A Short Course on Software-Based Deception and Counter Deception

Session SC5,

"Information Security Strategy and Counter Deception"

IEEE WESCON 2003

San Francisco, California

August 13, 2003

Instructor

J. Bret Michael, Ph.D.
Associate Professor of Computer Science
Naval Postgraduate School
833 Dyer Road
Monterey, CA 93943-5118
Tel. +1 831 656-2655
bmichael@nps.navy.mil
http://www.cs.nps.navy.mil/people/faculty/bmichael/

Disclaimer

- The views and conclusions contained herein are those of the instructor and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the U.S. Government.

Course Outline

- Learning objectives
- Introduction to deception
- Motivation for applying deception in cyber space – A shift in paradigms
- Intelligent software decoys
- Overview of how to build intelligent software decoys using wrapper technology
- Concluding remarks

Learning Objectives

- At the conclusion of this short course, the participants will be able to
  - Articulate the need for a shift in computing paradigms to provide for the use of deception to responding to intrusions and malicious behavior
  - Describe the key properties of an intelligent software decoy
  - Provide examples of how decoys can be used to protect information systems
  - Specify a simple deception model using the Chameleon language

This short course will not…

- Make you an expert on the topic of software-based deception
  - There isn't enough time
    - Complex technically challenging subject
  - The issues are accumulating faster than we can keep up with them
    - New technological innovations
    - New applications of existing technologies

Some Reflections on the Subject of Deception

All warfare is based on deception.
- Sun Tzu

We are never deceived; we deceive ourselves.
- Goethe

Illusion is the first of all pleasures.
- Voltaire

One is easily fooled by that which one loves.
- Moliere

The mind is the greatest weapon.
- Rambo

What is deception?

- Deception is a conscious and rational effort to deliberately mislead an opponent. It seeks to create in an adversary a state of mind which will be conducive to exploitation by the deceiver.
- Counter deception is the act by the targeted party to try to mislead the deceiver.

Why and How

- Why deceive?
  - Freedom of action for deceiver
  - Disadvantageous action by opponent
  - To gain surprise by deceiver
  - To save lives
- How?
  - Increase ambiguity
  - Mislead by reducing ambiguity

Principles of Deception

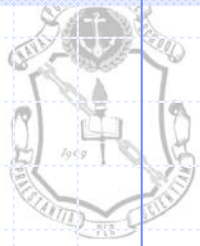
- Aimed at the mind of the opponent
- Aim is to make the opponent act
- Coordination and centralized control
- Preparation and timing
- Security
- Credibility and confirmation
- Flexibility

Principles for Effective Deception

(Fowler & Nesbit, *J. of Electronic Defense*, June 1995)

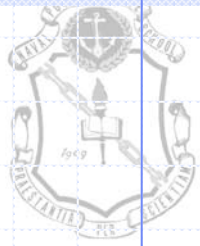
1. Deception causes the enemy to believe what they want to believe.
2. Deception involves timely feedback.
3. Deception is integrated with operations.
4. Deception conceals the critical true activities.
5. Deception is tailored to the task.
6. Deception should be imaginative and should not become stereotyped.

Deception in the Physical versus the Cyber Realm



- One cannot violate the laws of physics
  - Difficult to create believable deceptions in the physical world
- One can readily create deceptions in the cyber world, partly due to the fact that the internals of the computing system are often viewed as a black box

Need to Protect Components of Distributed Systems



- Software from which distributed systems are composed needs to be protected from malicious use
  - For example, recent "Code Red" worm
- Some of the features of distributed systems make them tempting targets
  - For instance, dynamic patching schemes (e.g., updating software on communications satellites)
- Software components with poorly designed interfaces are susceptible to misuse or modification by rogue programs
  - For example, the UNIX *fingerd* program and IIS Indexing Service DLL of Windows have been found to be susceptible to well-known (for many years!) buffer-overflow attacks

Legal and Societal Issues

- In addition to the technical feasibility of realizing software-based deception mechanisms, we need to explore the legal and societal issues associated with applying these mechanism
- Tom Wingfield's *Law of Information Conflict* (Falls Church, Va.: Aegis Research Corp., 2000), provides a good overview of the legal issues of conducting cyber warfare

View of Deception in Society

- Legal and cultural predisposition in U.S. against
  - Institutionalizing deception
  - Using non-defense sectors of government
  - Using culturally-respected portions of private sector (journalists, clergy, academia, NGOs) for deception
- Sissela Bok's *Lying: Moral Choice in Public and Private Life* (New York: Vintage Books, Second ed., 1999) gives perspectives on the morality of applying deception

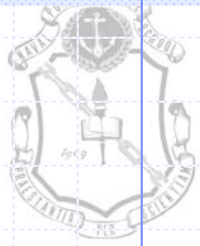
Combating Cyber Terrorism

- Challenge:
  - To use deception to combat cyber terrorism and protect cybernetic property (e.g., the public-switched telephone network), without having the use of deception fall victim to negative public sentiment (e.g., from misuse)
- Solution:
  - Perform principled analyses to proactively assess the
    - Legality of using software-based deception
    - Social implications of applying deception in terms of the level of intrusiveness and impact on civil liberties

Prevalent Approach Today to Intrusions: "Cyber Karate"

- "If you try to kill me, I will try to kill you" philosophy - Mas Oyama, famous Karate Master
  - Software systems and components usually terminate interaction when suspicious behavior is detected
    - Indicates to the attacker that the attack has been noticed
      - Problem: Difficult for the defensive system to learn about the nature of an attack (for use in improving defenses)
    - Can result in denial of service to legitimate users
    - Is not effective against sophisticated attackers who will adjust their strategy and tactics to either deceive or bypass the intrusion detection system
- Non-real-time analysis of behavior patterns from audit trails
  - Relatively large delay before compromise of system or component can be detected

