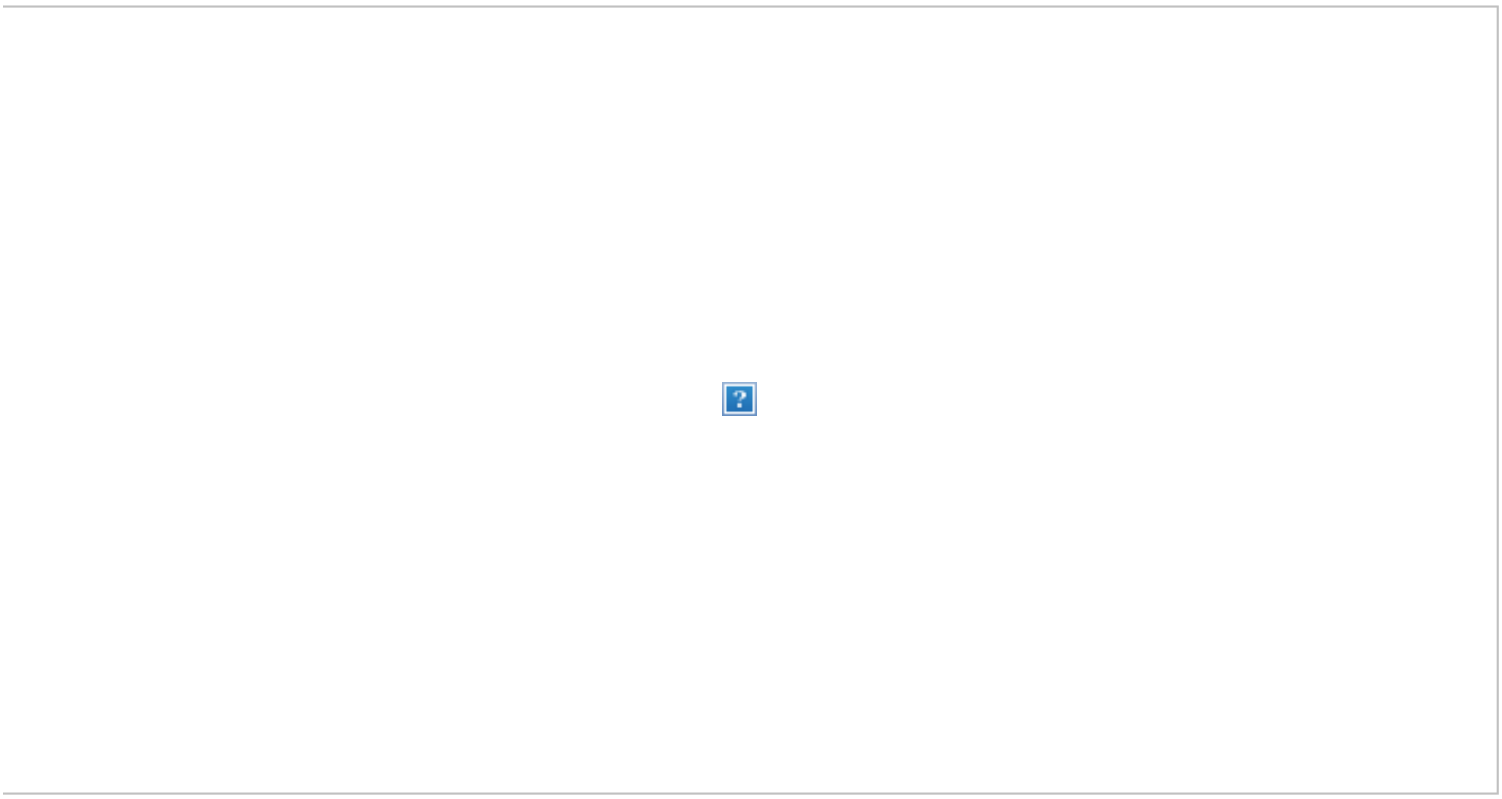
While there are issues in ethics concerning all lies, many moral systems consider it acceptable to lie in crisis situations (as when your software will be destroyed), to an enemy (like a cyber-attacker), or when you have been lied to yourself (as when an attacker masquerades as a legitimate user, the usual network attack) [17]. But the issues about what is acceptable cyber-deception, especially for legitimate users, are complex and need much further examination.

