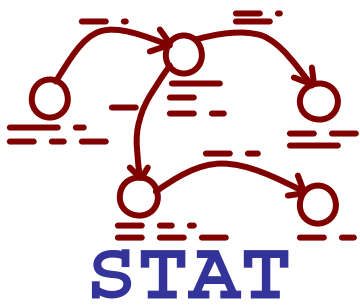


Designing a Web of Highly-Configurable Intrusion Detection Sensors

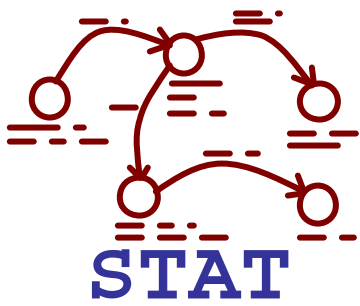
Giovanni Vigna, Richard A. Kemmerer, and Per Blix
RAID 2001

Reliable Software Group
University of California Santa Barbara
<http://www.cs.ucsb.edu/~rsg>



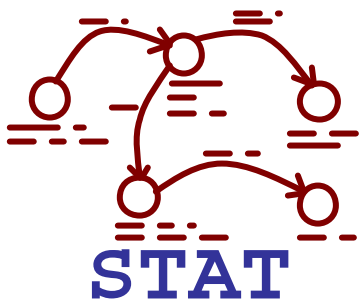
Intrusion Detection

- Intrusion detection traditionally based on analysis of low-level events: network packets, system calls, audit records
- Intrusion detection has evolved in several ways
 - New analysis techniques
 - Multiple event sources, possibly introducing distribution
 - Abstraction: fusion/correlation of high-level events, e.g., alerts
- Monitor and surveillance functionality always/still based on sensors



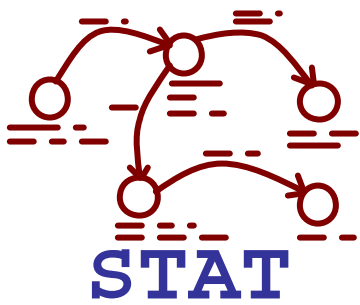
Intrusion Detection Sensor Limitations

- Sensors are developed in an ad hoc fashion to match specific environments/domains/event sources
- Sensors are hard to configure
- Sensors are hard to control
- Sensors are hard to extend
- Configuration/control/extension is mostly executed statically
- Configuration is mostly done manually
- Identifying “meaningful” sensor configuration can be difficult
- Number of sensors that can be easily managed is small



A Web of Sensors

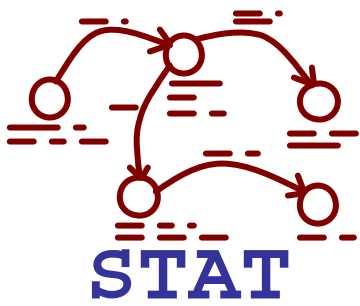
- Set of heterogeneous sensors that provide intrusion detection functionality within a protected network
 - STAT Framework
 - STATL and the STAT core
- Sensors controlled, coordinated, and configured by means of a distributed infrastructure
 - MetaSTAT
- Explicit modeling of component dependencies and current sensor configuration supports automated “meaningful” reconfigurations



The STAT Framework

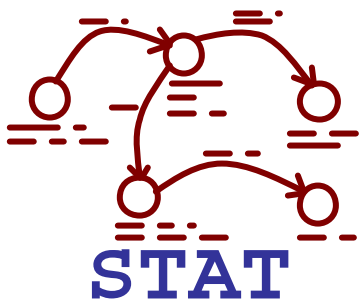
Framework supporting the development of intrusion detection infrastructures in heterogeneous environments

- Based on the *State Transition Analysis Technique*
- Defines a “core” language, STATL, that defines domain-independent abstractions
- Provides a “core” module that implements STATL semantics
- Supports development of core extension modules (Language Extensions, Event Providers, Attack Scenarios, Response Modules)
- Provides a communication and control infrastructure



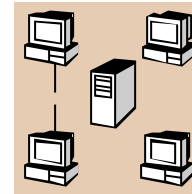
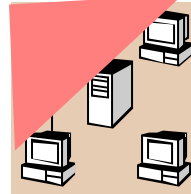
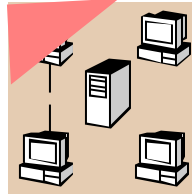
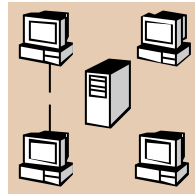
State Transition Analysis Technique

- STAT models penetrations as a sequence of state transitions
- Represents only key activities that lead from an initial safe state to a final compromised state
 - Signature Actions
 - State Assertions

