2004-06

Research on Deception in Defense of Information Systems

Rowe, Neil C.

Monterey, California. Naval Postgraduate School

Research on Deception in Defense of Information Systems

Neil C. Rowe, Mikhail Auguston, Doron Drusinsky, and J. Bret Michael

U.S. Naval Postgraduate School
Code CS, 833 Dyer Road
Monterey, CA USA 93943
ncrowe, maugusto, ddrusins, and bmichael @ nps.navy.mil

Abstract

Our research group has been broadly studying the use of deliberate deception by software to foil attacks on information systems. This can provide a second line of defense when access controls have been breached or against insider attacks. The thousands of new attacks being discovered every year that subvert access controls say that such a second line of defense is desperately needed. We have developed a number of demonstration systems, including a fake directory system intended to waste the time of spies, a Web information resource that delays suspicious requests, a modified file-download utility that pretends to succumb to a buffer overflow, and a tool for systematically modifying an operating system to insert deceptive responses. We are also developing an associated theory of deception that can be used to analyze and create offensive and defensive deceptions, with especial attention to reasoning about time using temporal logic. We conclude with some discussion of the legal implications of deception by computers.

This paper appeared in the Command and Control Research and Technology Symposium, San Diego, CA, June 2004.

A theory of deception in information systems

Deception is a key feature of human social interaction not much studied in either information security or artificial intelligence. In one part of our project, we are developing testable computational models of deception including the major sub-phenomena or trust, expectation, suspicion, surprise, deception plans, and manufactured patterns (Bell & Whaley, 1991). Such a theory can be used to explain both offensive deceptions (to gain some advantage) and defensive deceptions (to foil someone else's plans). Defensive deception could fool attackers into thinking our systems are not worthy of attack, or thinking incorrectly they have hurt us. It can be very convincing because people expect computer systems to be obedient servants. Paradoxically, deception against bad people can increase the trust level of good people, much like how an inquisitive police force increases social trust. Deceptions can also produce useful side effects like identification of the attacker by planting phone numbers we can trace or installing "spyware" on the attacker that will report their activities.

Producing a convincing defensive deception requires careful planning because people can easily recognize patterns suggesting it. So deception needs to be applied sparingly and thoughtfully based on a theory of trust and "suspiciousness" and its psychological consequences (Miller & Stiff, 1993). We are exploring applying it in two different ways. One will use a theory of "generic excuses" that will frustrate the attacker but appear to be highly plausible. The other way will use a theory of "counterplanning" that will frustrate the attacker in highly effective but hard-to-notice ways. Counterplanning can make use of artificial-intelligence planning

research but has unique issues too.

We must be careful not to apply deceptions to non-malicious users who accidentally act suspicious; thus intrusion-detection systems need to search for consistent patterns in lower levels of suspiciousness, a different set of criteria than employed today. Deception also raises ethical issues to be explored. In general, deception is judged ethical against serious consequences (e.g. destruction of a computer system in an attack) which may apply to cyberterrorism (Tan, 2003).

One part of our research on the theory of deception (Rowe and Rothstein, 2004) looked at the classic deception methods of conventional warfare: concealment, camouflage, ruses, demonstrations, feints, false and planted information, lies, displays; and insight. 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 may turn out to be easy to confirm most statements about cyberspace with sufficient time.

This leaves lies, displays, and insights as potential defensive tactics. Moreover, they are powerful tactics just beginning to be explored. 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. (Cohen, 1999) gives an interesting further taxonomy of deception methods.

Implementation of automatic deceptions and deception planning

We have implemented several demonstration systems for deceptive software. One was a Web browser that delays answering requests when it suspects it is under attack (Julian et al, 2003). Another was a file-transfer utility that pretends to succumb to a well-known buffer-overflow attack (Michael, Fragkos, and Auguston, 2003).

Another was a demonstration Web site including a number of deceptive practices. The site looks like a file directory hierarchy (see Figure 1); users can click on names of directories to see subfiles and subdirectories with fake dates and sizes, and can click on names of files to see captioned image data. Some of the files appear to be encrypted like that on the top right of Figure 1. Authorization errors are issued for some other files, and loading time is proportional to the listed number of bytes of the file but is exaggerated. However, the directories and files are all fake: Names are chosen randomly from a list of all the Web page names at our school, and contents of the accessible and readable files are generated by randomly choosing a picture and its caption from a school Web page. The picture at the bottom of Figure 1 was allegedly the file /root/Code05/WAT/policy/
Old_pages/23_AdvancesNonPhotoRealisticRendering/Locations/NavalPostgraduate
School/images/aries_thumbnail.html, which suggests that this Aries vehicle has something to do with

photographic rendering, policy, and wide-area timesharing, none of which is true. The site functions as a kind of "honeypot" for spies, encouraging them to find spurious connections between seemingly meaningful things, while the encryption and the authorization errors suggest hidden secrets.

