GIDA : A Game Inspired Defense Architecture

C. Ellis, V. Datla, and H. Bedi
Department of Computer Science: GTCS Research Lab
University of Memphis
Advisor: Dr. S. Shiva

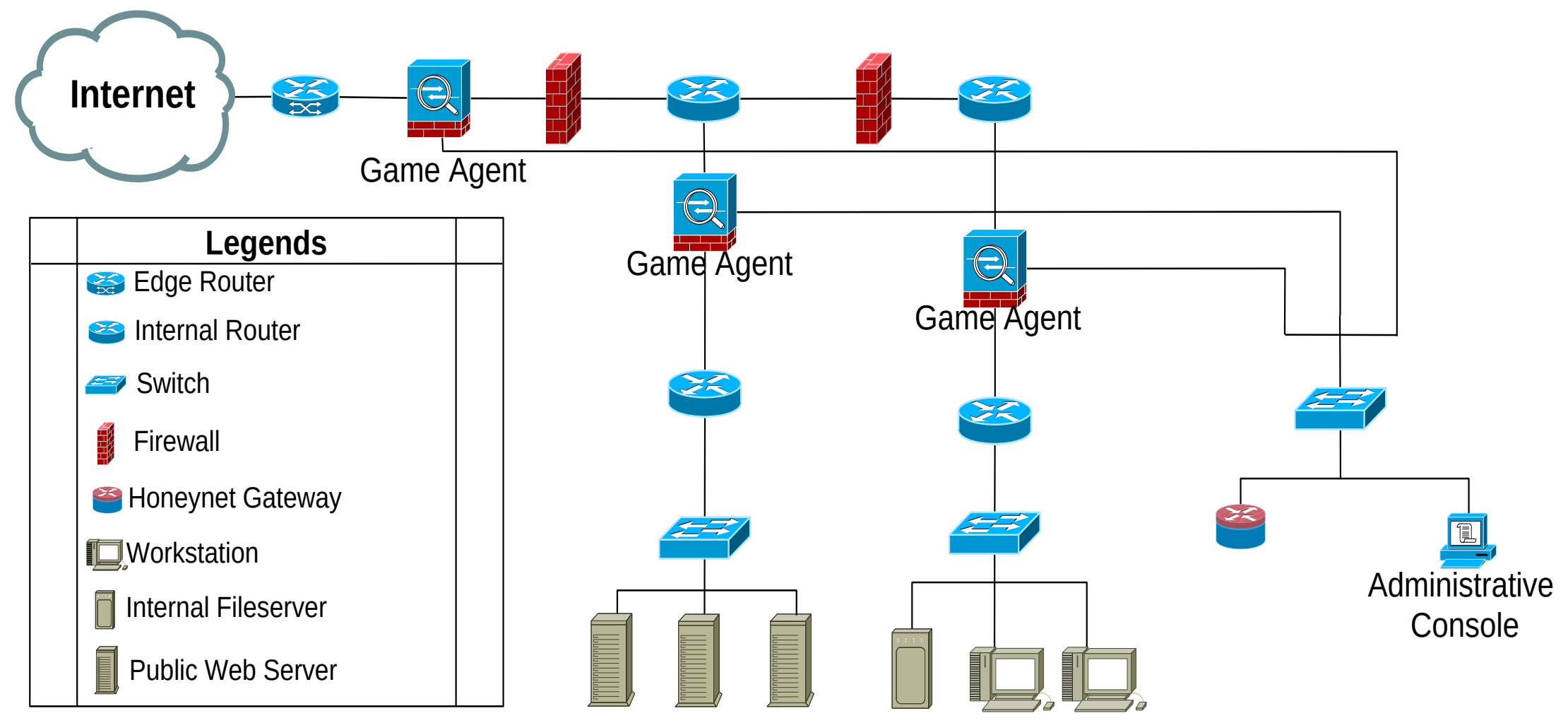
Introduction

The weakness of traditional network security solutions is that they lack a quantitative decision framework. As game theory deals with problems in which multiple players with contradictory objectives compete with each other, it can provide a mathematical framework for modeling and analyzing network security problems. As an example, a system administrator (defender) and an attacker can be viewed as two competing players participating in a game. In addition, game theory has the capability of examining large possible scenarios before taking the best action; hence, it can considerably enhance the decision making process of the system administrator. In cyber warfare, NO ONE is guaranteed information dominance in terms of intelligence and accessibility. As a result, a game-theoretic approach of collaboration (carrot) and compelling countermoves (stick) needs to be played efficiently. This notion is not unlike the mutually assured destruction (MAD) of nuclear deterrence. An open research problem is how to design a game-theoretic defense framework based on realistic assumptions considering a wide variety of cyber scenarios. It is to this end that we propose a unified framework for the defense of such nefarious actions on computer networks.

GIDA

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 in order to provide a means to identify, evaluate, and act upon network flows. The honeynet in particular, provides a means to redirect malicious flows into dynamically instantiated honeypots for observation of malicious activity and collecting the forensic data pertaining to such activity. Finally, the administrative console will provide a user interface that will allow for the correlation of network state data, provide a control channel for messaging, perform forensic analysis of honeypot data, and maintain the configuration settings for the various components.



GIDA: Game Agent

Each game agent will be responsible for implementing the game logic for the corresponding part of the network by determining the system state, generating action sets, and performing defense actions. We will also envision 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, albeit a transparently connected one. This transparent operation offers 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 therein. Transparent operation also greatly reduces the chance that the game agents might come under direct attack. Furthermore, 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.