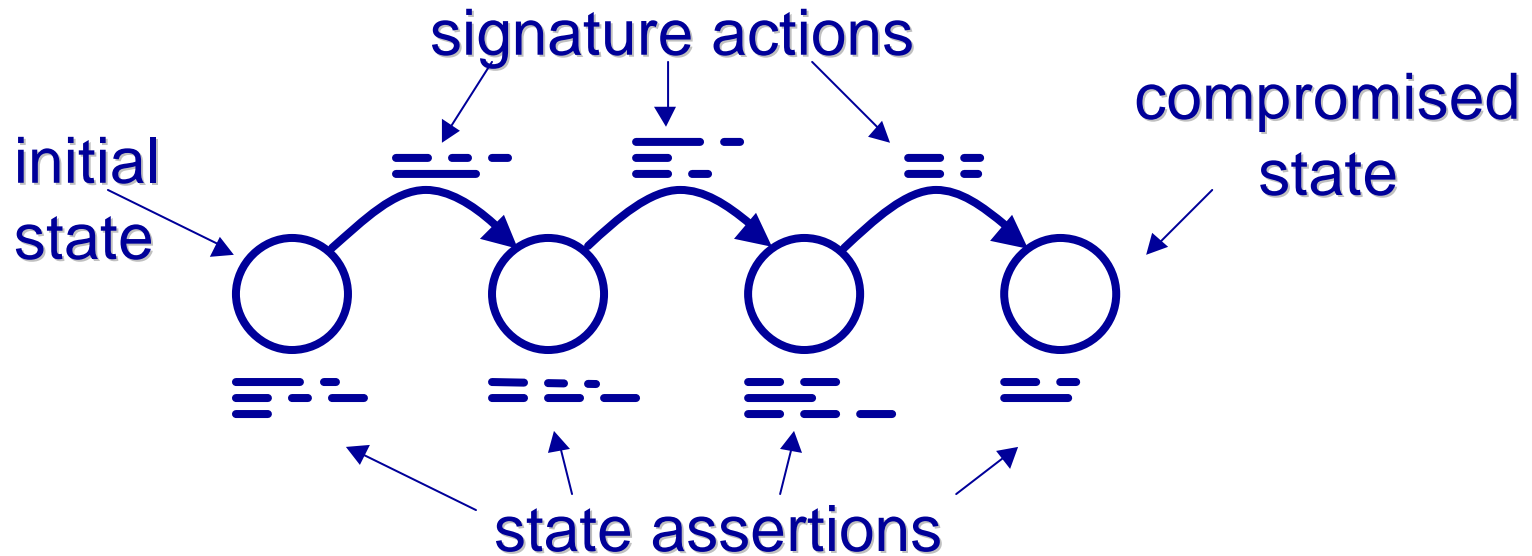


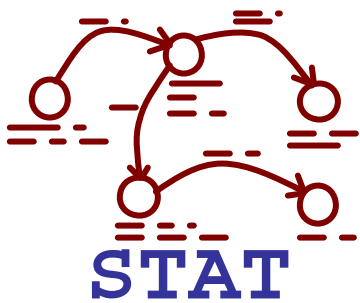
State Transition Diagrams

Attacker has limited privileges



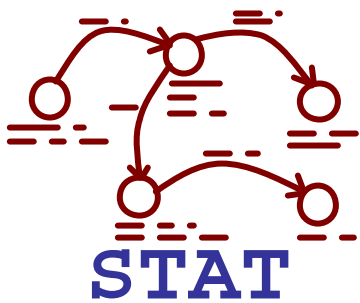
Attacker illicitly gains more privileges





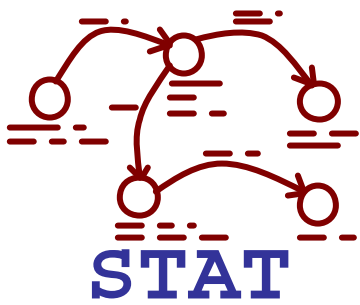
STATL

- A STATL specification is the description of a complete attack scenario (a signature) in terms of states and transitions
- Domain-independent language
 - Extensions for
 - IP networks
 - Solaris BSM
 - WinNT event logging facility
 - Apache event logs
 - Syslog facility
 - IDMEF Alerts
- Parameterized descriptions
 - Generic attacks customizable by installation or policy



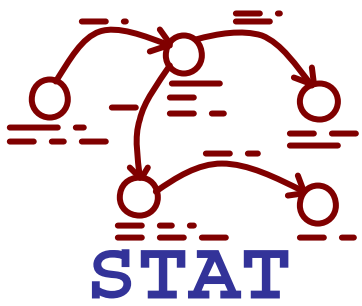
The STAT Core Module

- Implements STATL basic abstractions
 - Scenario
 - State
 - Transitions (consuming, non-consuming, unwinding)
 - Signature actions
 - Assertions
 - Global environment
 - Local environment
 - Code fragments
 - Events
 - Timers
 - Synthetic events
- Defines general semantics
 - Event matching
 - Scenario processing
 - Unwinding
- Can be dynamically extended to build a STAT-based sensor
 - Scenario plugins
 - Language extensions
 - Event providers
 - Responses modules

