Inconsistency in Deception for Defense

Vicentiu Neagoe, Matt Bishop
Dept. of Computer Science
University of California at Davis
One Shields Ave.
Davis, CA 95616-8562
email: {neagoe,bishop}@cs.ucdavis.edu

ABSTRACT
The use of deception is one of many defensive techniques being explored today. In the past, defenders of systems have used deception haphazardly, but now researchers are developing systematic methods of deception. The cornerstone of these methods is internal consistency: projecting a "false reality", or "fiction", that the attacker is to accept as reality. We challenge the necessity of this cornerstone, and explore the nature and possible uses of inconsistency in deception as a defense.

Categories and Subject Descriptors
D.4.6 [**Operating Systems**]: Security and Protection – *access controls, information flow controls, invasive software.*

H.1.2 [**Models and Principles**]: User/Machine Systems – *human factors, human information processing, software psychology.*

General Terms
Experimentation, Security, Human Factors.

Keywords
Deception, inconsistency, security, operating systems.

1. INTRODUCTION
The art of deception is invaluable in warfare and conflict. It is used to trick the adversary into taking actions that absorb resources or position resources to make them easier to attack. It is used to sap the morale of the adversary, thereby affecting their ability to initiate actions or to respond to attack. It is also used to conceal actions against the adversary. Sir Winston Churchill summarized the use of deception best when he said, "In time of war, the truth is so precious, it must be attended by a bodyguard of lies."

NSPW 2006, September 19-22, 2006, Schloss Dagstuhl, Germany.

A deception aims to force the target of the deception to perceive a false reality (called a *fiction* for short). Deception may be *consistent* or *inconsistent*. A consistent deception builds a fiction that functions under the rules of reality, so the attacker does not perceive the deception. This is the usual mode, because by controlling the fiction, the deceiver can control the perception of the adversary and, indirectly, affect the adversary's actions. With luck, the adversary will act as the deceiver wishes.

There are abounding wartime examples of deceptions where consistency was critical. For example, the British created a false military officer, faking his death, and causing the body to wash up on the Spanish shore. The body had on it papers indicating a false location for an Allied attack. Hence it was imperative the Germans not realize the deception. In fact, they did not, and diverted their resources to defend a coastline that was never attacked. Interestingly, even after the Allies landed, the German High Command did not realize that the landing was the real attack for several days; they continued to think it was a diversion [15].

Inconsistent deceptions serve an entirely different purpose. Their goal is to discombobulate and disorient the adversary. The adversary will realize that something is wrong, possibly even realize that there is a deliberate attempt to deceive them, but not know which perceptions are of fiction and which are of reality. Thus, they will be confused. Possible responses include trying to figure out which perceptions are of reality, or trying to withdraw from the situation, or acting on random perceptions.

In the "cyberworld," defenders have responded to attacks with deception. The best known case was Cliff Stoll's use of deception to keep an intruder on an international telephone line for several hours, downloading a bogus but interesting file [21]. The authorities were able to trace the call, and broke up a spy ring. Stoll raises the issue of whether defenders should remain open to an intruder once they are detected:

> *Should we have remained open? A reasonable response to the detection of this attack might have been to disable the security hole and change all passwords. This would presumably have insulated us from the intruder and prevented him from using our computers to attack other internet sites. By remaining open, were we not a party to his attacks elsewhere, possibly incurring legal responsibility for damage?*
>
> *... **Closing up ... would have done nothing to counter the ... offenders...***
>
> *... had we closed up, how could we have been certain that we had eliminated the intruder? With hundreds of networked computers at LBL, it is nearly impossible to change all passwords on all computers.*

Perhaps he had planted subtle bugs or logic bombs in places we did not know about. Eliminating him from LBL would hardly have cut his access to MILNET. And, ***by disabling [attacker's] access into our system, we would close our eyes to his activities****: we could neither monitor him nor trace his connections in real-time.*

We agree with Stoll that, in many cases, keeping the attacker in our sight is preferable to cutting her access to our systems and deception provides a means for doing just that.