We also implemented a software tool MECOUNTER (Rowe, 2003) to systematically "counterplan" (Carbonell, 1981) or find ways to foil a computer-system attack plan. This work is built on previous research in automated planning from constraints, and exploits a hierarchical and probabilistic attack model. The model has two agents with different goals, an attacker and a computer system. To test our counterplanning, we tested it on a particular model we built for attacks 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 nineteen action types which can be instantiated to ninety-three distinct actions. The specifications cover a possible 115 distinct facts (and their 115 negations) in states, preconditions, and goals, and allow for thirteen 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, forty-eight steps are sufficient to achieve the goal from the most complex start state.

Our approach to counterplanning is to first find all possible atomic "ploys" that can interfere with the plan. Ploys are simple deceits the operating system can perform such as lying about the status of a file. We analyze ploys as to the degree of difficulty they cause to the attack plan wherever they can be applied. This is more complex than it seems because 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. 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

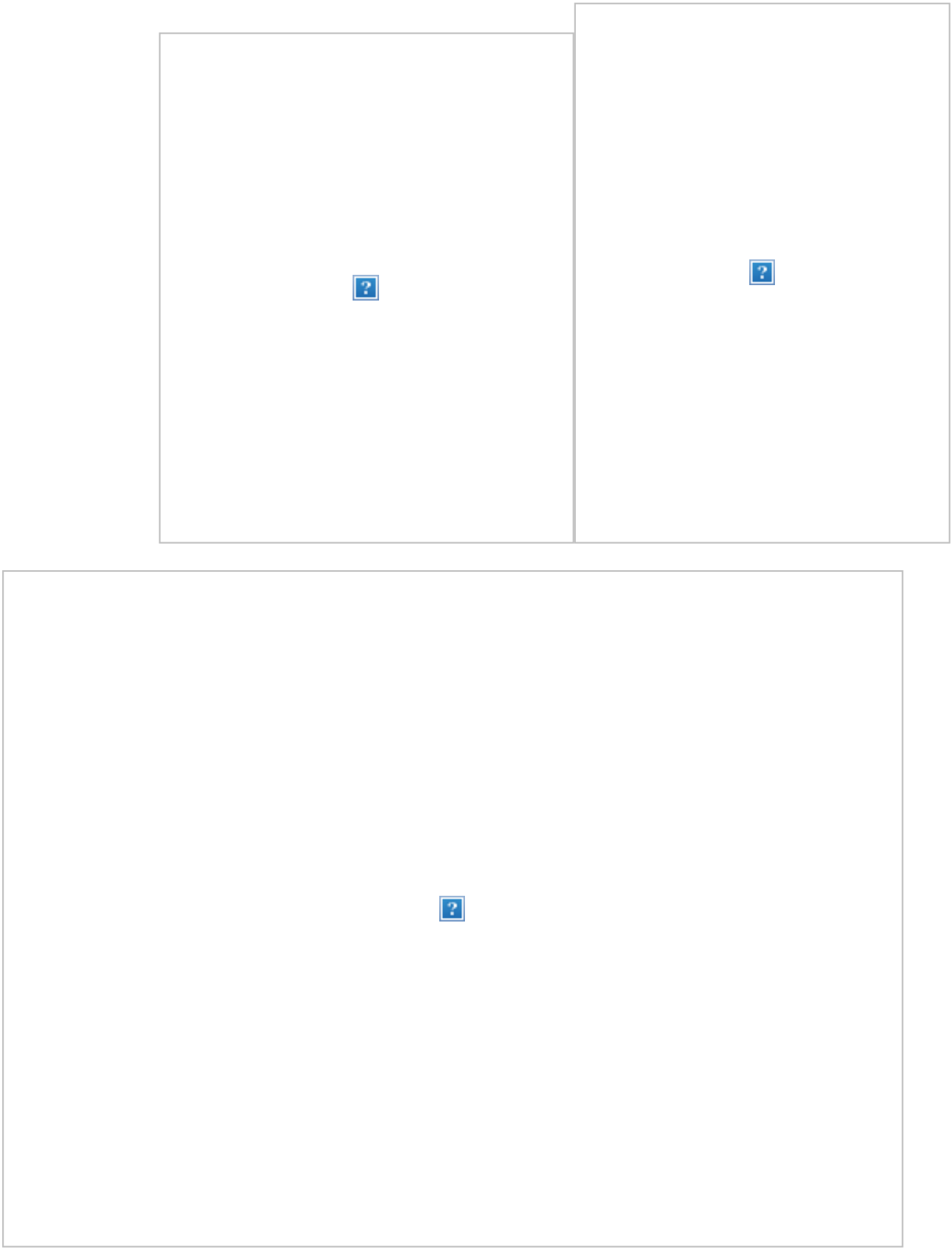

Figure 1: Example from the fake file directory deception.

examines the states resulting from a set of simulation runs of the planner and suggests implications.

Several principles then can be used to 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. 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 strong efforts at disinformation can sometimes work. It makes no sense to delete, add, or change an order with a ploy since computers are should be 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.

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 accomplished by a modified operating system. An exhaustive approach to finding the best ploys for the counterplan is to systematically consider each one for each state, an analogy to approximating partial derivatives of a cost function. The most troublesome ploy-state combinations for the attacker can be executed. However, much of this exhaustive 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 (allowing that attacker and system can work in parallel) to get 10,276 distinct states in 21,720 total states, which cut our work in half.

A second efficiency idea is to infer good ploys from similar situations using inheritance and inheritance-like rules. We devised useful rules for "backward temporal suitability inheritance", "downward fixplan inheritance", and "ploy fixplan following". 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 reduced our counterplanning effort to 1.4% of the cost of an exhaustive test of changes to the simulation, without eliminating any useful possibilities.

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. The simplest acceptable model of the expected benefit due to ploy i at state j in a counterplan we found was:

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 used 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 expected ploy time delays no longer exceed a fixed threshold. We use the same counterplan for each attack because consistency is important in deception. 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). In general, all ploys proposed by MECOUNTER were helpful and reasonable for defense of the system.

Prototype design of an intrusion interception language CHAMELEON

We also developed and implemented the first version of the rule language CHAMELEON (Michael, Fragkos, and Auguston, 2003) for experiments with software decoys in Red Hat Linux 6.2 environment. The implementation is based on the Linux kernel wrapper tools developed by NAI (Ko et al, 2000). This provides a more general approach to implementing deceptions in the form of a tool to systematically modify large software systems like operating systems.

This first version of rule language provides functionality for signature-based intrusion detection and countermeasures, like delay or total blocking of kernel command execution, and error message generation. Intrusion signatures can be specified as regular expressions over kernel subroutine calls and can involve also values of input and output parameters of kernel calls. The CHAMELEON compiler takes the rule source code and generates wrapper code in the Wrapper Definition Language (WDL) in addition to a script file necessary to compile the WDL code into executable and install and activate the wrapper in the kernel space. The Appendix shows the formal syntax of the lang[25].

Wrapper Support System (WSS) monitors install wrappers and creates a new instance of a wrapper for each process executed within the system. Since event patterns can be accompanied by actions written in the C language, it becomes possible to program arbitrary countermeasures in response to the detected intrusion activities.

As an example, the rule below detects a sequence of file operations, including opening the file, some read/write operations, and closing the file. Each event will cause a message to be sent to the system-monitoring log file with the file name involved. The **pre** and **post** options indicate whether the corresponding action is performed before or after the matching kernel call. $path is one of several predefined

access variables that provide values of kernel call parameters.

```
/* simple rule example */
    R1 :
    detect
 open pre   { wr_printf("open file %s", $path) ; }
 (          read pre { wr_printf("read file %s ", $path); }   |
           write pre {wr_printf("write file %s ", $path);}       ) *
            close post { wr_printf( "file %s  closed", $path); }
```

Besides executing arbitrary C code, the actions performed by wrapper rules may include two simple primitives: DENY, which will prevent execution of corresponding kernel call, and DELAY(N), which delays execution of kernel call by N milliseconds.

The CHAMELEON framework supports signature-based intrusion detection and a basic countermeasures mechanism. Future work includes an extension to accommodate anomaly-based statistical methods for detecting system misuse. One idea we are exploring is to use a third-party intrusion detection system that communicate with the wrappers (Monteiro, 2003), which appears a promising way to cut development costs significantly.