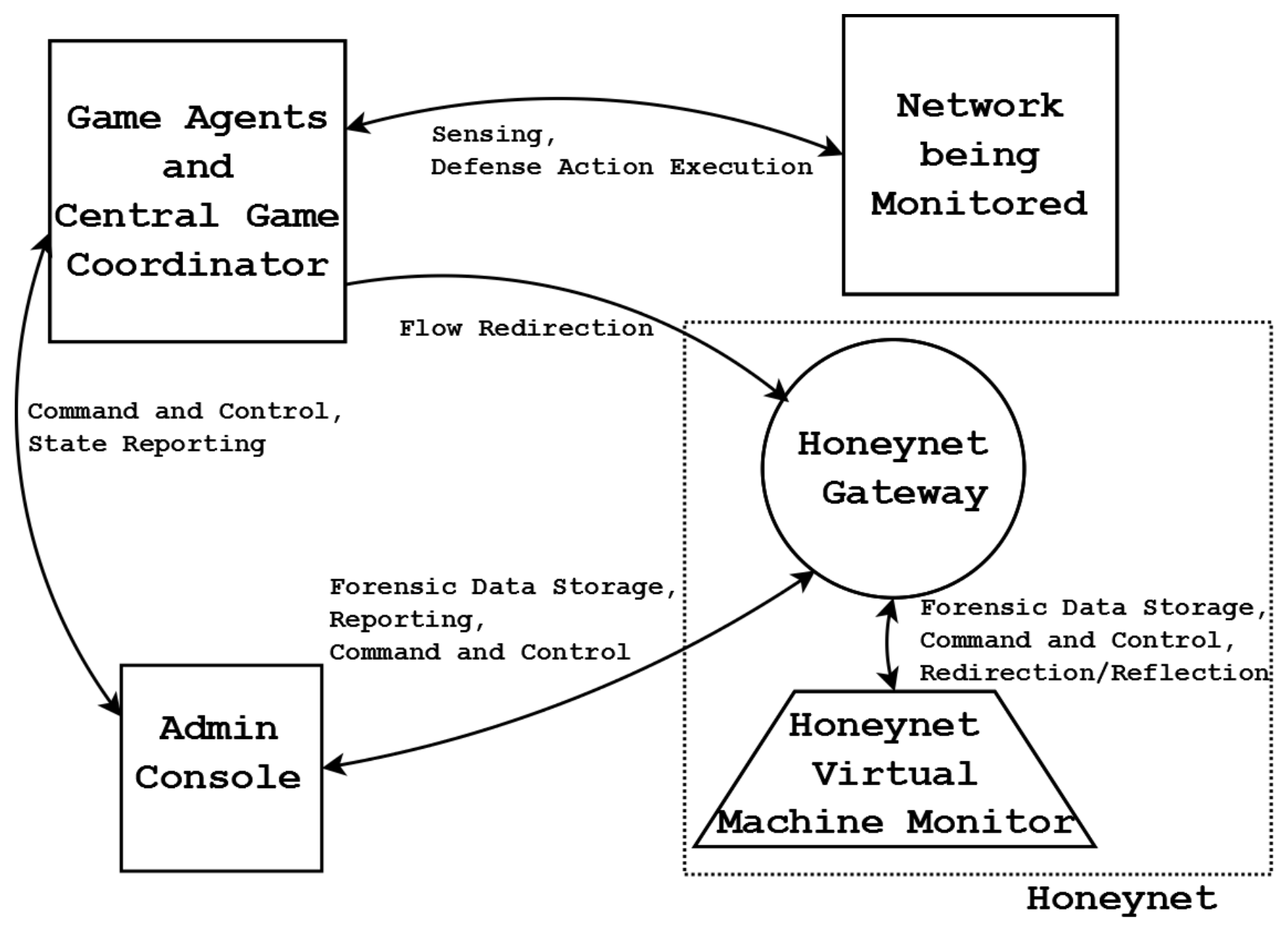
Game Agent Implementation

We chose to begin our proof-of-concept implementation of the GIDA framework by starting with the most complex component: the game agent. At first one might image that the features we wish to provide would require a great deal of development 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:

- Flow Identification
- Game Model Implementation
- Transparent Traffic Control
- Transparent Traffic Inspection
- Control Messaging

GIDA Component Workflow


Game Agents and Central Game Coordinator
Sensing, Defense Action Execution
Network being Monitored
Flow Redirection
Command and Control, State Reporting
Honeynet Gateway
Forensic Data Storage, Reporting, Command and Control
Forensic Data Storage, Command and Control, Redirection/Reflection
Admin Console
Honeynet Virtual Machine Monitor
Honeynet

Traffic Inspection Component

Bro is a Unix based Network Intrusion Detection System (NIDS) developed by Vern Paxson at Lawrence Berkeley National Lab and the International Computer Science Institute. Bro utilizes a tap interface to observe traffic and is capable of cooperating with other programs to initiate actions based upon network events.

Some interesting features of Bro:

- Policy based NIDS
- BSD style license
- Signature based detection capabilities
- Anomaly based detection capabilities
- Support for well established SNORT signatures
- Extensible policies through custom scripting language
- Dynamic policy modification at run-time
- Events can initiate cooperation with other processes
- Cluster operation

Transparent Redirection Component

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

Some interesting features of Click:

- MIT/BSD-like license called "the Click license"
- Element based structure
- Elements are written in C, or C++.
- Capable of operating at Data Link layer (OSI Layer 2)

One System to Bind Them All

FreeBSD is a freely distributable operating system that offers advanced networking, performance, security, and compatibility features that make for a well rounded research platform. Originally derived from the University of California at Berkeley's release of UNIX® (BSD), FreeBSD enjoys a long history of use as high-end network servers and a worldwide support base.