Cheswick's response to Berferd is another classic in this area [5], and foreshadowed much of the honeypot work. Honeypots and honeynets are systems designed to look like production systems, to deceive intruders into attacking the systems or networks so that the defenders can learn new techniques or better understand the attackers' goals without risking their production systems. Sandboxes and virtual machines limit the actions of the attackers while giving the appearance of unfettered access to resources.

All these deceptions are consistent. Stoll's actions were designed to make the attacker think he had found a system with classified documents on it. Cheswick created a falsity of a system that was old, slow, and vulnerable. Honeypots and honeynets are systems, so they present a consistent reality of being systems. Sandboxes and virtual machines attempt to present themselves as systems; while they may be less successful, especially if the attacker can determine he is running in confinement, they present a consistent falsity (or reality, depending on the goals of the confinement).

Our work begins with the assumption that the attacker has already been detected, so regular users are not affected by these tactics.

We argue that inconsistent deception has been overlooked as a viable tool for defense in the cyber-world. Our paper is organized as follows. First, we review some of the work being done in deception. We then discuss inconsistent deception in more detail, and give a scenario as an example of where inconsistency would be more useful than consistency. We present a model of deception, and discuss the advantages of allowing inconsistency in the deception. We conclude with suggestions for future research.

2. BACKGROUND

Several technologies for providing deception are currently being studied. Software decoys are agents that protect objects from unauthorized access [12,13,14,19,20]. The goal is to create a fiction that the defended systems are not worth attacking or that the attack was successful. The researchers considered tactics such as responding with common system errors and inducing delays to frustrate attacker. The work assumed consistency of the deception.

Red-teaming experiments at Sandia tested the effectiveness of network deception on attackers working in groups [6]. Results indicated that deception mechanisms at the network level could successfully delay attackers for a few hours. Deception apparently wore down those who were exposed to it and prompted some experimental groups to quit the experiments before they were over. Even the teams not being deceived doubted truthful answers and contemplated whether these responses were deceptive. In one experiment designed to lead attackers to perform a sequence of particular actions, researchers developed an attack graph and planned deceptions in such a way that an attacker would follow a predetermined path through the graph. The attackers followed many of the predicted sequences.

Deception at the host level modifies system behavior when an attacker is logged in. One method of implementation is to use an execution wrapper that intercepts program execution requests and optionally runs a different program without the user detecting the switch [18]. The problem here is that many command interpreters perform some of the requested actions directly, without invoking system calls and thereby bypassing the wrapper.

Symantec's ManTrap [11] is a honeypot that automatically generates content such as email exchanged between different users. If an attacker observes these indicators, he will be under the impression that the system has active users. It implements separate environments through cages, so attackers are unaware of their isolation and cannot escape from the cage.

Castelfranchi's research combines cognitive science with computer science and artificial intelligence [2,3,4,8,9]. He explores whether computers can choose a deception ploy that would be cognitively plausible [8] by comparing the computer's selection with those of human subjects. About 50% of the selections matched, indicating that the computer could generate fictions that the human would believe to be reality half of the time.

All the above work either assumes consistency of the deception or tries to implement a deception that is consistent. Apparently, none have explored inconsistent deception.

The goals of deception can be either to keep the attacker on the system in order to trace him, or to have them go away. If we desire for them to go away, we must induce the attacker to lose interest so that she leaves out of her own initiative. If we simply cut them off, they may try to come back through a different way, but if they go away on their own, then they are unlikely to come back.

If we want the attacker to stay, we may distract her by taking her down a path that leads nowhere. For example, we may present her with a fake vulnerability or misconfiguration so that she moves away from her original attack which might have succeeded.

3. INCONSISTENT DECEPTION

Current deception research focuses on consistent deception because of an underlying paradigm that consistency is more beneficial than inconsistency. If the attacker sees something inconsistent, then he will conclude that deception is being used on the system. Previous work assumes it would be bad if deception is detected by the attacker.

Tognazzini discusses stage magic principles for interface design and emphasizes the importance of consistency [22]. "Consistency is the key to conviction... Irregularities destroy naturalness... When naturalness disappears... the spectator's attention becomes vigilant and alert". He supports that vigilance and alertness would be disastrous to deception.