Run-time monitoring of Knowledge Temporal Logic specifications

Another aspect of our research is modeling the time sequence of events in deceptions and attempts to foil them. We have found temporal logic quite useful. (Pnueli, 1977) suggested using Linear-Time Propositional Temporal Logic (LTL) for reasoning about concurrent programs. LTL is an extension of propositional logic where, in addition to the well-known propositional logic operators, there are four future-time operators (◊-*Eventually*, ÿ-*Always*, *U-Until*, O-*Next*) and four dual past-time operators. (Fagin et al, 1995) developed a general framework for modeling knowledge and computation and apply it towards reasoning about knowledge in distributed systems. This framework, which we refer to as *knowledge logic,* is based on propositional modal logic.

A card-game example was introduced in (Fagin et al, 1995). It consists of two players (1 and 2) and three cards (A, B and C). Each player takes one card and the third card is left face down on the table. A possible world is described in the formal model by the cards that the players hold. If player 1 holds A and player 2 holds B we denote the world as (A, B). Hence there are six possible worlds (A, B), (A, C), (B, A), (B, C), (C, A) and (C, B). Now if the player 1 has card C then he clearly thinks that there are two possible worlds, one where player 2 holds card A and in the other where player 2 holds card B. The model is represented visually in Figures 2 and 3.

```
<WORLD NAME="AB">
<PL CODE="player1HasA" TRUTH="1" />
<PL CODE="player1HasB" TRUTH="0" />
<PL CODE="player1HasC" TRUTH="0" />
<PL CODE="player2HasA" TRUTH="0" />
<PL CODE="player2HasB" TRUTH="1" />
<PL CODE="player2HasC" TRUTH="0" />
</WORLD>
```

Figure 3: Representation of the world AB in the card games? knowledge file.

Run-time Monitoring of KTL

Run time Execution Monitoring (REM) is a class of methods of tracking temporal requirements for an underlying application. Of particular interest are on-line REM methods where temporal rules are evaluated without storing an ever-growing and potentially unbounded history trace. (Drusinsky, 2000) and (Drusinsky and Shing, 2003) describe the DBRover and Temporal Rover REM tools. These tools support REM and simulation of properties written in LTL (future-time and past-time) with counting operators, as well as real-time and time-series constraints. We have extended the specification language used by these tools with two new operators, *knows* (**K**) and *believes* (**B**). In addition, the user specifies a possible worlds model, such as in Figure 2, using an XML *knowledge* file. The Rover tools assume that agent relations, such as the Figure 2 relation (<C,B>,<C,A>, player-1), are transitive. The end user can select whether the agent?s relation is reflexive and symmetric, though we have only experimented with a full equivalence relation.

Consider the KTL rule for the card game: R_1=à£3**K**1**K**2ÿ£2{*player1HasCardB* Ú *player1HasCardC*}, i.e., R_1 states that "within three cycles player-1 will know that player-2 knows that for two consecutive cycles player-1 has card B or C." When the Rover tools monitor rule R_1 they repeatedly read a stream of propositional statements about the status of each player. In the implementation of the card game demo we chose to represent each such reading as a 6-tuple <*player*1*HasCardA, player*1*HasCardB*?

*player*2*HasCardC>;* the reading of a new tuple constitutes a cycle from the temporal perspective. The domain of propositions in the input stream (six propositions in our case) and the domain of propositions in the possible worlds model are related by knowledge file which assigns a truth value to propositions in each world. Listing 1 contains the specification of the world *AB* in knowledge file for the card game.

Consider the following sequence of readings (i.e., cycles), where *XY* represents the world where player-1 has card *X* and player-2 has card *Y*: s=*AB.AC.BC.BA*. Consider for example world *AB,* which is the world of cycle 0. Using the possible worlds model of [FHMV], *AB* |- $\mathbf{K}_1$ r if r holds in all worlds that are equivalent to AB for player-1, i.e., in worlds *AB* and *AC*. *AB* |- r i.e., *AB* |- $\mathbf{K}_2$ j if j holds in all worlds that are equivalent to AB for player-2, i.e., in worlds *AB* and *CB*. However, *player1HasCardB* Ú *player1HasCardC* is not true in world *AB*. Hence the monitoring result for R_1 in cycle 0 is *false.* Consider the world for cycle 2, namely *BC*. r= $\mathbf{K}_2$ j must hold in both (i) *BC* and (ii) *BA*. For (i) j must hold in *BC* and *AC*, and for (ii) j must hold in *BA* and *CA*. One of those four worlds*, AC,* does not satisfy *player1HasCardB* Ú *player1HasCardC* resulting for R_1 in cycle 2 is false.

