**GIDA : A Game Inspired Defense Architecture**

C. Ellis, V. Datla, and H. Bedi

Department of Computer Science: GTCS Research Lab

University of Minnesota

Advisor: Dr. S. Shiva

---

### Introduction

The weakness of traditional network security solutions is that they lack a game-theoretic perspective. The decision making done by the most complex component: the game agent. At first one might imagine that the features we wish to provide would require a great deal of computational time and considerable effort on the part of the implementers. This however is not entirely true. We can expedite the development of the framework by taking advantage of existing components and integrating them together to provide the features we require. Further, this method of implementation grants the flexibility to modify or even replace certain components as technology advances.

### Game Agent Implementation

We chose to begin our proof-of-concept implementation of the GIDA framework with the most complex component: the game agent. At first one might imagine that the features we wish to provide would require a great deal of computational time and considerable effort on the part of the implementers. This however is not entirely true. We can expedite the development of the framework by taking advantage of existing components and integrating them together to provide the features we require. Further, this method of implementation grants the flexibility to modify or even replace certain components as technology advances.

#### Game Agent Requirements:

- \( \text{Firewall Identification} \)
- \( \text{Game Model Implementation} \)
- \( \text{Agent Control} \)
- \( \text{Transparent Traffic Inspection} \)
- \( \text{Control Messaging} \)

### Game Agent Component Workflow

We propose a semi-autonomous defense architecture which leverages a game theoretic model to counter cyber attacks. GIDA will be capable of transparently observing network traffic, identifying malicious activity, measuring the risk, and acting upon that information in a way that will offer the best defense measure for that situation. The brain of GIDA is a game model which decides the best countermeasure after a thorough analysis of the cost and reward.

GIDA consists of three key components:

- A set of game agents, an administrative console, and a dynamic honeynet. These three components interact in a semi-autonomous fashion to provide a means to identify, evaluate, and act upon network flows. The honeynet provides a means to collect malicious flows into a dynamically instantiated honeypot for later analysis of malicious activity to collect the forensic data pertaining to such attacks. Finally, the administrative console will allow a user interaction interface to provide control over traffic, data, and maintain the configuration settings for the various components.

### Traffic Inspection Component

Each game agent will be responsible for implementing the game logic for the corresponding part of the network by determining the state of the system, generating action sets, and performing defense actions. We will also introduce a central game coordinator which continuously receives information from all the game agents and keeps a summary state of the whole network. The game agents will be the only GIDA presence on the production network’s environment, alerting a transparently connected one. This transparent operation allows the game agents the capability of acting automatically on malicious network flows without needing to affect the configuration of the network and the information assurance appliances themselves.

The transparency operation also allows access to the change at any point in time. Furthermore, the game agents might come under direct attack. However, the game agents will use the information gained as inputs to a game model in order to identify the best defense measure. The game agent can transparently drop or redirect malicious packets instead of having the administrator create a temporary firewall rule to drop the packets.

### One System to Bind Them All

FreeBSD is a freely distributable operating system that offers advanced networking, performance, security, and compatibility features that make it well suited for building a robust game defense system. Originally derived from the University of California Berkeley BSD (UCB-BSD), FreeBSD enjoys a long history of use as high-end network servers and a worldwide support base.

The FreeBSD kernel possesses a unique feature that suits our project’s requirements. FreeBSD provides delivery and control messaging. This feature, Netgraph, is an extensible framework for in-kernel operations involving network traffic and device interaction. The netgraph consists of modules, called nodes, that provide some arbitrary functionality that can either operate directly on traffic data or abstract control commands delivered as part of the generic message delivery system. Edge nodes represent points of ingress and egress for the netgraph and are used to control either of the two traffic types available: control and data.

Netgraph nodes can be connected and restructured at runtime in any arbitrary topology. Many usable node types have already been developed and provide generic means to manage traffic and execute the process of protocol development and proof-of-concept designs.

### Netgraph Implementation

Netgraph exists as a collection of independent kernel modules that can have an arbitrary number of functions, called hooks, that exist to connect nodes to one another. A hook is a function whose purpose is to either send or receive data and control messages. At minimum each network node must have at least two functional nodes, control, and receive. These functions are then connected via a callback mechanism where the sending function of a node calls the receiving function of its adjacent peer. Further, control messages can be sent across these hooks or delivered globally. Control messages are used to structure the netgraph topology as well as provide a generic means to deliver custom control to nodes.

The following figure provides a visual representation of how netgraph is utilized to provide traffic delivery from bro and into click. It also shows the independent node that represents the game control node that will function to deliver control messages for action decisions across member nodes and act as the main control channel for the game agent. The diagram also depicts the flexibility of configurations by including an optional interface to observe all traffic as it passes from one node and into the next via a tee type node.

### Transparent Redirection

Transparent delivery of malicious traffic is achieved by utilizing a novel approach to bridging at the data link layer. When a new malicious flow is identified by the Control module, a creation message is sent to the honeynet gateway and a new honeypot is spawned for the honeypot. This honeypot assumes the identity of the target machine, this identify includes the IP address, hostname, and application specific configuration. When the honeypot is ready to accept connections, the MAC address is altered and associated with the malicious flow in the form of an ARP record. This MAC address replaces the destination MAC in each ethernet frame as it enters the redirection module, likewise the original target MAC address is written as the destination MAC. When frames leaves the redirection module and injected into the outgoing packet stream at the game agent. The isolated network and redirection component allows this trickery to take place without affecting otherwise normal traffic in the network.

### Limitations

Any security architecture is limited by the capabilities of their component parts, and GIDA is no different. While this system shows great promise there are a few areas that need more attention. The following list outlines some of these limitations:

- \( \text{NIDS policy needs major refinement and currently only considers simple cases of malicious traffic.} \)
- \( \text{The game model is severely limited in terms of available actions. These actions are currently centered around dropping, redirection, and logging of raw flow data.} \)
- \( \text{In some validation tests exceeded the network TTL for a packet, then a flow is typically reset under normal operation.} \)
- \( \text{A functional means to assess the capabilities and services that exist on the protected network.} \)
- \( \text{The inclusion of this assessment within the GIDA architecture would greatly improve the effectiveness of our framework and provide improved accuracy in the defense of the GIDA protected network.} \)

### Future Work

- \( \text{Utilize current research in the areas of Network, and Attack Graphs in order to provide an accurate assessment of network services. These assessments will then be used to generate possible action sets for predicting the attackers probable steps in attacking the network as well as methods of preventing these attacks.} \)
- \( \text{Utilize Bro’s ability to dynamically modify inspect policies during runtime to provide an evolving inspection element and improve the accuracy of the inspection process.} \)
- \( \text{Implement multiple game control modules that are based on differing game models in order to measure the performance and applicability of each possible control model.} \)

---

**GIDA: Agent**

---

**Transparent Redirection Component**

The Click Modular Router is a flexible routing framework developed by the MIT Lab for Computer Science. Click provides an extremely extensible framework for creating and extending elements for the purpose of routing network traffic.

Some interesting features of Click:

- \( \text{MIT-BSD-like License called ‘the Click license’} \)
- \( \text{Element based structure} \)
- \( \text{Elements are written in C, \texttt{C++}, or \texttt{Java}} \)
- \( \text{Capable of operating at Data Link layer (OSI Layer 2)} \)