The Greek philosopher Parmenides summed this up succinctly in his law of non-contradiction: "Never shall this be proved—that things that are not, are." [16]. But other Greek

philosophers disagreed. In particular, Heraclitus asserted that contradictions existed and indeed were central to identity: "Not only could it be stated that identity is the strife of oppositions but that there could be no identity without such strife within the identity." [7].

On a philosophical note, Gotesky [10] mentions that inconsistency is an accepted part of life. People cope with it. Inconsistency does not stop people from acting and it can be used as a means for attaining goals. In combat situations, deception is assumed, so the enemy may not believe a message unless a contradiction is asserted. In mathematics, paraconsistent logic allows one to study contradictions formally [17].

This suggests that inconsistent deception may be as useful a defensive technique in "cyberspace" as in real life. It suggests that even if an attacker identifies inconsistencies, she may not conclude that deception is involved, but may attribute the inconsistencies to system problems or errors, or even her own misperceptions.

To present a consistent reality, the deception mechanisms need to keep track of previous answers given. Inconsistency has advantage over consistency because there is no need to keep track of previous responses.

4. MODEL OF DECEPTION

All commands can be classified in two categories: *do* commands and *tell* commands. *Do* commands (write) request some change in the system state. *Tell* commands (read) obtain system information without requesting any change. These are analogous to a mutator and observer in object-oriented programming. *Do* is like a mutator and *tell* is like an observer.

Do commands can be modeled as a 3-tuple: (*command*, *action performed*, *system response*). *Command* is the set of commands available on the system. *Action performed* reflects whether the system faithfully performed the requested action. *System response* indicates whether the system says the request was performed or not. Deleting and modifying files are examples of *do* commands.

Tell commands are modeled as a pair: (*command*, *system response*). *Command* is the set of commands available to the system and *system response* is the information returned by the system as a response to the command. Directory listings and reading files are *tell* commands.

Table 1 depicts all cases for a *do* command. The *action performed* column describes whether or not the system performed the requested action. S*ystem response* indicates whether the system said the command was executed (success) or whether it was not executed (fail). The system will either indicate that the request was fulfilled or give an error indicating why the request was not fulfilled. *Response truthfulness* is true when the *system response* is consistent with the *action performed*.

The *verify response* column gives the system response to a subsequent verification request. Both *do* and *tell* commands can be used to verify whether an action was performed in a previous request.

Lines 3 and 8 represent the behavior of a normal system. In lines 2 and 5, the answers returned in a verification request are consistent with the answers given in the original request. *Verify truthfulness* indicates whether the *verify response* is an accurate reflection of the system.

For each (system response, verify response) combination, there is both a Yes and an No in the "action performed" column, so an attacker cannot determine what happened in the system, even if multiple verifications are made and inconsistent results are obtained.

Through inconsistent deception, a system that normally replies with a status for commands executed can be essentially turned into a system that returns no status for requested actions.

Table 1. Deception Options for a *do* Command

	Action performed	System response	Response truthfulness	Verify response	Verify truthfulness	Consistent
1	No	Success	False	Fail	True	No
2	No	Success	False	Success	False	Yes
3	No	Fail	True	Fail	True	Yes
4	No	Fail	True	Success	False	No
5	Yes	Fail	False	Fail	False	Yes
6	Yes	Fail	False	Success	True	No
7	Yes	Success	True	Fail	False	No
8	Yes	Success	True	Success	True	Yes

The *consistent* column indicates whether the *system response* of the original request and the *verify response* were consistent. In half the cases, replies are consistent with each other. Two cases are consistent because they are part of normal system behavior. The other two cases represent deception.

As an example, Table 2 instantiates the above for a request to delete a file. *Action performed* describes whether the system deleted the file. *System response* indicates the response of the system call to delete the file when the user executes the delete command. The system will either indicate that the request was fulfilled or give an error indicating a reason for failure.

Table 2. Deception options for file deletion.