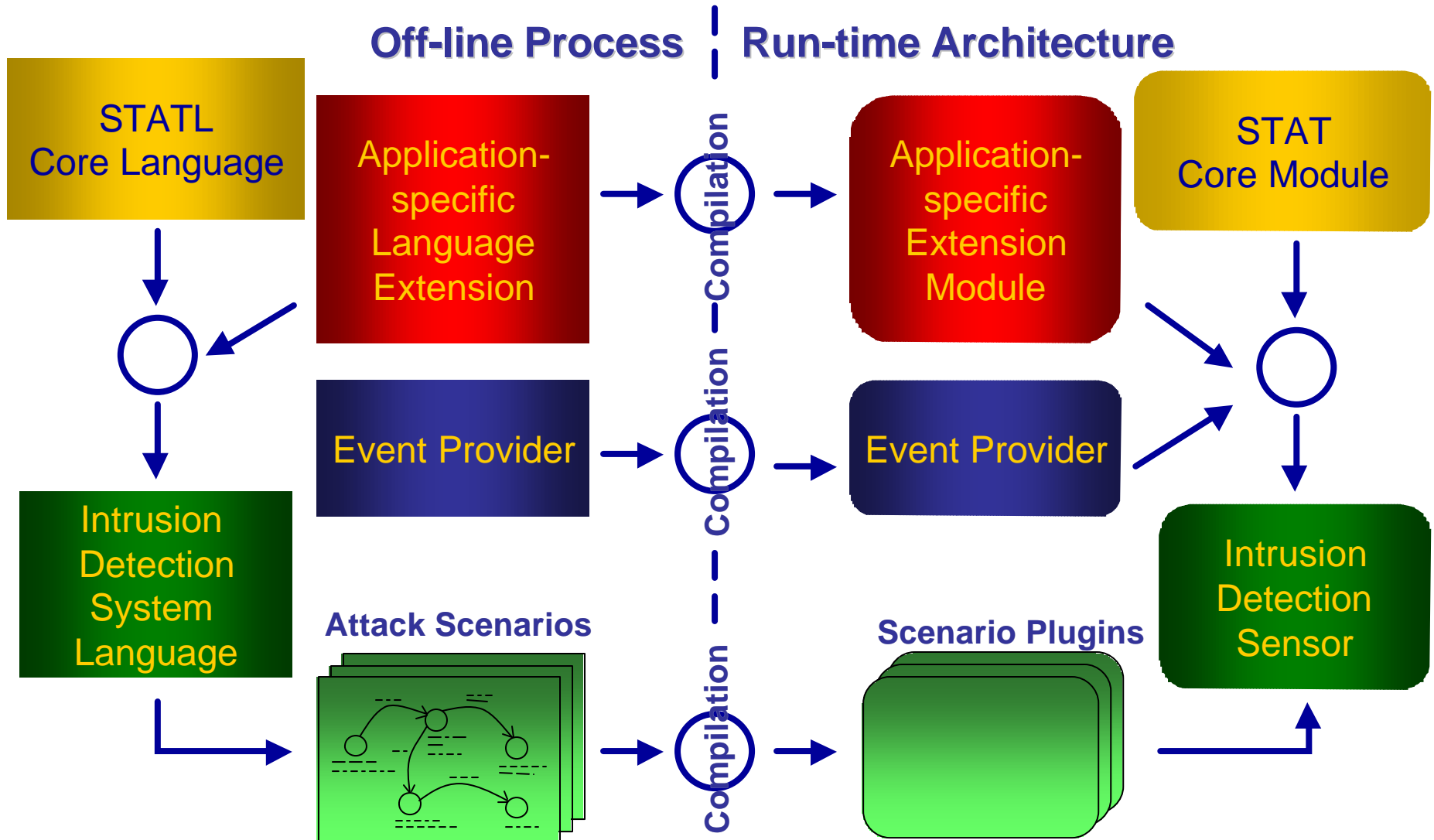


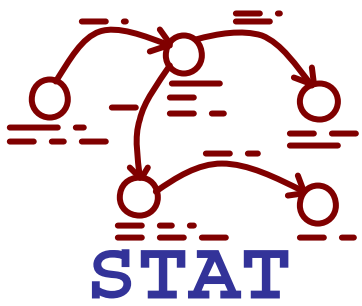
The Framework At Work

- Define a Language Extension, i.e., the events, types, and predicates to be used in a specific domain
- Compile the extension into a Language Extension Module
- Develop an Event Provider that transforms external data into events as defined by one or more Language Extensions
- Compile the Event Provider into a dynamically linkable module
- Develop STATL scenarios that use the events defined in one or more Language Extensions
- Translate/compile the scenario into a Scenario Plugin
- If necessary, develop response libraries to be used with the scenario
- Link everything together (shake well) and run your sensor



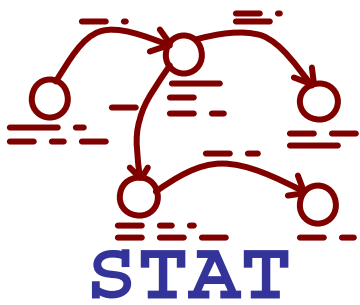
Creating a Sensor





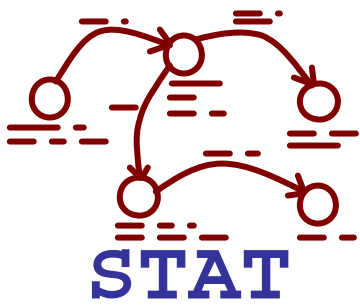
OK, You Can Develop Your Own IDS, But...

- What if one wants to change the configuration of a sensor at run time, without having to stop the whole thing?
- How can one be sure that all the pieces (extensions, providers, scenarios) fit together?
- What if one wants to control a multitude of sensors deployed throughout the network?
- What if one wants to aggregate/fuse/correlate the alerts produced by the deployed sensors?