B. Choosing When to Apply Ploys

An exhaustive approach to finding the best ploys is to systematically consider each one for each state, an analogy to approximating partial derivatives of cost functions. The most troublesome ploy-state combinations can be executed. If the base plan involves S states, and the average state can be modified in M ways, we must replan MS times to assess the effects of each change. A replanning from state K from the end of the base plan involves selecting an action at each of K states, so systematic counterplanning requires about $0.5MS^2$ action selections for a single linear plan. Since MECOUNTER is intended for stochastic models, we must repeat each of these runs many times for the same starting state and goal to assess a ploy, and then we must try different starting states and goals, so the total number of action selections is multiplied several times more, if indeed we are to be systematic. If this is too many cases, we could just randomly sample the space of ploy-time possibilities. We will get a distribution of costs associated with each run with a ploy; distributions that are significantly slower than normal reflect promising ploy options. But we found this approach was impossibly slow with our 48-step rootkit-installation model.

However, much of this analysis is unnecessary. We can collapse analysis of identical states in different runs, building a Markov state model with state-transition probabilities. State equivalence can be qualitative to further reduce the possibilities. We ran our rootkit-installation model 500 times (in about 700 minutes) with random starting states drawn from the set of 3072, the defined random postconditions, and durations defined by probability distributions (attacker and system can do some actions in parallel, so timing affects the possible states) to get 10,276 distinct states in 21,720 total states which cut our work in half. Figure 2 shows the Markov model for ten runs of the simpler attack model used for Figure 1. Transitions between nodes are labeled with their actions and probabilities; nodes are labeled with their state numbers (within the boxes), and expected time to the goal state (besides the boxes) was obtained by averaging results of several runs.

A second efficiency idea is to infer good ploys from similar situations using inheritance and inheritance-like rules. Typically, a ploy directly affects only a few actions in a plan. Thus the ploy will be feasible in most states ? this is what makes good planning still valuable in environments with significant uncertainty. Let ploy C



19

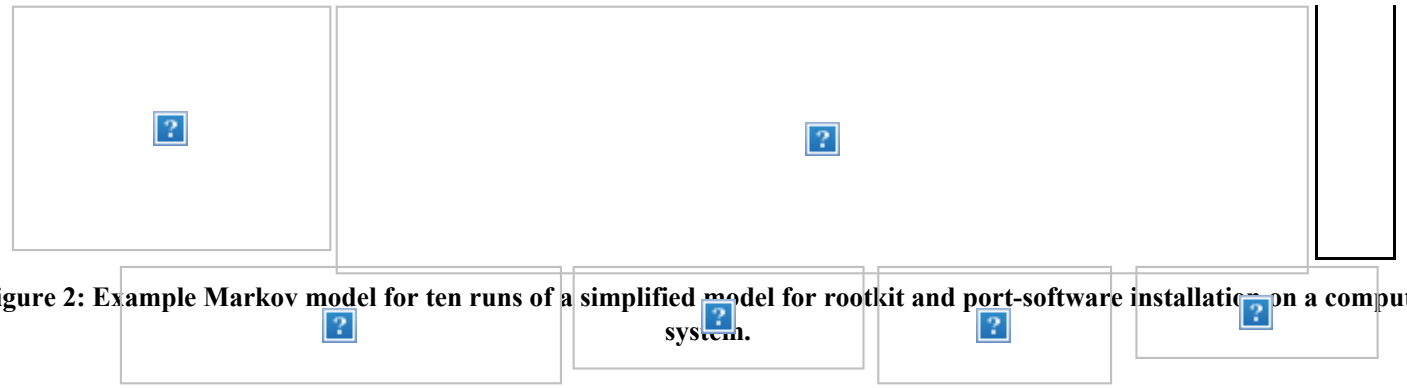


Figure 2: Example Markov model for ten runs of a simplified model for rootkit and port-software installation on a computer system.

involve deletion of facts D and addition of facts A. Let the "fixplan" be the planner's (attacker's) plan to respond to C. Then these inference rules apply.