	File deleted	System response	Response truthfulness	Verify response	Verify truthfulness	Consistent
1	No	Deleted	False	File exists	True	No
2	No	Deleted	False	File gone	False	Yes
3	No	Not deleted	True	File exists	True	Yes
4	No	Not deleted	True	File gone	False	No
5	Yes	Not deleted	False	File exists	False	Yes
6	Yes	Not deleted	False	File gone	True	No
7	Yes	Deleted	True	File exists	False	No
8	Yes	Deleted	True	File gone	True	Yes

An attacker can use both *do* and *tell* commands for subsequent verifications on whether an action was performed. He can verify whether the file was deleted by using the *tell* command "list directory contents". *Verify response* indicates whether the file is present in the directory listing returned. Probing for the existence of a file can be done without obtaining a directory listing. For example, opening the deleted file or requesting its status information with the 'stat' command will indicate whether the file still exists.

Do commands can also be used to probe for the existence of a file. If a user tries to delete a non-existent file, the system will respond with an error because the file does not exist.

Suppose an attacker deletes a file and the system responds with "the file was deleted". If the attacker later uses the directory listing program to verify that the file is not displayed in the directory listing, he has no way of knowing whether the file was actually deleted and both the system response and the verify response were true, or if the system response was false and the listing of the file is currently being hidden. Responses can also be randomized so that the responses to various requests are inconsistent. If we desire to lead the attacker in a specific direction, a weighing function can be used to increase the frequency of some responses, for example to lead the attacker to believe that the less frequent responses are probably erroneous—but leaving doubt in the attacker's mind.

Human nature suggests that attackers will trust sources with structured and redundant information more than single item information sources. Implementing consistent deception for highly structured information sources is more difficult, such as for a raw device. If the attacker knows how to decipher the raw format of a file system, and notices in the file allocation table that the file claimed to be deleted earlier still exists, he would have some (high) degree of assurance the file was not deleted. It is easier to reply falsely to a deletion request than to forge the raw contents of a file system. The file system has redundancy and a complex semantic structure. It is difficult to capture this structure in a false, yet consistent, way for all requests. So an attacker can conclude that the information obtained from reading the raw file system is more trustworthy than a result returned by a system call.

Table 3 represents an exploit for privilege escalation. Suppose the requested action is a buffer overflow exploit. *Performed action* indicates whether the exploit was successful. If the exploit is successful, the system responds with no error, while some error is returned if the exploit is not successful. Various *tell* commands can be used to verify whether the attacker obtained the desired privileges.

Table 3. Deception options for privilege escalation.

	Escalated privileges	System response	Response truthfulness	Verify response	Verify truthfulness	Consistent
1	No	No error	False	Not admin	True	No
2	No	No error	False	Admin	False	Yes
3	No	Error	True	Not admin	True	Yes
4	No	Error	True	Admin	False	No
5	Yes	Error	False	Not admin	False	Yes
6	Yes	Error	False	Admin	True	No
7	Yes	No error	True	Not admin	False	No
8	Yes	No error	True	Admin	True	Yes

In some cases there are many paths to a piece of information, and no perfect method exists for attaining the confidence that all paths are covered. For example, consider the Linux system, and how an attacker might determine the current

working directory of her process, which is stored in the kernel. Linux provides at least three different mechanisms:

1. Read the kernel memory directly and parse out the location for the current working directory. This requires the attacker to read /dev/kmem using the sys_read() system call, and translate the stored information into the directory path name.
2. Run the *pwd* command, which uses the sys_getcwd() system call to access the information. That system call in turn uses the kernel function d_path() to convert the current working directory's internal identifier into a string (directory name).
3. The Linux system keeps some process information in a special file system "/proc". In the subdirectory corresponding to the current process is a file called "cwd". This is a (symbolic) link to the current working directory. So, use "ls" to obtain the target of the link called "cwd". The "ls" command uses the system call sys_getdents() to obtain the contents of the "/proc" file system. As "/proc" is a virtual file system, its interface uses the kernel function d_path() as above to obtain the directory name that "cwd" links to. Note that the underlying mechanism, although appearing to be a conventional symbolic link, does not implement actual symbolic links.

See Figure 1.