Consider the slightly modified rule R_2=à£3$\mathbf{B}_1\mathbf{K}_2$ÿ£2 {*player*1*HasCardB* Ú *player*1*HasCardC*}. Using the same input sequence s, we examine R_2 for the world for cycle 2, namely *BC*. r= $\mathbf{K}_2$ j must hold in either (i) *BC* or (ii) *BA*, where (i) and (ii) are described above; case (ii) holds in cycle 2. The world for cycle 3 is BA where r= $\mathbf{K}_2$ j must hold in either (i) *BC* or (ii) *BA,* which is identical to cycle 2. Hence, r holds for two consecutive cycles starting in cycle 2; therefore R_2 is true.

Modeling the WWII "Man Who Never Was" Deception Ploy

Let us apply this to a classic military example from (Dunnigan and Nofi, 2001). In the spring of 1943 the Allies began to consider options for the invasion of Europe. Strategically located in the Mediterranean, Sicily was a good target. However, due to various obstacles facing the Allied command it was decided to not invade Sicily but rather to form a complete set of fake plans for another invasion site and time and then convince the Germans of this plan. A plan named ?Operation Mincemeat," was created in which a British spy would be ?captured? with the false documents. This spy was Major Martin, a corpse. Martin was provided with false papers in a briefcase attached to his body to give a false view of when and where the invasion would occur. A 30-year-old pneumonia victim who had recently died and who resembled a typical staff officer was chosen as Major Martin. The fluid in his lungs would suggest that he had been at sea for an extended period. Fake love letters, overdue bills, and a letter from the Major?s father, and some personal belongings were put on his corpse. Martin?s obituary was in the British papers, and his name appeared on casualty lists. Major Martin was taken to a point just off the coast of Spain where the Allies knew the most efficient German military intelligence network was in place, put in a life jacket, and set adrift. The body soon washed ashore practically at the feet of a Spanish officer conducting routine coastal defense drills. He notified the proper authorities, which notified the Germans. On the return of Major Martin?s body to England, they discovered that his briefcase had been very carefully opened, then resealed. The Germans had photographed every document on Martin?s body and in his briefcase, then released him to the Spanish authorities for return to England, for the English authorities had been demanding return of Martin?s body. One additional less known fact is that during the time that the leading German coroner was in Spain during when the Spanish found Martin?s body. The German coroner was capable of uncovering the real cause of death, pneumonia. However, as it happened the dots were not connected and this coroner did not examine Major Martin?s body.

The logic representation of the "Man who never was" deception ploy is concerned with all possible worlds seen by the three agents, the British, the Germans, and the Spanish. We define the following three Boolean propositions, which together induce a space of eight possible worlds:

- H- represents possible worlds where Major Martin episode is a deception.
- G- represents possible worlds where the German coroner is in Spain and is working on the case.
- M- represents possible worlds where Major Martin drowned.

Hence, for example, w1 = <H, ØG, ØM> is the possible world where the Major Martin episode is a deception, the German coroner is not in Spain, and Major Martin did not drown. This is the possible world the British considered they were in, but in fact, they were unable to distinguish between this world and w2 = <H, G, ØM> and could have very well been in world w2. As in the card example of Figure 2, KL relations are used to aggregate worlds an agent is unable to distinguish. Figure 4 illustrates the KL model for the "Man who never was" deception ploy.

Figure 4: The KL model for the "Man who never was" ploy, where B=British, D=German, S=Spanish.

Deception and the law

Response to attacks

In addition to the technical aspects of realizing cyber-based deceptions, we are investigating social aspects, in particular those relating to conducting deception within the confines of the law. For instance, (Michael, Wingfield, and Wijesekera 2003) demonstrated how the Schmitt Analysis (Schmitt, 1998) can be used to perform an academically rigorous evaluation of the factors affecting a lawful response to a terrorist attack, regardless of whether the attack is effected via kinetic or cyber means. Schmitt Analysis is not intended to provide a mechanical algorithm for solving what are some of the most technically and legally challenging questions we face; instead, it is designed to be a useful framework for analyzing the effect of key factors on the legal nature of an attack and the appropriate response. As such, it provides an invaluable tool for clarifying thought and highlighting areas of misunderstanding or disagreement. It is an excellent basis for training lawyers, technologists, and decision makers in government. Finally, Schmitt?s methodology shows the way for parallel efforts to make more rigorous and more transparent the legal analyses in neighboring areas.