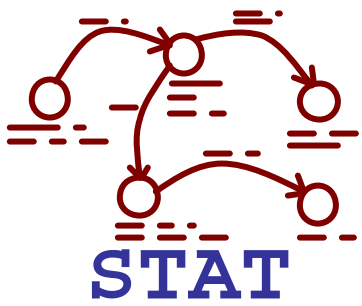
MetaSTAT

- A communication and control infrastructure for STAT-based sensors
- CommSTAT communication infrastructure allows for the exchange of alerts and control commands over secure connections
- MetaSTAT Controller dispatches commands to the sensors
- The STAT Proxy mediates communication
 - Performs local module management (installation/configuration)
 - Relays commands to sensors (loading/activation)

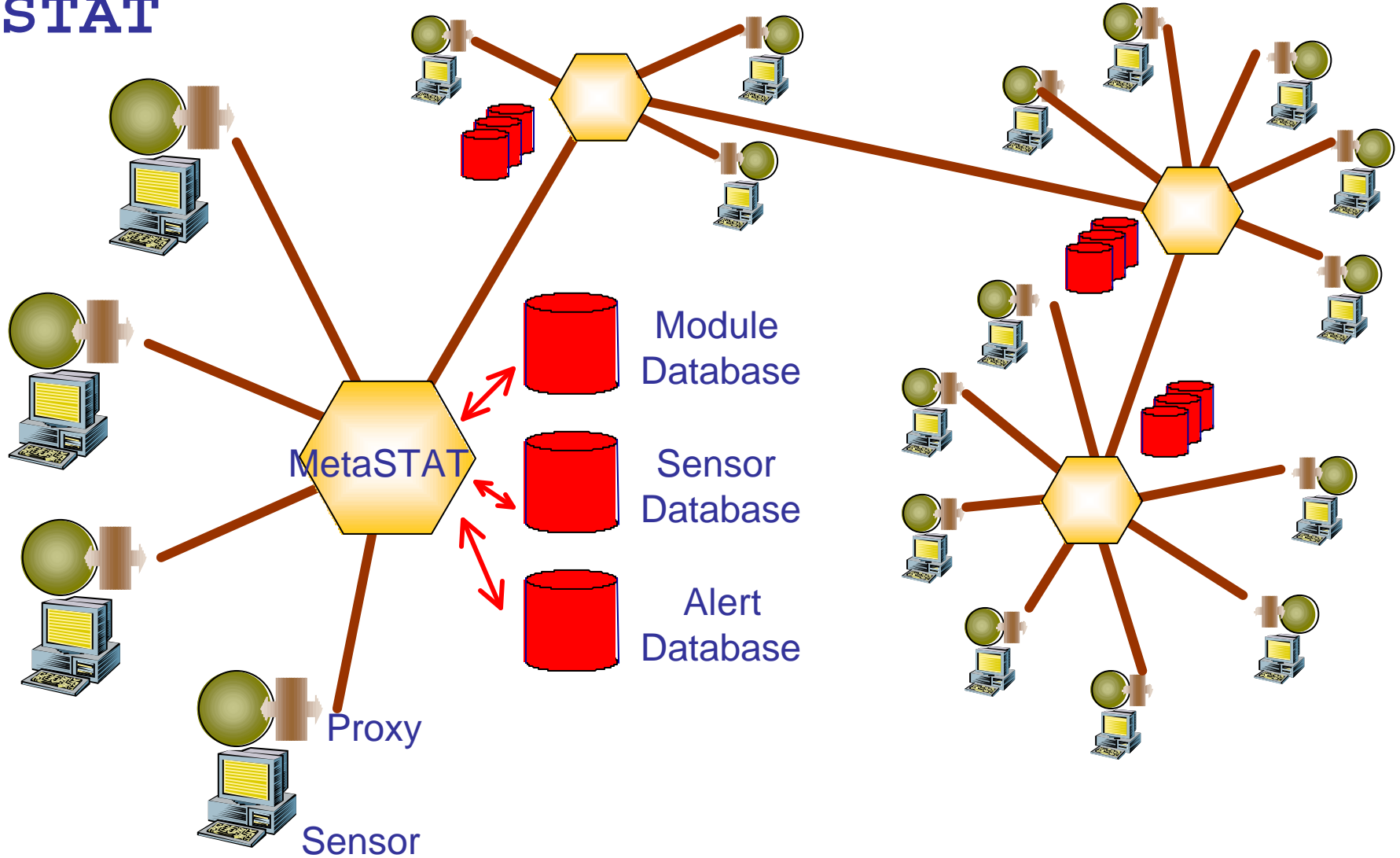


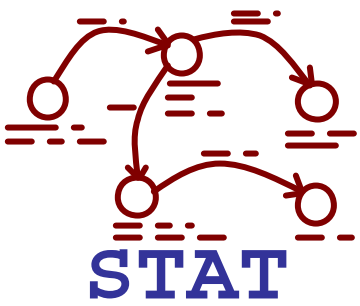
MetaSTAT

- MetaSTAT Configurator manages sensors
 - Database of available modules and corresponding dependencies
 - Database of current sensor configurations
 - Allows the manager to submit reconfiguration requests
 - Checks for meaningfulness of reconfiguration
- MetaSTAT Collector component aggregates sensor alerts in a centralized database to support analysis and correlation