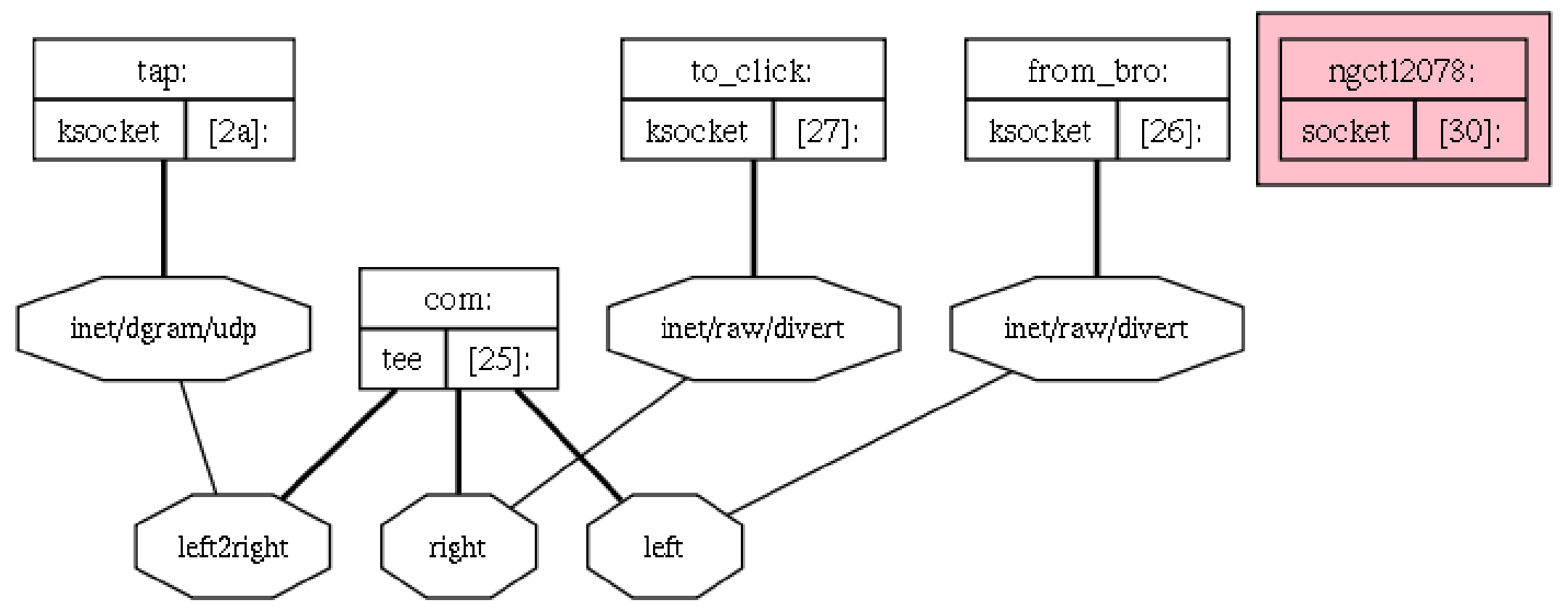
The FreeBSD kernel posses a unique feature that suits our project's requirements for traffic delivery and control messaging. This feature, Netgraph, is an extensible framework for in-kernel operations involving network traffic and device interaction. The netgraph consists of modules, called nodes, that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

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

Netgraph Implementation

Netgraph exists as a collection of independent kernel modules that have an arbitrary number of function, 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 data node must have at least two functions: send(), and receive(). These functions are then connected via a callback mechanism where the sending function of a node calls the receive function of it's 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 messages across member 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 displays the flexibility of configurations by including an optional interface to observe all raw traffic as it passes from one node and into the next via a tee type node.

