2001-07-31

Intelligent Software Decoys: Presented at the Eiffel Summit

Michael, Bret

Intelligent Software Decoys

Presented at the Eiffel Summit

Santa Barbara, California, July 31, 2001

Bret Michael

Department of Computer Science

Naval Postgraduate School

Draft 12/28

Presentation at TOOLS 2001 Eiffel Summit, July 31, 2001

Outline

- What is an intelligent software decoy
- Motivation for exploring the concept of a decoy
- Overview of related work
- Introduction to a software-decoy architecture
- Discussion of language and operating system support for software decoys
- Current work, other research, and funding status

Intelligent Software Decoy

- An abstraction for protecting objects from malicious attacks by mobile agents
  - 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 it is the object it advertises itself to be
- Provides a means for revealing the presence of an attacker

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 agent is controlled internally via a constraint language
    - The constraint language is an extension to the semantics of the Eiffel programming language

Concept Based on the Unified Power of Attack (UPA)

- One should try to neutralize one's opponent by means of, in the following order, reducing or eliminating the opponent's
  - Will to attack
  - Proximity of attack
  - Ability to attack
- Our conceptualization of intelligent software decoys is based on "software *jiu jitsu*"

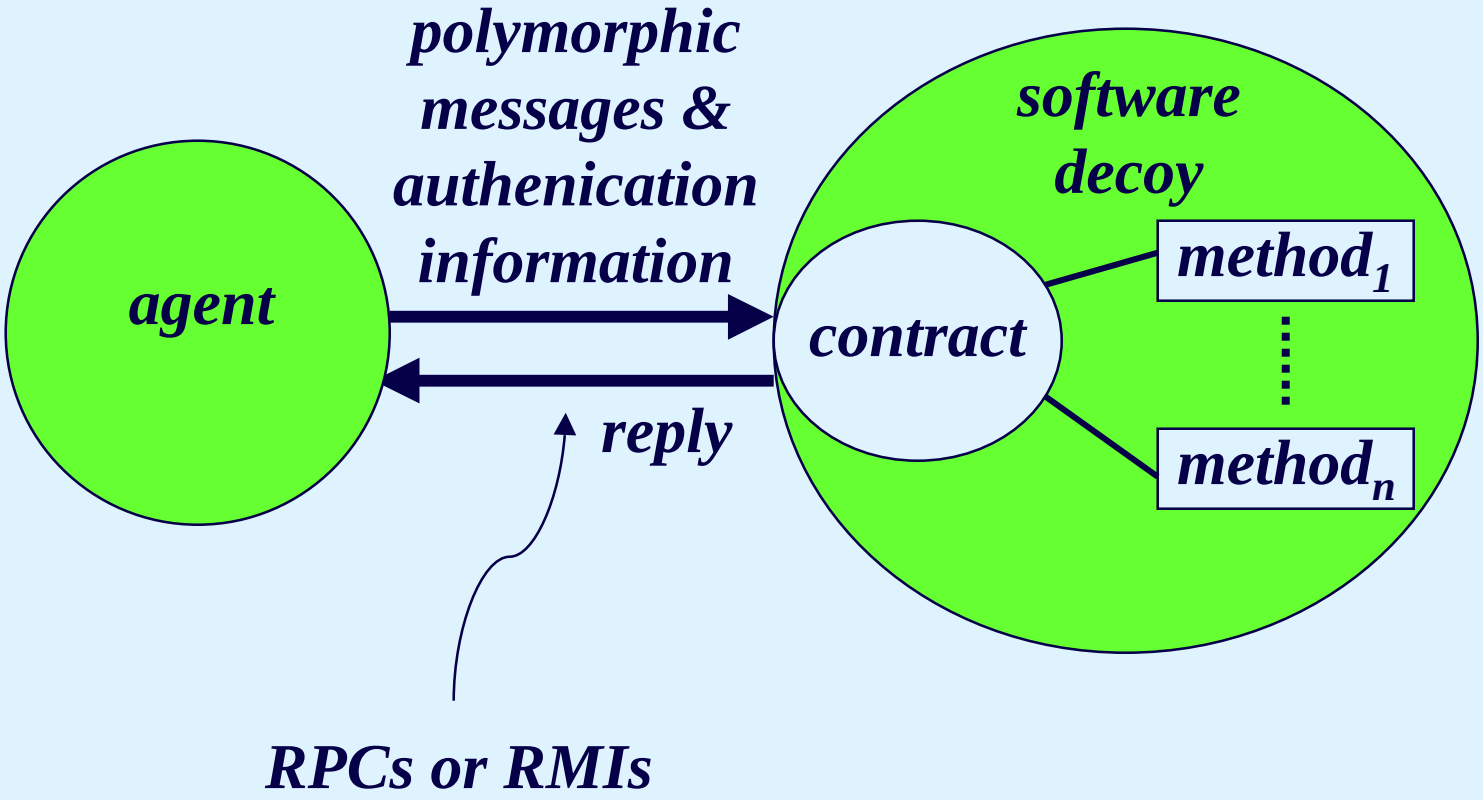
Properties of Intelligent Software Decoys