A New Way to Think about Protecting Distributed Resources



- Think in terms of protecting software components, and the object and methods that reside in components, using
  - Deception techniques built into components
    - Use of decoying actions to learn about the nature of attacks and influence the behavior of the attacker
  - Strong interfaces to components
    - Survivability while tolerating inappropriate interaction by a rogue program with a component
  - Automatic instrumentation of software components
    - Runtime monitoring of behavior patterns over event traces
    - Formalization of knowledge of typical intrusion patterns and decoy strategies

A New Approach to Protecting Software Systems

- Use of decoying actions to learn about the nature of attacks and influence the behavior of the attacker
  - Adapt traditional military deception for use in software-based deception
- Explicit provision for survival and adjustment of system behavior to attack
  - Provide for intrusion tolerance within operating systems, middleware, and applications
- Automatic and selective instrumentation of software systems for intrusion detection and decoying actions
  - Provide for efficient (selective) and effective (changes in nature or understanding of the attack) runtime adaptation of detection and decoying actions (runtime extensibility)
- Explicit computer-based support for coordination among software-based deception mechanisms
  - Coordinate actions across subsystems to maintain deceptions and improve, via feedback between the subsystems, the effectiveness of the intrusion detection mechanisms

Consider Using "Cyber Akido"

- Cyber Akido
  - Decoy-enabled software component tolerates an attack and learns about the opponent's (i.e., rogue program's) weaknesses, sources, and methods
  - Decoy tries to neutralize its opponent by taking the following steps:
    - First, try to reduce or **eliminate** the **will** of the attacker (e.g., insert delays into responses to method calls)
    - Next, **change proximity** to opponent (e.g., direct attacker to a honeypot)
    - Lastly, reduce or **eliminate** the **ability** of the opponent to attack (e.g., terminate calling process, launch counter denial-of-service attack)
  - Here, the goal is not to reveal the purpose of the deceptive actions to the targeted party (i.e., intruder/malicious user)

Intelligent Software Decoy

- An abstraction for protecting objects from malicious attacks by calling processes
  - We assume the attacker (i.e., rogue agent) will try to change the behavior of the targeted object
- Intended to deceive an agent into believing that
  - The decoy is the object it advertises itself to be
  - All of the decoy's responses are legitimate
- Discovers and reveals the presence of an attacker
- Learns about both the usage of known attacks, and the existence and details about previously unknown types of attacks
- Neutralizes the effects of an intrusion
- Initiates countermeasures (i.e., responds to attacks) based on information about the nature (and possibly the source) of an attack

Examples of Types of Responses

	Low runtime resources	***Medium runtime resources***	***High runtime resources***
Low setup resources	Fake error messages	Scripted response to known attack	Operating system "sandbox"
Medium setup resources	Exaggerate processing times	Fake debugging tools	Dynamically simulate new viruses
High setup resources	Fake directories	Fake a known-virus infection	Dynamically counterplan on attack

Decoys as an Airlock between Technology and the Law

- Can be programmed with a wide spectrum of options for taking action
- Provide for anticipatory exception handling
  - One could develop policy that places boundaries on the extent and type of deception to be employed, but provide 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

All Objects Can Serve as Decoys

- Decoy mode is triggered when the target object receives a message that violates the object-agent contract in one or more ways
- Decoy behavior is specified in an ante chamber
  - When the precondition fails, the object's interaction with the calling process is controlled internally via decoying action language

Objectives

- Provide a generic framework for intrusion detection and countermeasures based on precise models of system behavior
- Explore decoy strategies that alleviate the lack of variability in previous work, and provides for a broad spectrum of responses to attacks
- Develop an architecture that supports formalization of knowledge of intrusion patterns and decoy strategies
- Provide for a significant level of automation of decoy activities, making much of the low-level details of instrumentation of systems for detection and response transparent to the user of the decoy technology

Properties of Intelligent Software Decoys

- Intelligent
  - Adapt their behavior to changes in their operating environment
- Autarkic
  - Do not rely on the internal state of other objects to protect themselves
  - However, for certain types of interactions (e.g., chained calls between components), decoys can cooperate using global knowledge
- Polymorphic (chameleon-like character)
  - Disguise themselves by altering their object-interface contracts at run-time