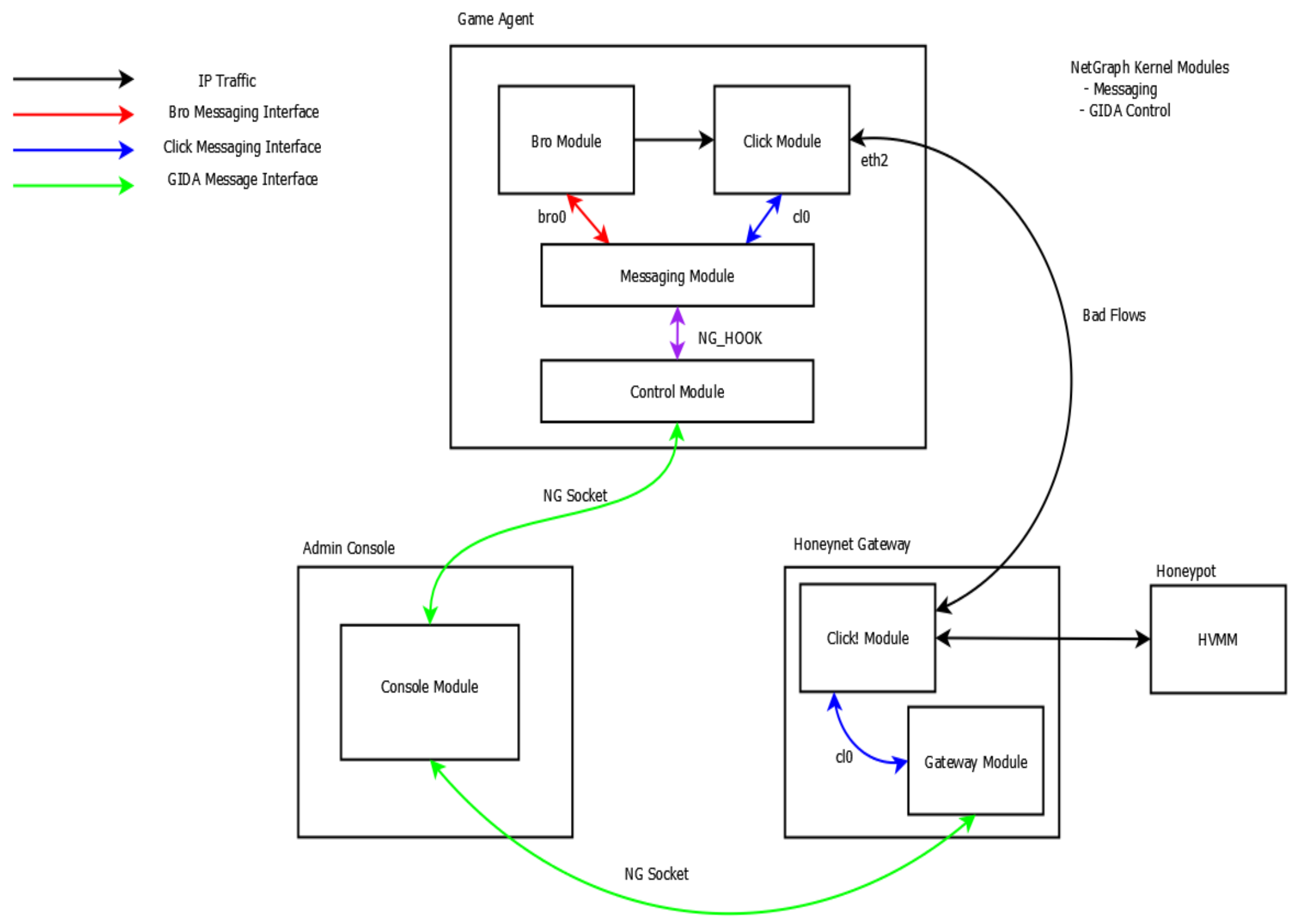


Game Model

We plan on developing the GIDA framework to be as flexible as possible in terms of available game theoretic models for use within the system. By implementing the model as a netgraph node we gain the ability to pick and choose any arbitrary model given that it provides the expected methods for interaction within the GIDA framework and change these nodes in real-time.

Our first game model features an imperfect information game in which each player is unaware of the actual state of the system but can associate a probability that the system is in X state at Y time and can choose the most applicable action based upon a given action set and overall system state.

Integration Visualization



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 instance is created upon the honeynet. 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 stored and associated with the malicious flow in the form of an ARP record. This MAC address replaces the destination MAC on each ethernet frame as it enters the redirection module, likewise the original target MAC address is written as the source address when frames leaves the redirection module and injected into the outgoing packet stream at the game agent. The isolated nature of the game network 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 still a few areas that need more attention. The following list outlines some of these limitations:

- NIDS policy needs major refinement and currently only considers simple cases of malicious traffic.
- The game model is severely limited in terms of available actions. These are currently centered around dropping, redirection, and logging of raw flow data.
- If the honeypot instantiation exceeds the network TTL for a packet, then a flow is typically reset under normal operation.
- A functional means to assess the capabilities and services that exist on the protected network. 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

- 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 for defense actions.
- Utilize Bro's ability to dynamically modify inspection policies during runtime to provide an evolving inspection element and improve the accuracy of the inspection process.
- 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.