- "Backward temporal suitability inheritance": State S has the same fixplan for C that applies to later state S2 if all actions between S and S2 do not require any facts in D, do not require the negation of any facts in A, do not add any facts in D, and do not delete any facts in A.
- "Downward fixplan inheritance": A state S2 that is an intermediate state in action A inherits the fixplan for state S that precedes action A if S inherits the fixplan for the state S3 immediately after A.
- "Ploy fixplan following": If ploy C can be applied at state S with fixplan F, and F starts with the reverse of action A, any state S2 for which S is the immediate successor due to A has a fixplan consisting of all but the first step of F, provided no postconditions of A are necessary for the rest of F.

For example, consider the ploy of deleting a suspicious downloaded file. A general fixplan for the attacker would be to download it again by establishing a file-transfer connection to a hacker site, locating the directory of the file, transferring it, and closing the connection. In Figure 2, this ploy for file "rootkit" would be useful at state 18 but less so at state 25 when "rootkit" has just been downloaded because the fixplan would be shorter. The suitability of the ploy and fixplan will inherit temporally backward to states 17, 28, 16, 23, 22, 30, 27, 29, and 26. The suitability will also inherit to the subparts of decompressing "secureport" in state 22 such as setting the destination directory. Ploy-fixplan extension would occur from state 15 to state 16, so the fixplan for 15 could just omit the first step ("ftp") for the fixplan for 16.

Fixplan planning could either try to undo the effects of a ploy before resuming the original plan ("local repair") or replan to achieve the original goals from the state after the ploy ("global repair"). Local repair is faster, but may be suboptimal because the ploy may affect reasons for the actions in the original plan while not affecting its adequacy. Adequacy usually suffices because people are creatures of habit and often prefer to persist with now-suboptimal original plans. We used global repair in our experiments because of the suboptimality and because caching of subproblems can make it efficient. We also assumed no counter-counterplanning is done by the attacker since we try hard to make counterplanning invisible.

For our full example of rootkit installation with 500 simulation runs, there were 70 ploys for the 10,276 possible states, providing 616,560 ploy-to-state match possibilities. Only 19,112 of these were needed for initial analysis, and temporal inheritance expanded this to 104,971. Of these, only 18,835 (3.0%) were found meaningful and damaging to the attacker. The average fixplan needed 6.6 steps to return the attacker to a known state. Combining this data with the benefits of the Markov model, we have reduced our counterplanning effort to 1.4% of the cost of an exhaustive test of changes to the simulation, without eliminating any useful possibilities.

C. Building the Counterplan

After promising ploys and their best times of application to the plan have been computed, we must choose a good set of them to comprise the counterplan. Our benefit metric must reflect the results of numerous "red team" experiments that showed that well-trained and determined attackers usually eventually gained control of target computer systems. So our only realistic objective is to delay the attacker maximally. But we cannot just do this by implementing every ploy every time because the more ploys we use, the more likely the attacker will suspect they are being fooled. Then the more likely the attacker will give up and log off, and we cannot then delay

them further.