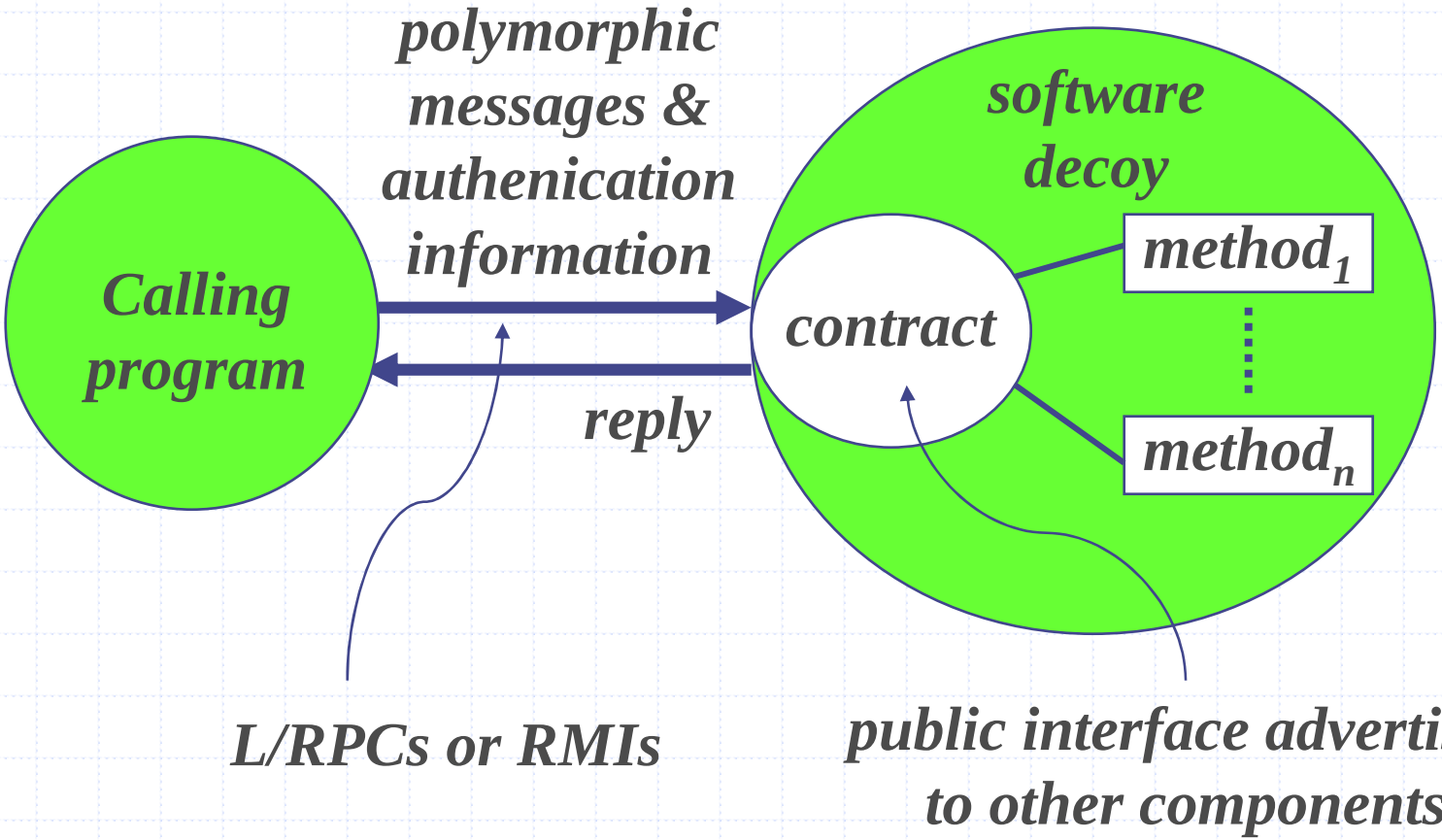
Properties of Intelligent Software Decoys

- Policy-governed
  - Behavior governed by pre- and postconditions, in addition to class invariants
    - Deception policy determines decoying actions to apply
- Self-replicating
  - Replicate themselves, either in an autonomous or cooperative manner

Primary Differences between Decoys and Honey pots/nets

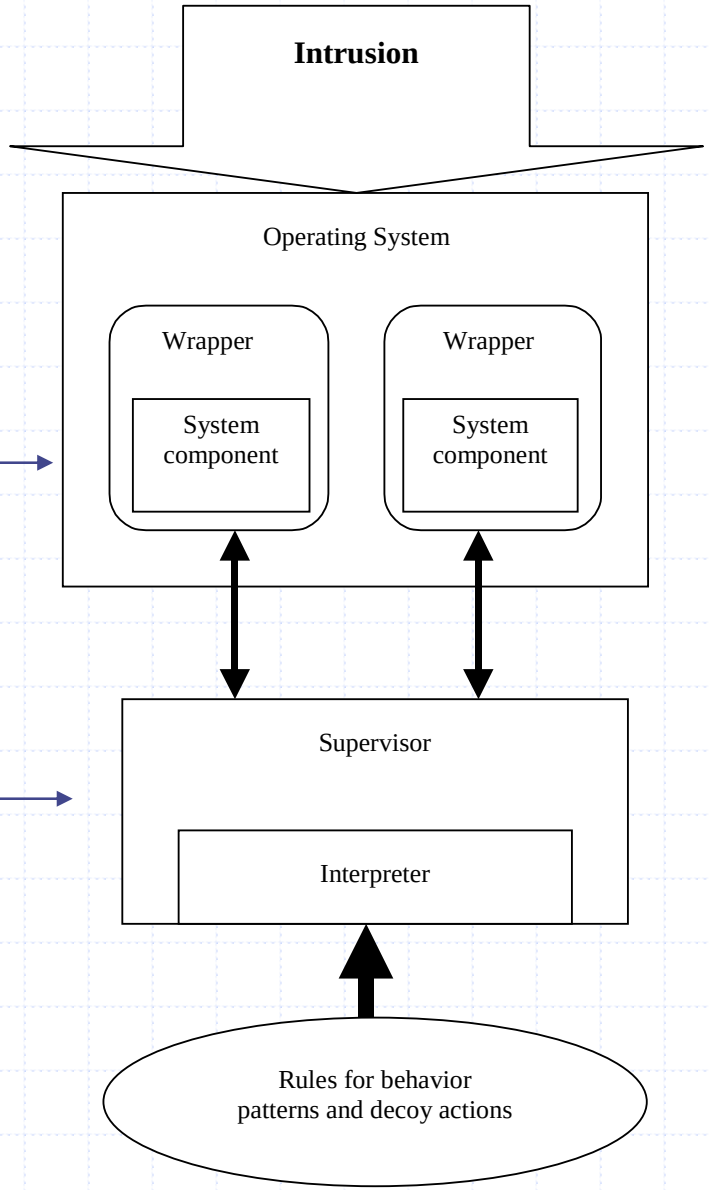
- In contrast to honey pots and honey nets, intelligent software decoys
  - Are part of the operational system, rather than a separate system
    - Decoys must meet or exceed the performance criteria specified for the software-based systems they protect
  - Learn about the nature of the attack by encouraging continued interaction with the attacker
  - Can be used in a defensive or offensive manner, rather than just for analysis purposes