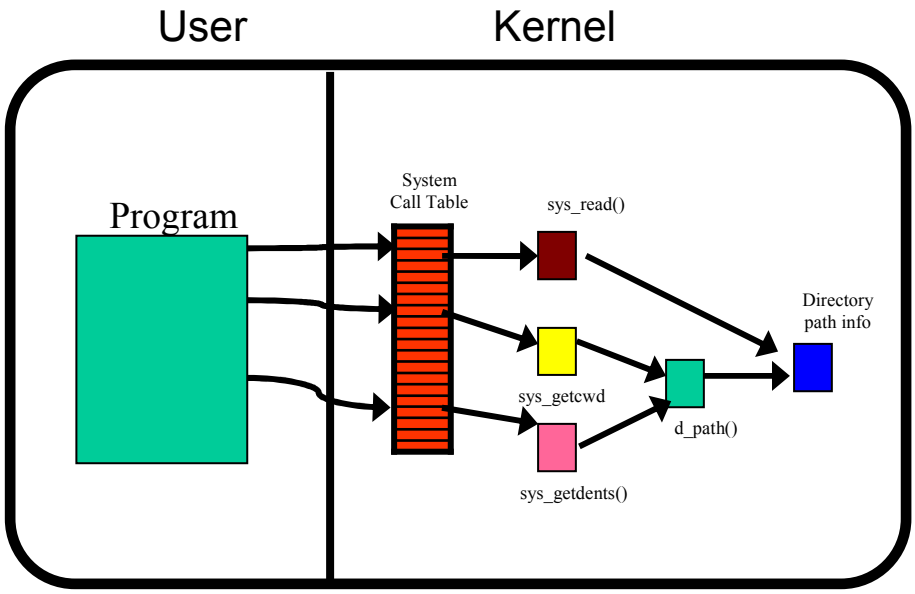

Figure 1. Multiple paths to directory information

Developing a credible, consistent deceptive mechanism under Linux that provides access to a fiction of the kernel memory is perhaps the most difficult aspect of deception. Implementing such a mechanism requires keeping track of the location of data in the kernel memory, as well as the deceptive responses. When a memory location is accessed through a memory reference or through the file corresponding to the kernel memory, "/dev/kmem", the buffer containing the real information needs to be overwritten with deceptive information before being passed to the user. If the result from kernel memory is different that the other two sources, an attacker would find the kernel memory more credible for the reasons discussed above.

While not all paths need to be intercepted for implementing inconsistency, the paths that are not intercepted will always give accurate information. The attacker may be able to hone in on the sources that are always accurate and consistent within themselves. On the other hand, the attacker does not know whether that particular path is consistent because it is not intercepted, or because the deception tactic on that path is designed to give consistent and predictable answers.

Inconsistency can be used to confuse the attacker and may be the best tool for misdirecting the attacker to focus his efforts in other areas. When exposed to inconsistencies, attackers may become distracted from their original goal by trying to reconcile the different system responses. Even when people are presented with apparent inconsistencies and contradictions in honest non-deceptive systems, figuring out why a system is acting inconsistently is frustrating. Attackers may not even suspect deception when inconsistencies are presented since encountering apparent contradictions is a part of everyday interaction with a computer.

If the attacker recognizes that inconsistent deception is being used against him, he might attempt to compensate by gaining as much information from various independent sources (various paths and system calls). Then he must decide which of those sources has a better probability of being reliable based on how difficult it would be to implement a semantically meaningful deception for a certain source. For example, it would be more difficult to create meaningful semantics for a raw disk device than for a system call returning one number.

Prohibited requests will never be performed, so this complies with the principle of least privileges. Though the replies will be inconsistent, the attacker can be sure that such a request is not performed, unless the attacker succeeds.

We assume hackers are entirely logical and rational entities. While this may not be the case, it is a safer assumption and will allow for a more robust model.

Because of the nature of deception, the capacity to determine the presence of deception is eliminated. If an attacker detects inconsistency, he doesn't know if he is exposed to a deception that is simulating plausible malfunctions, or to a normal system that is truly malfunctioning.

For example, when a user deletes a file on a UNIX system, only the inode is actually deleted. If this inode is a symbolic link naming the file, then the file still exists. Because of insufficient information in a user's mental model, such system behavior can appear inconsistent even though no deception is at work. The Windows file system generates a temporal inconsistency between the cache and the disk. There is a time delay from when a file is deleted to when it is actually deleted from the disk.

For greater homogeneity between normal systems and systems that implement inconsistent deception, normal systems should ideally portray some degree of inconsistency also. Fortunatelly, normal systems already have some inconsistencies due to imperfect design and implementation, so no additional inconsistencies need to be added to real systems.