So we need to calculate an expected amount we will delay an attacker by a ploy at a state in the plan. Several factors must be considered:

- The likelihood that a ploy fools someone is a monotonically decreasing function of the a priori probability of the ploy actions occurring during normal activity. We can use a sigmoid-shaped function as in neural-network methods for similar problems.
- Some a priori probabilities of ploys occurring during normal activity are supplied by the plan model (like the probability the network connection is not working), and a default (we use 0.02) can be assumed for others.
- Classic decision theory says then that the expected delay to the attacker is the product of the excess time induced by the ploy's fixplan (repair plan) and the probability they are fooled into incurring it. The excess time is the expected duration of the fixplan plus the cost-to-goal of its final state minus the cost-to-goal of the state to which the ploy was applied (averaging quantities over several runs for better estimation).
- Not all ploys succeed (e.g. an attacker may not notice a false report). We must multiply the expected delay by the ploy-success probability. When a several facts are implied by a ploy, we use only the probability of the initiating ploy since the other probabilities are dependent.
- When a ploy likelihood is not independent of the likelihood of a previously-chosen ploy (such as an FTP-failure ploy after a login-failure ploy, both of which can be explained as "network problems"), the probability of both is found assuming disjunctive independence (the inverse of the product of the inverses) as a simple way to model the effect.
- We may not be certain that a sequence of actions constitutes an attack ? it might just be unusual behavior of legitimate users. Then ploys would delay the legitimate user, and we must decrease the ploy value accordingly. A simple model can assume a constant rat[illegible] 25 [illegible]etween the value of legitimate user's time to the value of attacker's time, a ratio that can be set based on the security needs of the computer facility. A model also needs a probability that the sequence of actions by the user up to ploy i is malicious, which can be obtained from an intrusion-detection system.
- If the expected delay induced on a user by a ploy is , then the expected benefit to us when we are not certain the user is illegitimate is: provided that we assume that the delay to a legitimate user in being required to fix the damage of a ploy is identical to the delay to an attacker, something often but not always true. Since this quantity must be nonnegative to justify deception, a necessary condition for any ploys at all is that .

Putting this all together, the simplest acceptable model of the expected benefit due to ploy i at state j in a counterplan is:

where:

- is the frequency of the occurrence of state j in a randomly chosen complete attack plan;
- is the probability that a user beginning an attack plan including j actually reaches state j (without being discouraged, etc.);
- is the probability that ploy i succeeds;
- is the expected time added to the plan by the ploy i at state j;
- r is the ratio of the value of a delay of one unit of a legitimate user's time to the delay of one unit of an attacker's time;
- is the probability the behavior observed so far is part of the attack plan for which the ploys were generated;
- f(x) is a sigmoid function between 0 and 1 (we used in experiments);
- is the a priori probability of the ploy occurring at random;
- is the association probability between this ploy and the most-similar previous one.

We use a greedy search to choose the best set of ploys since it performed almost as well as branch-and-bound in test runs. That is, we choose the best ploy-time pair, then the second best of the remaining ploy-time pairs after adjusting probabilities, then the third, and so on until ploy-time delays no longer exceed a fixed threshold. We considered counterplanning several "runs" simultaneously, each representing a separate attacker's application of their attack plan, but felt consistency is more effective in deception than surprising the attacker the next time they log in. So we recommend the same counterplan for each attack.

Final decisions that must be made for the counterplan are in the choice of the ploy presentation strategies discussed in section III.A. If expected delay is the ploy assessment metric, the tactics of honesty, excuses, and equivocation are suboptimal because the attacker is

thereby quickly informed of unexpected events and can counter-counterplan right away. (More sophisticated models could include a "trustworthiness" factor that would be enhanced by such strategies, but it is hard to estimate.) Overplaying is very action-dependent and hard to implement. That leaves only stealth and lying as our choices, and they are mutually exclusive since lying only makes sense after actions related to the ploy and stealth only makes sense otherwise. Thus our choice of ploy presentation strategy is determined for each ploy.

In applying this theory to our root-compromise model, additional assumptions were made. Two clues as to an attack are very strong, buffer overflows and unexpected obtaining of administrator status, so  for these and subsequent states and 0 otherwise. We chose  = 0.95 if an identical ploy is done in this run, and 0.9 if a ploy with the same predicate name done is in this run.