Interaction between Calling Program and Software Decoy



Local-level wrapping of system and application calls with detection and decoy capability

Supervisor performs instrumentation, and coordinates the actions of software decoys, other supervisors, and global-level detection-and-response tools

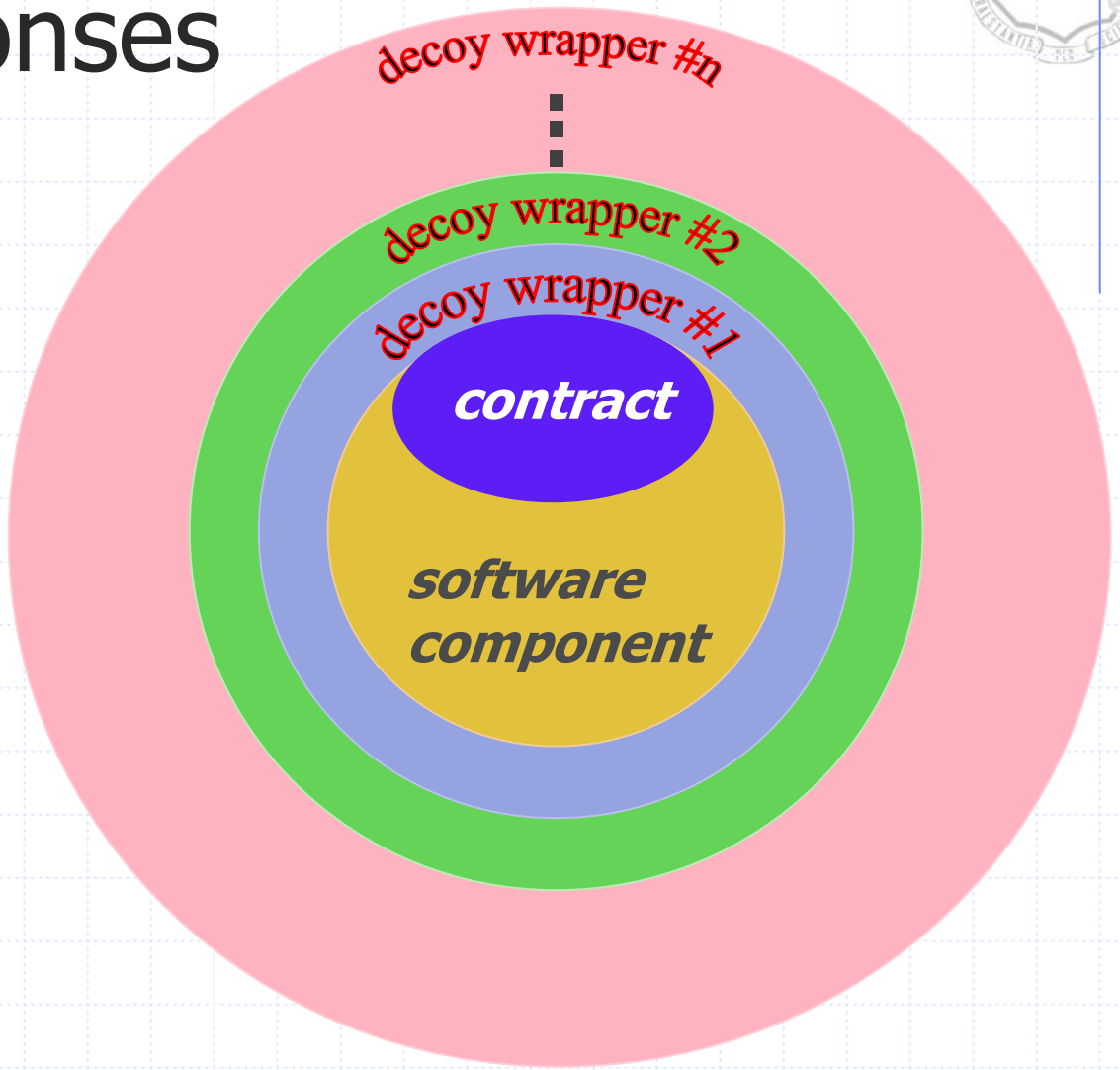


Intrusion/Misuse Detection and Decoy Responses