- **Intelligent**
  - **Adapt their behavior to changes in their operating environment**
- **Autarkic**
  - **Do not rely on the internal state of other objects to protect themselves**
- **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**
- **Self-replicating**
  - **Replicate themselves, either in an autonomous or cooperative manner**

Agent-Decoy Interaction



Motivation

- Software from which distributed systems are composed needs to be protected from malicious attack
  - Recent “Code Red” worm
- Some of the features of distributed systems make them tempting targets
  - for instance, dynamic patching schemes
- Explore the use of Meyer’s design-by-contract model to protect the components of distributed systems

Related Work

- **Concept of 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)**

Related Work

- **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**

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 the agent to change the decoy's behavior via the interface**
- **Decoy can change its appearance via polymorphism**

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 agent 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

Language and Operating System Support for Software Decoys

- Eiffel is a natural choice because it provides explicit support for
  - Design-by-contract
  - Inheritance of the assertions from ancestor classes by a descendant class
    - Needed to preserve the integrity of the software contracts
- Sombrero or StratOSphere distributed operating system are also a natural choices
  - Single address space operating systems support the naming conventions needed for implementing intelligent software decoys

Practical for Use with Legacy Systems?

- **Yes**
  - **One can wrap existing software objects with contracts, especially objects that are critical for the correct behavior and availability of mission-critical systems or the underlying information infrastructure**
- **Eventually, one would want to rewrite the software objects, such as the implementation of the File Transfer Protocol**
  - **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 object denies service to a legitimate user (or agent)
  - 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 object time to assess the nature of interaction and signature

Potential Weakness and Solution

- **Relies on a strong foundation: the distributed operating system (e.g., 2K or Legion) along with the local network operating system (e.g., Windows NT)**
  - **If the operating system is not trusted, then the attacker will circumvent the contract interface of the object, 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 contracts**

Significantly Different from Other Approaches?

- Yes
  - Represents a major shift in security paradigm
    - Integration of formal methods (design-by-contract), deception techniques, and Unified Power of Attack, and actual application of software engineering principles
- Complementary to some other approaches
  - For example, it could be used in conjunction with the Return Address Defender (Chiueh and Hsu, State University of New York at Stony Brook)
    - Addresses the primary weakness of RAD: RAD itself is susceptible to attack via a buffer overflow
      - RAD could be wrapped with the decoy software (i.e., a contract)

Current and Proposed Work

- Developing a formal model of the intelligent software decoy architecture
- Exploring the use of various types of decoys
  - For instance, “volunteer” and “drafted” decoys
- Intend to investigating the technical feasibility of realizing the intelligent software decoy architecture by using the “Code Red” worm
  - The worm exploits the IIS Indexing Service DLL via a buffer-overflow technique

Status of Publications and Presentations

- Published an article on the concept in the Proceeding of the Workshop on Engineering Automation for Software Intensive System Integration (Monterey, Calif., June 2001).
- Several presentations at universities and research institutes within the United States

Acknowledgement of Funding Sources

- All work through July, which started in February 2001, was funded by an internal grant from the Naval Postgraduate School
- This research is currently funded by a grant from the Naval Research Laboratory and other commands within the US Navy

Contact Information

J. Bret Michael
Associate Professor
Department of Computer Science
Naval Postgraduate School
Code CS/Mj
833 Dyer Road
Monterey, CA 93943-5118
tel. (831)656-2655
fax. (831)656-2814
bmichael@cs.nps.navy.mil
http://www.cs.nps.navy.mil/people/faculty/bmichael/