Furthermore, inconsistencies can be *accidental* or *deliberate*. *Deliberate* inconsistency can be used to divert attention. Defenders can use deliberate inconsistencies to manipulate the attacker's beliefs and control where their attention is focused. Inconsistency induces delays in decision making as people evaluate and assess the information. Inconsistency also distracts because humans try to resolve inconsistency, and this wastes the attacker's time and energy. As an example, a mechanism that selected randomly from 15 false error messages is a mechanism for deliberate inconsistency.

Accidental inconsistencies occur due to factors outside the defender's control. For example, if the defender intends to present a consistent fiction, but the mechanisms designed to do so are incomplete or fail, then the attacker may notice the inconsistency and act in a manner that the defender does not expect. This type of inconsistency typically is a problem, because the deception is designed to drive the attacker towards some goal—and the inconsistency will distract the attacker while she tries to reconcile the conflicting information. In the example of determining the current working directory, if the first two methods returned a (consistent) false result, but the third (erroneously) returned the correct result, then the inconsistency is accidental.

Finally, inconsistencies may be *semantic* or *data* inconsistencies. If deceptive results are to meet a goal of tricking the defender into taking some particular action, the deceptions must be consistent enough to convince the attacker of the truth of the fiction that the defenders project. This means that if the attacker expects results or data with specific semantics, the deception must provide it. This is a form of consistency, and as noted can be very difficult to achieve. Inconsistent deception suffers from no such difficulty. If the semantics can be preserved, then the data itself can be inconsistent; but the semantics themselves can be made inconsistent.

For an example, let's return to our file deletion example. The semantics of the deletion command may allow one of three error messages: "file not found", "not enough privilege", or "file in use". Suppose the user tried to delete a file and received the error message "not enough Xenix semaphores". This message is semantically meaningless because it is inconsistent not with the data but with the semantics of the system itself. Even if every attempt to delete the file causes the same error message, the data (that there is an error) is consistent. But the particular result ("not enough Xenix semaphores") is inconsistent with the semantics of the delete command. Hence we have *semantic inconsistency*.

The effect of semantic inconsistencies may lead the attacker to think her understanding of the semantics of the system is erroneous. If she looks at the documentation for the system, the most probable reaction is that the documentation is out of date or itself inconsistent with the actual semantics of the system. This leads to more confusion on the part of the attacker.

5. IMPLEMENTATION ISSUES

Consider the example of determining the current working directory. If a single method of obtaining that information returns the same result for any given directory, but the three methods each return different results for the same directory, the system is *vertically inconsistent*. If one method returns different results for any given directory, the system is *horizontally inconsistent*.

Vertical inconsistency is useful when the defender wants each path to some information to return predictable results, but these results to differ when compared with other paths. In this case, the attacker will likely conclude that a deception is under way. An open question is whether the attacker can determine the reality based on an analysis of the sources—and whether the defender could reason similarly and spend her efforts making the most credible sources deceptive.

Horizontal inconsistency is useful when the defender wants to confuse the attacker, and make her think that either a very poor deception is under way, or that the system is damaged and errors are inhibiting its correct operation. It is difficult to see what purpose other than frustrating the attacker this method serves; but frustration is a valid defensive goal, and if the attacker can be driven away, so much the better.

This leads to the question of where to place the deceptive mechanism if one wants to ensure inconsistency. The answer depends in part upon the goals of the deception and in part upon the architecture of the system.

First a general observation. One achieves the best control over the deception by placing the mechanisms as close to the resource as possible, ideally in the reference monitor controlling access to the resource (if such a reference monitor exists). For example, if the resource is a computer system serving web pages, the deceptive mechanisms should be placed in the servers guarding access to the system. For the current working directory name in our previous example, the deceptive mechanisms would be placed in the reference monitor controlling access to those bytes of memory.