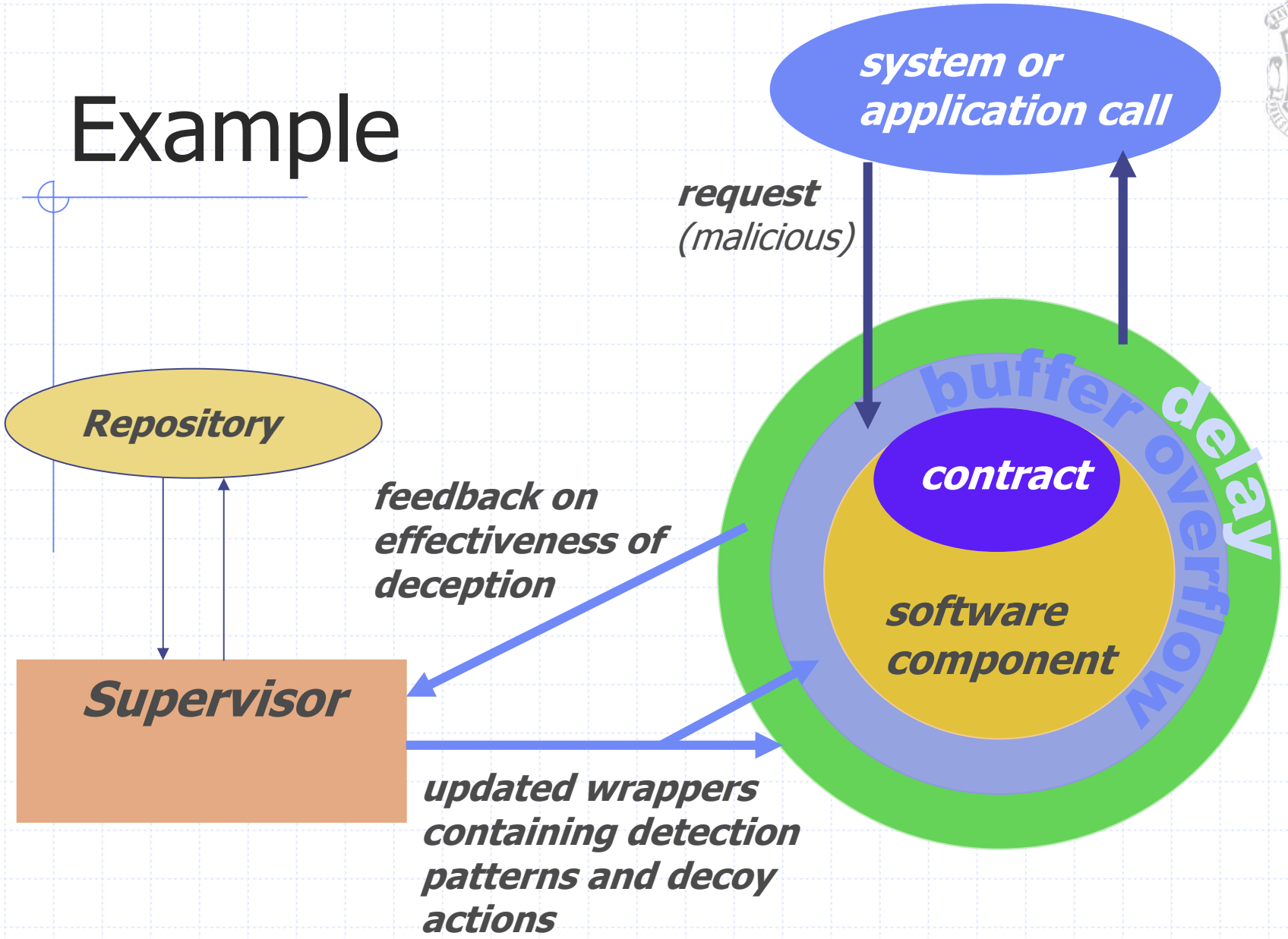
Software components are wrapped with decoy functionality on a selective basis

Wrapping can be performed at more than one level of abstraction. from application-level objects such as web applets to low-level operating system calls

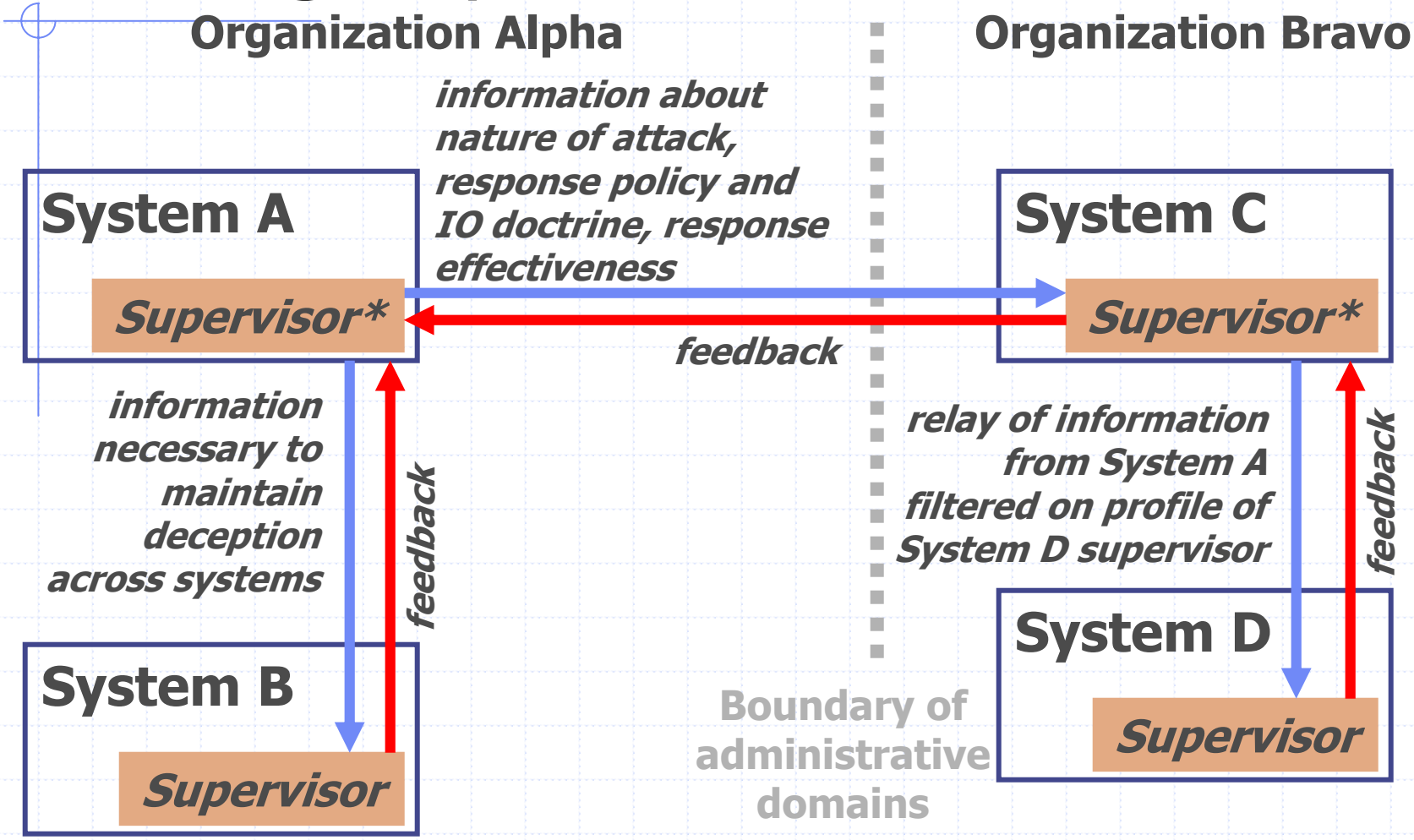
Software contract is the public interface of the component