Misuse of deception

Cyber decoys provide a means for automating, to a degree, counterintelligence activities and responses to cyber-attacks. Like other security mechanisms for protecting information systems, it is likely that cyber decoys will in some instances be misused. In the United States, criminal law provides us with analogies for preventing or punishing improper state use of deception, and criminal and civil law give us a range of tools to use against private actors. However, in addition to states, nongovernmental entities and individuals can employ cyber decoys. (Michael and Wingfield, 2003) presented a principled analysis of the use of cyber decoys, in which they explored the absolute minima in terms of customary principles for what might be considered to be acceptable use of deception.

According to (Michael and Wingfield, 2003), software decoys can be used as an airlock between the technology and the law in that the decoys can be programmed with a wide spectrum of options for taking action. Software decoys provide for anticipatory exception handling. In other words, the decoy anticipates the types of inappropriate interaction between itself and processes or objects requesting its services, providing in advance a set of rules for learning about and evaluating the nature of the interaction, in addition to rules for response. One could envision developing policy that places boundaries on the extent and type of deception to be employed, but providing some degree of latitude to the user of decoys to inject creativity into deceptions so as to increase the likelihood that the deceptions will be effective. The boundaries could be used to delineate the thresholds that if breached could result in the misuse or unlawful use of decoys. That is, principled analysis can be used to meet all domestic legal criteria, and set absolute minima in terms of the four customary principles of discrimination, necessity, proportionality, and chivalry.

The U.S. Department of Homeland Security will be responsible for coordinating the protection of both public and private cybernetic property using cyber weapons. There are gray areas in the law regarding how to coordinate counterintelligence activities and countermeasures that need to take place at the intersection of law enforcement, intelligence collection, and military activity. Principled analysis can help here too, but public policymakers will need technically and legally sophisticated advice to manage the best technological defense, including deception techniques, available within the framework of the law.

Tools and support for education

We have been working from two different but complimentary areas: lawful use of software-based deception (Michael and Wingfield, 2003), and cyber law (Wingfield, 2000). To this end, we are developing case studies, with the aid of some automated tools, to teach officials from the law enforcement, intelligence, and military communities how to reason about the legality of responses to terrorist attacks. With appropriate training, information, and analysis (both automated and with man-in-the-loop), it will be possible to reduce the ?gray area? of legal uncertainty to a minimum, and allow the most complete range of effective responses against those who attack a nation?s critical infrastructure.

(Farkas et al., 2004) reports development of THEMIS, the Threat Evaluation Metamodel for Information Systems. THEMIS is a description logic-based framework to apply state, federal, and international law to reason about the intent, with respect to collateral consequences, of computer network attacks. The purpose THEMIS is intended to serve is to provide law enforcement agencies and prosecutors with automated tools for building legally credible arguments, and for network designers to keep their defensive and retaliatory measures?including the use of deception mechanisms?within lawful limits. THEMIS automates known quantitative measures of characterizing attacks, weighs their potential impact, and places them in appropriate legal compartments.

Acknowledgements

Conducted under the auspices of the Naval Postgraduate School?s Homeland Security Leadership Development Program, this research is supported by the U.S. Department of Justice Office of Justice Programs and Office of Domestic Preparedness. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the U.S. Government.

References

J. Bell & B. Whaley, *Cheating and Deception*, New Brunswick, NJ: Transaction Publishers, 1991.
J. Carbonell, Counterplanning: A strategy-based model of adversary planning in real-world situations, Artificial *Intelligence*, vol. 16, pp. 295-329, 1981.
F. Cohen, Simulating cyber attacks, defenses, and consequences, retrieved from http://all.net/journal/ntb/simulate/simulate.html, May 1999.
D. Drusinsky, The Temporal Rover and ATG Rover, Proc. Spin2000 Workshop, Springer Lecture Notes in Computer Science, No. 1885, pp. 323-329.
D. Drusinsky and M. Shing, Verification of timing properties in rapid system prototyping, Proc. Rapid System Prototyping Conference 2003 (RSP'2003).
J. F. Dunnigan and A. A. Nofi, *Victory and Deceit, 2nd edition: Deception and Trickery at War*, San Jose, CA: Writers Press Books, 2001.
R. Fagin, J. Halpern, Y. Moses, and M. Vardi, *Reasoning About Knowledge*, The MIT Press, 1995.
C. Farkas, T. C. Wingfield, J. B. Michael, and D. Wijesekera, THEMIS: Threat Evaluation Metamodel for Information Systems, Proc. Second Symposium on Intelligence and Security Informatics, Tucson, Ariz., June 2004, in Springer Lecture Notes in Computer Science, 2004.