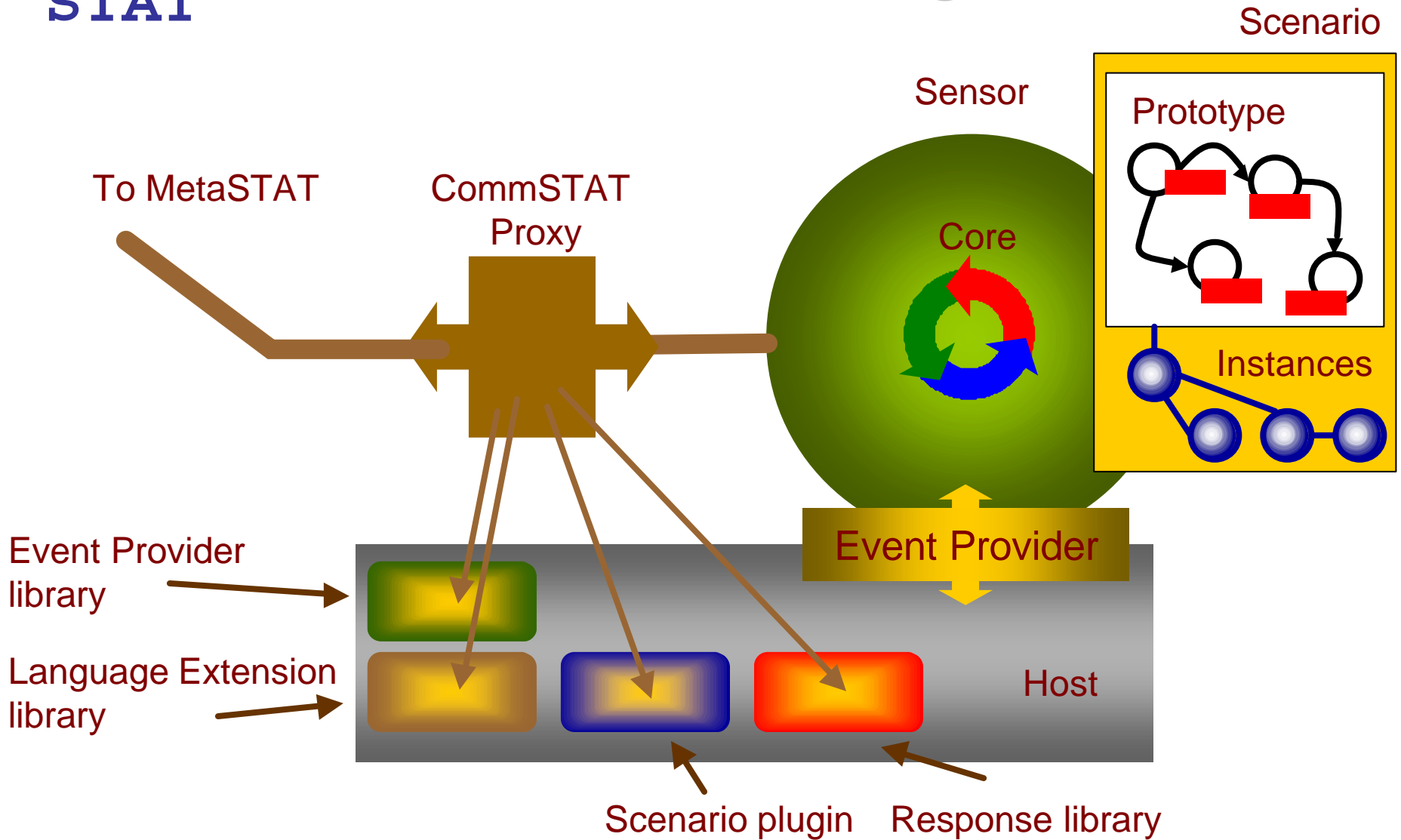


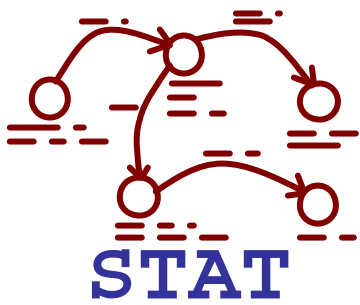
A Web Of Sensors





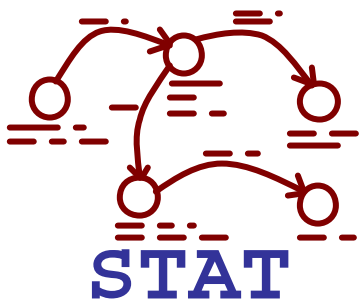
Sensor Configuration





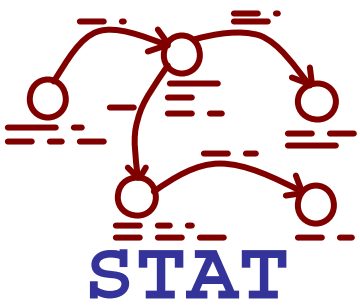
Module Database

- Models and stores the information about
 - The available *modules* (Language Extensions, Event Providers, Attack Scenarios, and Responses)
 - A number of *external components* (e.g., a specific auditing facility)
- Models and stores the dependencies between modules and components
 - *Activation dependencies*: Module A needs module B in order to be loaded and activated
 - *Functional dependencies*: Module A needs module B in order to produce meaningful results or any results at all