Example


system or application call
request
(malicious)
Repository
feedback on effectiveness of deception
Supervisor
updated wrappers containing detection patterns and decoy actions
buffer overflow
delay
contract
software component

Coordination and Interoperability Among Supervisors



Software-based Deception is not New

- Misrepresentation of an agent's true goal, for purposes of negotiating with other software agents
  - Reasoning with incomplete information
  - Byzantine Generals problem
- Reasoning about the intelligent behavior of software-based systems
  - Turing's "imitation game" (a.k.a., Turing test)
  - Self-deceiving software-based systems
- Software-based tools for constructing and maintaining deceptions in virtual worlds
  - for instance, Cohen's Deception Toolkit (DTK)

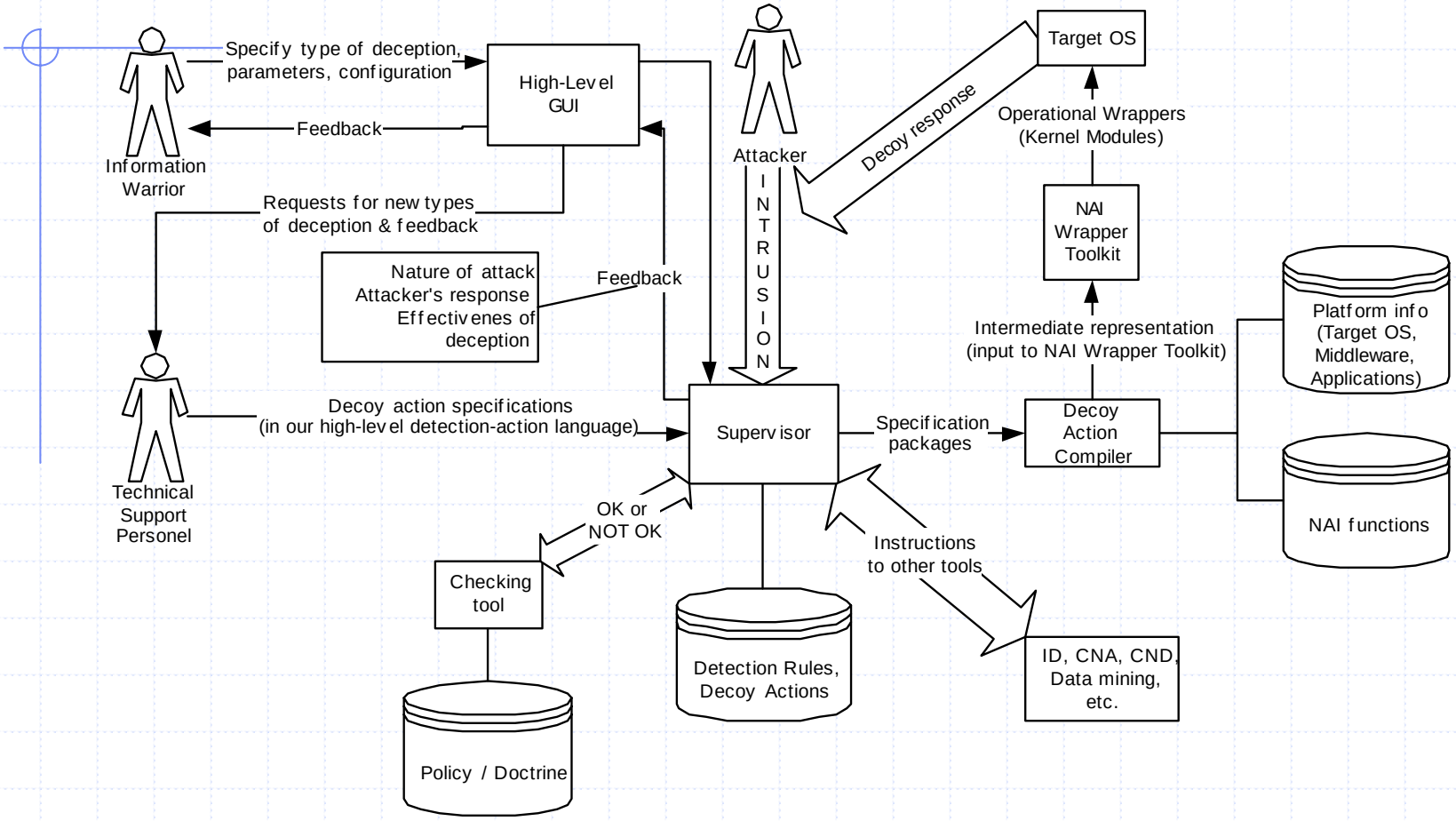
Software-based Deception is not New

- Information-theoretic techniques for detecting evidence of deception
  - for example, authentication-coding schemes
- Use of the "art of illusion" in the design of human-computer interfaces
  - Managing the virtual reality that the user of the interface perceives
- Inserting delays into operating-system responses
  - Somayaji's pH system provides the operating system with time to evaluate the nature of an pattern of system behavior

Novel Aspects of Our Software Decoys

- Software contracts are used to
  - Specify security policy, and mediate the interaction under policy between the intelligent software decoy and the attacker
- Postconditions and invariants place fail-safe constraints on the behavior of the decoy
  - Contain and observe the attacker, while attempting to prevent the attacker from learning that the attack has been detected
- Class invariant makes it impossible for a rouge program to change the decoy's behavior via the interface
- Decoy can change its appearance via polymorphism

A View of the Deception Process



Intelligent Software-Decoy Architecture

- Software decoys are objects within components
- Connectors between components are named interfaces
  - Name advertised to other components need not be unique
  - Consist of an ordered list of arguments
    - Primitive types or object classes (supporting polymorphic types)
- Each class is composed of its own arguments and behavior
  - Arguments are used to access methods of objects within a component

Intelligent Software-Decoy Architecture

- Modification of the software decoy's interface is supported by polymorphism
  - Change one or more of the
    - Arguments
    - Order of the arguments
    - Data type or class of arguments
    - Number and position of dummy arguments
- Component interaction is based on a contract that is controlled by assertions as well as a polymorphic type

Intelligent Software-Decoy Architecture

- An calling process cannot modify the behavior of the decoy beyond the extent to which such modification is permitted by the parent class of the decoy
- Venus flytrap model
  - If the precondition fails, then the decoy does not thwart the attack, but rather contains the attacker
    - Invariant and postconditions defend the object
    - The decoy deceives the attacker into thinking its attack has not been detected, maintaining the interest of the attacker