D. Julian, N. C. Rowe, and J. B. Michael, Experiments with deceptive software responses to buffer-based attacks, Proc. 2003 IEEE-SMC Workshop on Information Assurance, West Point, NY, June 2003, pp. 43-44.
C. Ko, T. Fraser, L. Badger, and D. Kilpatrick, Detecting and countering system intrusions using software wrappers, Proc. USENIX Security Symposium 2000, Denver, CO, August 2000, pp. 145-156.
B. Michael, G. Fragkos, and M. Auguston, An experiment in software decoy design: Intrusion detection and countermeasures via system call instrumentation. In Gritzalis, D., di Vimercati, S. D. C., Samarati, P., and Katsikas, S., eds. *Security and Privacy in the Age of Uncertainty*, Norwell, Mass.: Kluwer Academic Publishers, 2003, pp. 253-264.
J. B. Michael, T. C. Wingfield, and D. Wijesekera, Measured responses to cyber attacks using Schmitt Analysis: A case study of attack scenarios for a software-intensive system, Proc. IEEE Twenty-seventh Annual Int. Computer Software and Applications Conf., Dallas, Tex., Nov. 2003, pp. 622-627.
J. B. Michael and T. C. Wingfield, Lawful cyber decoy policy. In Gritzalis, D., Vimercati, S. C., Samarati, P., and Sokratis, K., eds., *Security and Privacy in the Age of Uncertainty*. Boston, Mass.: Kluwer Academic, 2003, pp. 483-488.
G. Miller and J. Stiff, *Deceptive Communications*. Newbury Park, UK: Sage Publications, 1993.
N. C. Rowe, Counterplanning deceptions to foil cyber-attack plans, Proc. 2003 IEEE-SMC Workshop on Information Assurance, West Point, NY, June 2003, pp. 203-211.
V. Monteiro, How intrusion detection can improve software decoy applications, M.S. thesis in Computer Science, Naval Postgraduate School, March 2003.
A. Pnueli, The temporal logic of programs, Proc.18th IEEE Symposium on Foundations of Computer Science, pp. 46-57, 1977.
N. C. Rowe and H. Rothstein, Two taxonomies of deception for attacks on information systems, submitted to *Journal of Information Warfare*, March 2004.
M. N. Schmitt, *Bellum Americanum*: The US view of Twenty-first Century war and its possible implications for the law of armed conflict, *Mich. J. Int. Law* 19, 4 (1998), pp. 1051-1090.
G. Tan, "Cyberterrorism and Cyber Deception", M.S. thesis in Computer Science, Naval Postgraduate School, December 2003.
T. C. Wingfield, The *Law of Information Conflict: National Security Law in Cyberspace*, Falls Church, Va.: Aegis Research Corp., 2000.

Note: Papers and theses of our deception research group are available at www.cs.nps.navy.mil/research/deception/.

Appendix: The syntax of CHAMELEON rule language

Rule::= Rule_name **':' detect** Event_sequence

Event_sequence::= (+ Event_pattern +) |
'(' Event_sequence **')'** [(**'*'** ! **'+'** ! **'?'**)] |
'(' Event_sequence (**'|'** Event_sequence)+ **')'**

Event_pattern::= Event_type [**probe** Condition] [**pre** Actions] [**post** Actions]

Condition::= **'('** C_expression **')'**

Actions::= **'{'** Action * **'}'**

Action::= **DENY** | **DELAY '('** number_of_msec **')'** | C_operator

Event_type::= **ANY** | kernel_subroutine_name

Rule_name ::= identifier

Notes:

- Conditions and actions may include references to the kernel call parameters, like $path.
- The current wrapper system can monitor approx. 150 kernel subroutine names.
- Actions can be performed just before (**pre**) or after (**post**) the detected event.
- The **probe** condition provides for additional context checking when the event is detected.