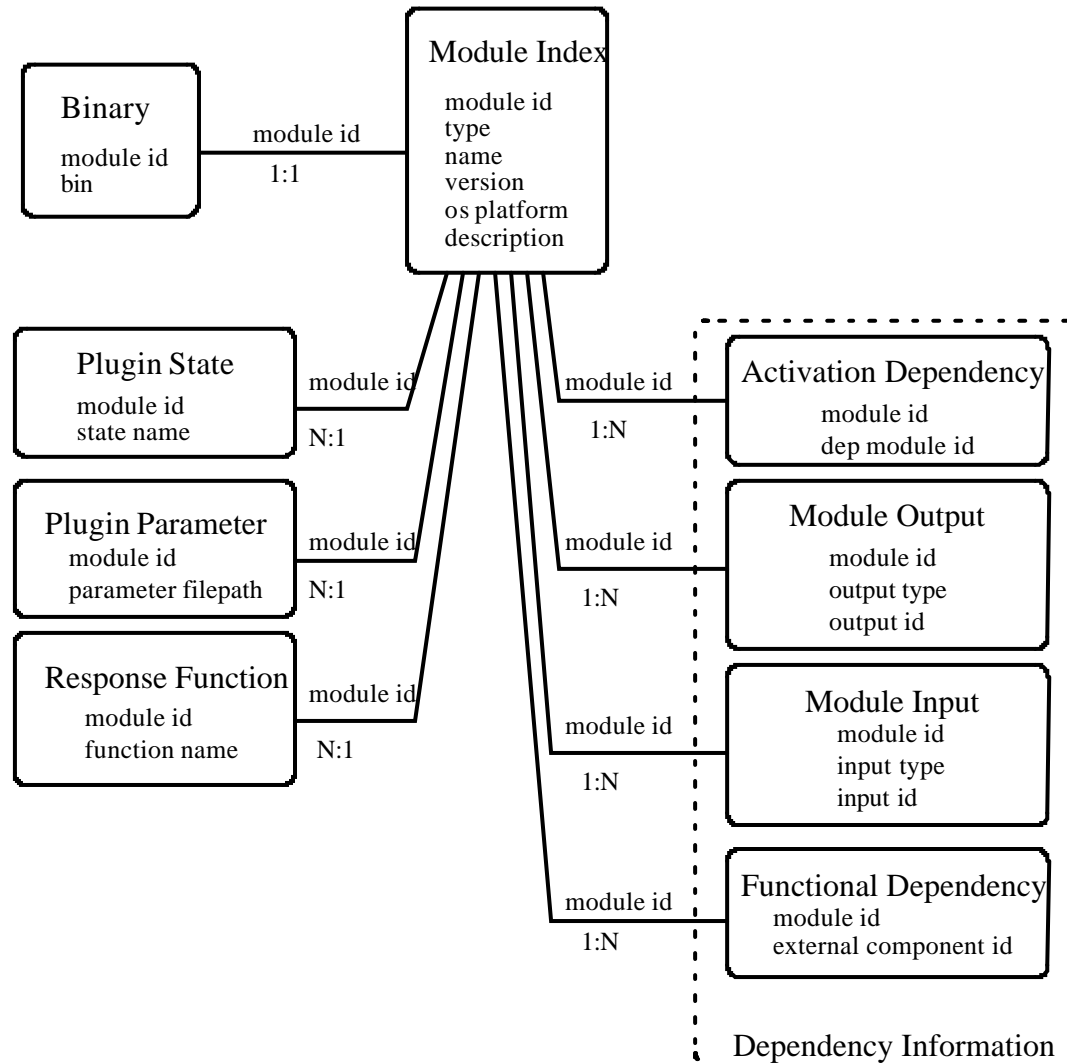


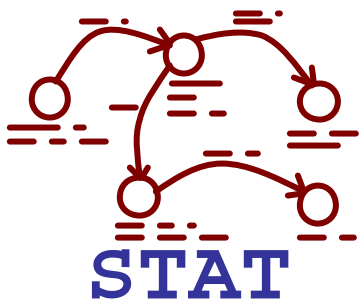
Module Management

- Each Module may be
 - Installed
 - Loaded
 - Activated
- A STAT sensor configuration is uniquely defined by a set of installed/activated modules and available external components
- A configuration is *valid* if all the activation dependencies are satisfied
- A configuration is *meaningful* if it is valid and all the functional dependencies are also satisfied