    - Observe and try to determine intent and source of attack

Principles of Implementation

- Based on precise system behavior model
- Automatic translation of decoy strategy rules into component selective instrumentation
- Automatic generation of wrappers for system and application components
- Perform computations over event traces in order to both detect patterns of behavior and associate decoy actions with events

Precise Model of Behavior

- Event
  - An abstraction of any detectable action performed at runtime
- Binary relations over events (partial ordering):
  - Precedence and inclusion
- Event attributes
  - Beginning & end of an event, source code associated with event, etc.
- Event trace
  - Representation of system execution as a set of events with the two basic relations between them
- Event grammar (for describing the structure of events)
  - Set of axioms that determines possible configurations of events of different types within the event trace

Event

- Any action that can be detected during program execution
- Can have attributes
- Example of an event with two attributes:

```
event read (buf, nbyte)
```

Event Trace

- Two binary relations are defined for events
  1. Precedence
  2. Inclusion
- These relations suffice to describe a program execution as a partially ordered set of events—an *event trace*

A Lightweight Semantics Specification with Two Parts

- Part 1: Axioms (event grammar rules)
  - Define constraints on the behavior of components

```
execute-assignment::
  (evaluate-right-hand-part
   perform-destination)
```

- Part 2: Patterns of behavior
  - Expressed in terms of event patterns
- Use model to automatically instrument source code for intrusion detection and monitoring activities
  - Recognize patterns and perform computations over event traces
  - Selectively instrument, on a component-by-component basis, the event types and event attributes-of-interest

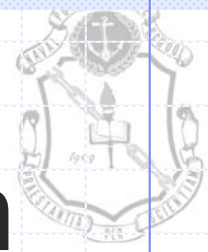
Chameleon Specification Language

- Developed at the Naval Postgraduate School
- Used for specifying axioms and patterns of behavior in the context of software deception

Example of a Specification (Domain Model) for *fingerd_call*

```
fingerd_call
    Attributes:  caller_id
                 begin_time
param_pass
    Attributes:  length
read
```

Example of a Behavioral Specification for the Morris Worm



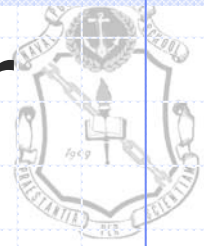
computation over an event trace (separated from the source code)

```
fingerd_call:: ( x: param_pass

& length(x) > max_buffer_size
read +
probe( buffer_overflow )
```

- A *probe* is a Boolean expression evaluated immediately after the preceding event pattern has been matched successfully
  - Used to filter events to create views of the event trace subspace, evaluating the truth of assertions, computing specific values, etc.

Example of Decoying Strategy for Use Against the Morris Worm



```
detect [x: fingerd_call::
( y: param_pass )
& length(y) > max_buffer_size
read + ] && CONST(process_id)
probe (buffer_overflow)
from execute-program
do enable delay (t) to z: utility_call
& process_id(x) = process_id(z)
```

insert process-suspension time (create illusion that a delay has been introduced for system calls in the same process)

Practical for Use with Legacy Systems?