We discovered with MECOUNTER analyzing the results of the 500 simulation runs that the best way to impede the root-compromise plan was to do 22 ploy-state combinations. These changes included some obvious ones (like faking general network problems and deleting the rootkit after the attacker has logged out) and less obvious ones (such as issuing false reports that the downloaded files are now recompressed). A more complex ploy proposed for the state just after the user has overflowed a buffer to gain administrator status was:

```
Delete the connection at port 80, delete the fact the buffer is overflowed, add the fact
there are problems in the file system, report the connection is terminated, delete
administrator status.
```

In general, all ploys proposed by MECOUNTER were helpful and reasonable ploys for defense of the system. Forthcoming research will analyze our counterplanner performance systematically.

IV. REFERENCES

[1] McClure, S., Scambray, J., and Kurtz, G., *Hacking Exposed: Network Security Secrets and Solutions, third edition*. New York: McGraw-Hill Osborne Media, 2001.

[2] The Honeynet Project, *Know Your Enemy*. Boston: Addison-Wesley, 2002.

[3] Fowler, C. A., and Nesbit, R. F., Tactical deception in air-land warfare. *Journal of Electronic Defense*, Vol. 18, No. 6 (June 1995), pp. 37-44 & 76-79.

[4] Dunnigan, J. F., and Nofi, A. A., *Victory and Deceit, second edition: Deception and Trickery in War*. San Jose, CA: Writers Club Press, 2001.

[5] Michael, B., Auguston, M., Rowe, N., and Riehle, R., Software decoys: intrusion detection and countermeasures. Proc. 2002 Workshop on Information Assurance, West Point, NY, June 2002.

[6] Carbonell, J., Counterplanning: A strategy-based model of adversary planning in real-world situations, *Artificial Intelligence*, Vol. 16, 1981, pp. 295-329.

[7] Applegate, C., Elsaesser, C., and Sanborn, J., An architecture for adversarial planning, *IEEE Transactions on Systems, Man, and Cybernetics*, 20 (1), January/February 1990, pp. 186-194.

[8] Carley, K., Inhibiting adaptation, *Proc. Command and Control Research and Technology Symposium*, Monterey CA, USA, June 2002.

[9] Lowry, J., An initial foray into understanding adversary planning and courses of action. Proc. DARPA Information Survivability Conference & Exposition II, Anaheim, CA, June 2001, vol. 1, pp. 123-133.

[10] Cohen, F., Simulating cyber attacks, defenses, and consequences. http://all.net/journal/ntb/ simulate/simulate.html, May 1999.

[11] Rowe, N., and Galvin, T., An authoring system for intelligent procedural-skill tutors, *IEEE Intelligent Systems*, 13(3), May/June 1998, pp. 61-69.

[12] Rowe, N., and Schiavo, S., An intelligent tutor for intrusion detection on computer systems, *Computers and Education*, 31, 1998, pp. 395-404.

[13] Rowe, N., and Andrade, S., Counterplanning for multi-agent plans using stochastic means-ends analysis, IASTED Artificial Intelligence and Applications Conference, Malaga, Spain, September 2002, pp. 405-410.

[14] Weiss, G., (Ed.), *Multiagent Systems*. Cambridge, MA: MIT Press, 1999.

[15] Tognazzini, B., Principles, techniques, and ethics of stage magic and their application to human interface design. Proc. Conference on Human Factors and Computing Systems (INTERCHI) 1993, Amsterdam, April 1993, pp. 355-362.

[16] Heuer, R. J., Cognitive factors in deception and counterdeception. In *Strategic Military Deception*, ed. Daniel, D. C., and Herbig, K. L., New York: Pergamon, 1982, pp. 31-69.

[17] Bok, S., *Lying: Moral Choice in Public and Private Life*. New York: Pantheon, 1978.

Acknowledgement: This work is part of the Homeland Security Leadership Development Program supported by the U.S. Department of Justice Office of Justice Programs and Office for Domestic Preparedness. The views and conclusions contained herein are those of the author and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the U.S. Government. *This paper appeared in the Proceedings of the 2003 IEEE Workshop in Information Assurance, West Point, NY, June 2003.*

---

[1]