The architecture of the system determines whether a reference monitor exists for the desired resource. If so, then

placing the deceptive mechanism in the reference monitor allows one to ensure vertical inconsistency (or consistency), simply by returning different (or identical) results every time the resource is accessed. If no such reference monitor exists, then there may be multiple paths that access the resource directly. In this case, ensuring vertical inconsistency (consistency) requires that the mechanism co-ordinate responses from different paths, to ensure the inconsistency (consistency) of the results. This may require complicated mechanisms. It may also cause some paths to be left alone, because monitoring them may be too difficult.

Returning to our current working directory example for a Linux system, the deceptive mechanisms must be placed in the kernel, and ideally would detect any attempt to access that location of memory. But the reality is not ideal, and without a complex, slow mechanism the kernel could not detect the first method (reading the raw kernel memory file). An astute attacker could use this to uncover the real current working directory.

A last issue is that of diminishing returns. Adding a consistent deception mechanism to an existing system requires analyzing the system and determining where to place the mechanisms, designing an infrastructure to let the different mechanisms communicate, and studying the inferences that an attacker might make to ensure that the mechanisms force the attacker to draw consistent inferences. As the mechanisms are refined to produce deceptions that inhibit inconsistent inferences, and eliminate both vertical and horizontal inconsistency, they become more expensive to implement and use. By way of contrast, inconsistent mechanisms are simpler, cheaper, and as discussed above may be equally effective in handling attackers.

6. FUTURE DIRECTION

Developing a comprehensive deception model for classifying the various options available for creating deceptive ploys would provide defenders with a variety of ploys to achieve a particular end. For this to work, the effects of each type of ploy need to be determined. There will, of course, be variations based on the attackers' knowledge and personalities, but researchers should be able to establish some general guidelines. We plan to conduct psychological experiments with human subjects to determine the effectiveness of various deceptive tactics.

Our current efforts focus on human users. Future efforts testing inconsistency on automated attacks will be useful.

Whether attackers tend to go with their initial decision and pay less attention to subsequent inconsistent inputs is to be discovered through such experiments. Studies on whether information theory can be used for measuring the degree of inconsistency and determine the benefits of such a measure will also be useful.

7. CONCLUSION

The object of this exercise is to explore the nature of deception, and to argue that inconsistent deception merits attention. Achieving consistency of deception can be difficult if not infeasible in many realistic situations. This raises an obvious question: why is consistency important? In some cases, the answer is obvious; for example, Cliff Stoll's attempts to lure the attacker into staying connected to the system for hours while his phone call was traced required that the attacker not realize he was being deceived. But in other cases, where the goal is to dissuade the attacker from probing further, or to confuse the attacker, consistency is not necessary. We do not propose the substitution of consistency with inconsistency. However, we suggest that inconsistency has been overlooked as a viable option.

In *Rules for Radicals*, Saul Alinsky wrote that a fundamental rule of power tactics is "whenever possible, go outside the experience of the enemy" ([1], p. 127). Alinsky was writing about politics, in which consistency is prized but often lacking. Even in that environment, inconsistency causes problems. Alinsky used Sherman's march to the sea as an example of something inconsistent with prior military tactics—and devastating in its results.

As with any scientific contribution, the methods developed can be used for good or evil depending on whose hands they are in. Inconsistency and deception in general will most likely contribute to the 'arms race' of computer security. For example, rootkit creators can glean the findings of such research and improve their malicious software. While consistent deception will give rootkit creators better tools for hiding their malicious code, inconsistent tactics will not be attractive to them because the inconsistency should attract the attention of administrators who will investigate the inconsistency. On the other hand, deception will give attackers less assurance about success when installing a rootkit.

With computers, consistency is not prized; it is expected. Computers are consistent because they are deterministic. Given the same circumstances, one action will produce one result. When this fails, and the same action is expected to produce the same result but does not, an attacker will wonder what is going on. The result is "confusion, fear, and retreat" ([1], p. 127).

This paper presented some ideas on inconsistency, and a model to demonstrate that it is a natural aspect of deception. We discussed some implementation issues. Inconsistency is feasible, indeed easier to implement than the consistency other research in deception strives for. It is a natural part of existence, yet rare in the world of computing, where one expects predictability unless a system is malfunctioning.

We would do well to consider adopting Alinsky's tactic.

Acknowledgement: This work was supported by award CCR-0311723 from the National Science Foundation to the University of California, Davis.