- Yes
  - One can selectively wrap existing software components, especially components that affect the correct behavior and availability of mission-critical systems or the underlying information infrastructure
- Eventually, one would want to rewrite the software components, such as the implementation of *fingerd,* using strong software contracts (in the Meyer sense)
  - This is costly, but one can gradually introduce contracts through the use of contract-wrappers

Practical from a False-Positive View?

- Need to minimize the probability that a decoy will turn a false positive into a situation in which the component denies service to a legitimate user
  - For example, a buffer overflow can be due to either
    - An egregious use of the interface (i.e., non-attack scenario)
    - An attack
  - In signature-based systems, the signatures tend to be general, and there are issues of temporal validity of the signatures
- Decoy provides the component time to assess the nature of the interaction and signature

Potential Weakness and Solution

- Relies on a strong foundation: the distributed operating system (e.g., 2K, StratOSphere, Legion) along with the local network operating systems (e.g., Windows NT)
  - If the operating system is not trusted, then the attacker will circumvent the software decoy, preferring to instead attack the weak infrastructure
- Proposed solution
  - Incorporate the intelligent software decoys into the design of the operating system itself
    - Most modern operating systems are full of security holes that could be partially addressed with the use of software decoys

Examples of Related Work

- Wrapping operating-system calls, such as
  - Sekar & Uppuluri (1999)
  - Ko et al. (2000)
    - WDL
- Instrumenting object code and virtual machines (with hypervisors)
  - Erlingsson & Schneider (1999)
  - Bressoud and Schneider (1996)
- Use of event traces, such as
  - Vigna and Kemmerer (1998)
    - NetSTAT, STATL
- Intrusion tolerance and confinement, such as
  - Liu & Jajodia (2001)
  - Somayaji (2002)
    - pH
  - Sekar et al. (1999)
    - ASL
- Architectures for intrusion-response systems, such as
  - Lewandowski et al. (2001)
    - Survivable Autonomic Response Architecture (SARA)
  - Neumann and Porras (1999)
    - EMERALD

Future Directions

- How would one measure the effectiveness with which software decoys and other security mechanisms guard against attacks on semantic webs?
- Is it technically feasible to coordinate a deception across a distributed system (e.g., a cooperative engagement grid) if the attacker has multiple avenues for both launching attacks and observing the state of the targeted data, information, and software components?
- How does one protect the software decoy or other security mechanism itself from being compromised?
- Can defensive responses by an intelligent software decoy cause unintended side effects on the attacker's computing platform, due to software defects in software that the attacker uses?
- To what extent can we integrate existing intrusion-detection-and-response technology into architectural frameworks for software decoys (to avoid "reinventing the wheel")?

Future Directions

- Is it technically feasible to
  - Identify the true source of an attack?
  - Develop a precision-guided cyber weapon for use in responding to attacks?
  - Determine with a high level of confidence that the use of the cyber weapon will not have effects beyond those that are both intended and permissible (e.g., if one targets the enemy's command and control system, that the loss of the C2 system will not have an adverse affect on the function of civilian information systems that were somehow coupled to the C2 system)?
- To what extent can the details of creating and maintaining deceptions be made transparent to the users of such technology?
- Just because one demonstrates the technological feasibility of a realizing a security mechanism does not mean that a user can legally apply that mechanism, the software decoy being a case in point. Legal experts be involved in the development of such technology early in the process (i.e., at the time of system conceptualization and requirements specification)?
  - In terms of *jus in bello* (i.e., the rules of war), what constitutes a perfidious act on the part of a decoy?

To Learn More…

- "Intelligent Software Decoys." In *Proc. ARL/DARPA/NSF/ONR Workshop on Eng. Automation for Software Intensive System Integration* (Monterey, Calif., June 2001)
- "Intelligent Software Decoys: Intrusion Detection and Countermeasures." In *Proc. IEEE Workshop on Information Assurance* (West Point, N.Y., June 2002)
- "Software Decoys for Software Counterintelligence." To appear the *IA Newsletter* (June 2002)
- "On the Response Policy of Software Decoys: Conducting Software-based Deception in the Cyber Battlespace." In *Proc. Computer Software and Applications Conf*. (Oxford, Eng., Aug. 2002)

To Learn More…

- "Lawful Cyber Decoy Policy." In Gritzalis, D., Capitani di Vimercati, S., Samarati, P., and Katsikas, S., eds. *Security and Privacy in the Age of Uncertainty*: *IFIP TC11 Eighteenth International Conference on Information Security*. Boston: Kluwer Acad. Publishers, 2003.
- "An Experiment in Software Decoy Design." In Gritzalis, D., Capitani di Vimercati, S., Samarati, P., and Katsikas, S., eds. *Security and Privacy in the Age of Uncertainty*: *IFIP TC11 Eighteenth International Conference on Information Security*. Boston: Kluwer Acad. Publishers, 2003.

To Learn More…

- "Experiments with Deceptive Software Responses to Buffer-Overflow Attacks." In Proceedings of the 2003 IEEE Workshop on Information Assurance (West Point, N.Y., June 2003)
- "Measured Responses to Cyber Attacks Using Schmitt Analysis: A Case Study of Attack Scenarios for a Software-Intensive System." In *IEEE Proc. Twenty-seventh Annual Int. Computer Software and Applications Conf.* (Dallas, Tex., Nov. 2003)