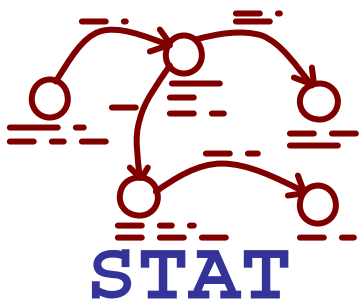
Module Database Schema



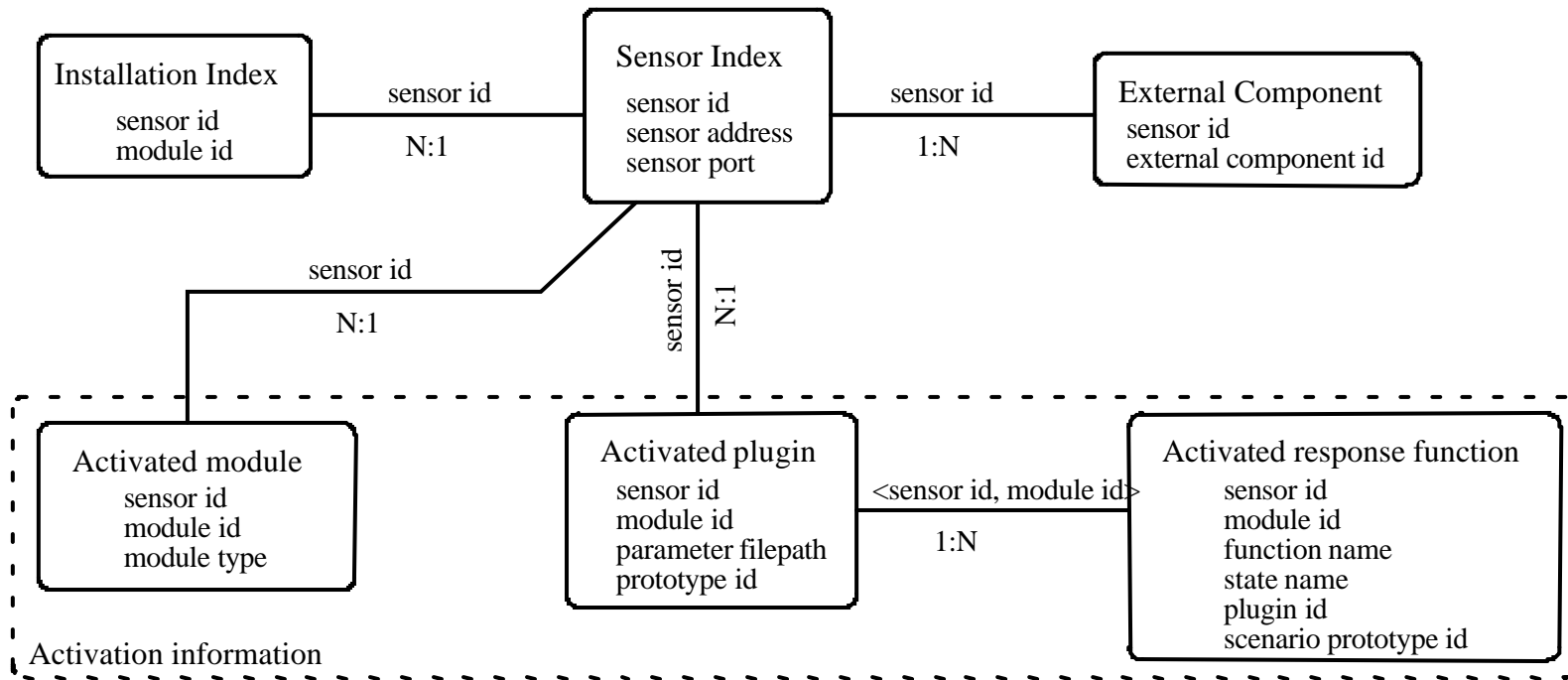


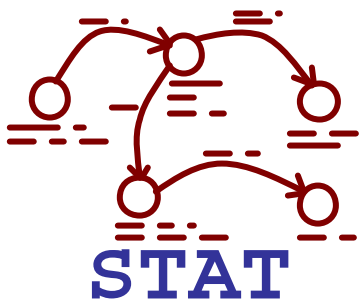
Sensor Database

- Models and stores information about the current configuration of a Web of Sensors
 - Installed modules (at each STAT Proxy site)
 - Loaded/Activated modules (in each STAT Sensor)
 - Available external components (at each host)



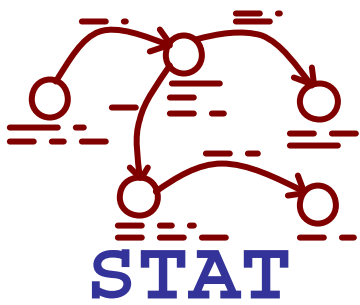
Sensor Database





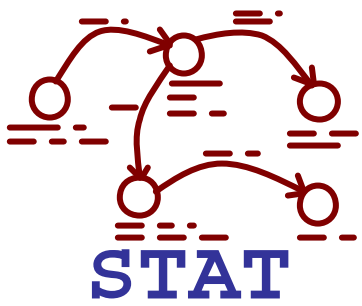
MetaSTAT Configurator

- Intrusion Detection Administrator (IDA) requires high-level reconfiguration
- The MetaSTAT Configurator determines the required sensor configuration examining the Module Database
- The MetaSTAT Configurator determines which modules are already available using the Sensor Database
- The MetaSTAT Configurator determines the steps that are necessary to complete the reconfiguration
- The MetaSTAT Controller sends the appropriate control messages
- STAT Proxies perform installation
- STAT Sensors reconfigure accordingly