8. REFERENCES

[1] Alinsky, S. *Rules for Radicals*, Vintage Books, New York, NY (1971).

[2] Carofiglio, V., de Rosis, F., and Castelfranchi, C. "Ascribing and Weighting Beliefs in Deceptive Information Exchanges," *User Modeling*, Springer-Verlag, Berlin pp. 222-224 (2001).

[3] Castelfranchi, C. "Artificial Liars: Why Computers Will (Necessarily) Deceive Us and Each Other," *Ethics and Information Technology* **2** (2) pp. 113–119 (2000).

[4] Castelfranchi, C., and Poggi, I. "Lying as Pretending to Give Information," *Pretending to Communicate*, H. Parret (ed), Springer Verlag, Berlin (1993).

[5] Cheswick, W. "An Evening with Berferd, in Which a Cracker is Lured, Endured, and Studied," *Proceedings of the Winter 1992 USENIX Conference* pp. 163–173, (Jan. 1992).

[6] Cohen, F., Marin, I., Sappington, J., Stewart, C., and Thomas, E. "Red Teaming Experiments with Deception Technologies" (2001); available at http://all.net/journal/deception/experiments/experiments.html

[7] Danaher, J. "The Laws of Thought," *The Philosopher* **92**(1) (Spring 2004).

[8] de Rosis, F., Castelfranchi, C., Carofiglio, V., and Grassano, G. "Can Computers Deliberately Deceive?" *Computational Intelligence* **19** (3) p. 235 (Aug. 2003).

[9] Falcone, R., and Castelfranchi, C. "The Socio-Cognitive Dynamics of Trust: Does Trust Create Trust?" *Trust in Cyber-societies*, Springer-Verlag, Berlin pp. 55–72 (2001).

[10] Gotesky, "The Uses of Inconsistency," *Philosophy and Phenomenological Research* **28** (4) pp. 471–500 (June 1968).

[11] Hernacki, B., Bennett, J., and Lofgran, T. "Symantec Deception Server: Experience with a Commercial Deception System," *Proceedings of the Seventh International Symposium in Resent Advances in Intrusion Detection* pp. 188–202 (Sep. 2004).

[12] Michael, J., Auguston, M., Rowe, N., and Riehle, R. "Software Decoys: Intrusion Detection and Countermeasures," *Proceedings of the 2002 Workshop on Information Assurance* (June 2002).

[13] Michael, J., Rowe, N., Auguston, M., Drusinsky, D., Rothstein, H., and Wingfield, T. *Phase II—Report on Intelligent Software Decoys: Counterintelligence and Security Countermeasures*, Technical Report, Naval Postgraduate School, Monterey, CA (Apr. 2004).

[14] Michael, J. "On the Response Policy of Software Decoys: Conducting Software-based Deception in the Cyber Battlespace," *Proceedings of the 26th Annual International Computer Software and Applications Conference* pp. 957–962 (Aug. 2002).

[15] Montagu, E. *The Man Who Never Was*, J. B. Lippincott Company, Philadelphia, PA (1954).

[16] Plato, *Sophist* 237A.

[17] Priest G., and Tanaka, K. "Paraconsistent Logic," *The Stanford Encyclopedia of Philosophy*, E. Zalta, ed. (Winter 2004); available at http://plato.stanford.edu/archives/win2004/entries/logic-paraconsistent/.

[18] Rogers, D. *Host-Level Deception as a Defense Against Intruders*, Master's Thesis., Dept. of Computer Science, University of California at Davis, Davis, CA (June 2004).

[19] Rowe, N. "Counterplanning Deceptions to Foil Cyber-Attack Plans," *IEEE Workshop on Information Assurance* pp. 221–228 (June 2003).

[20] Rowe, N., Michael, J, Auguston, M., and Riehle, R. "Software Decoys for Software Counterintelligence," *IA Newsletter* **5** (1) pp. 10–12 (Spring 2002).

[21] Stoll, C. "Stalking the Wily Hacker*," Communications of the ACM* **31** (5) pp. 484–497 (May 1988).

[22] Tognazzini, "Principles, Techniques, and Ethics of Stage Magic and Their Application to Human Interface Design," *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* pp. 355–362 (1993).