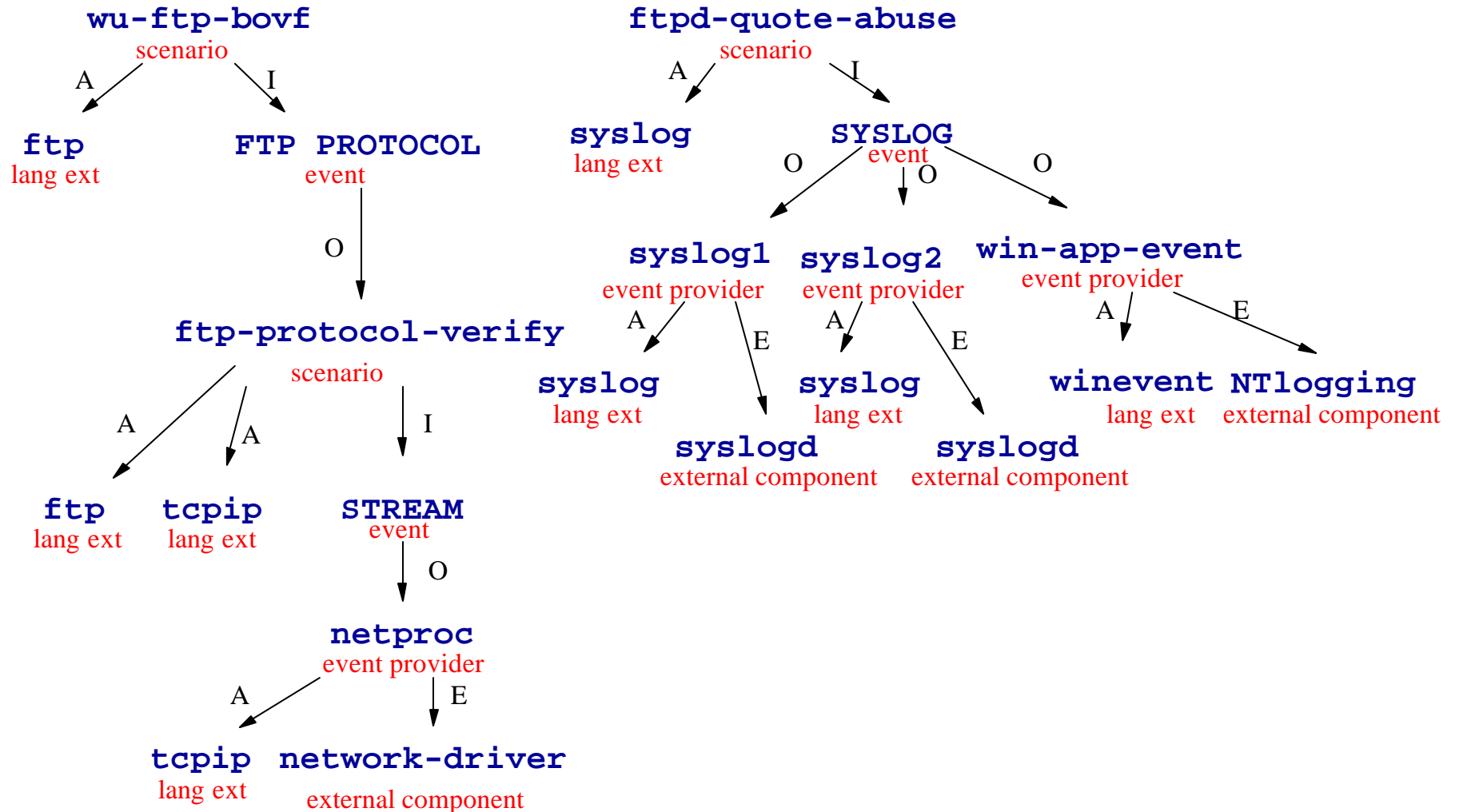


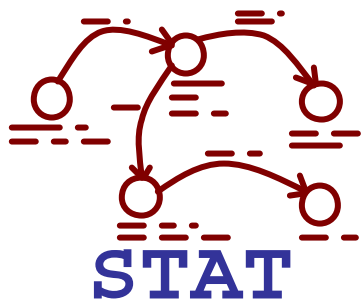
Example

- Intrusion Detection Administrator (IDA) wants to deploy FTP monitoring scenarios
- The Module Database is searched for suitable modules
- A subset is selected
- The Module Database is examined for possible activation dependencies
- The Module Database is searched for possible functional dependencies
- Results trigger a new series of queries



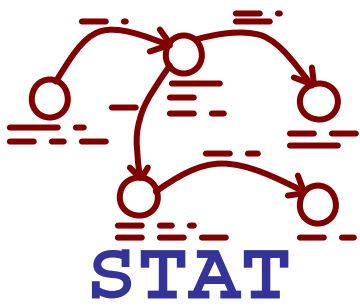
Dependency Graph





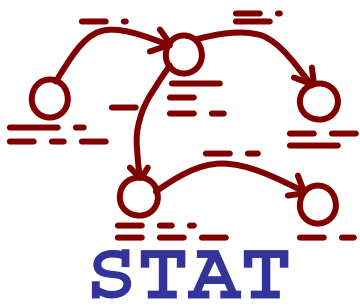
Example

- Configurator determines the complete set of dependencies
- Configurator compares required modules with installed/activated modules as stored in the Sensor Database
- Configurator compiles a *deployment plan*
- Plan passed to the Controller
- Controller ships messages to Proxies
- Proxies perform installations and forward loading/activation messages to sensors
- Detection begins...
- Possible custom responses are shipped/installed/activated



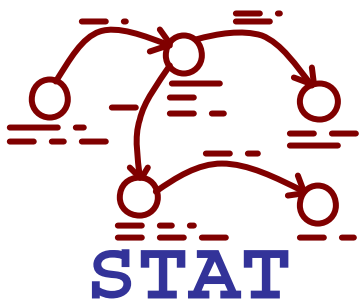
Advantages of the Approach

- High customizability
- Dynamic re-configurability
- Support for automated reconfiguration allows management of a high number of sensors
- Separation of analysis mechanisms from domain-dependent elements and response functionality
- Modules can be reused across sensors



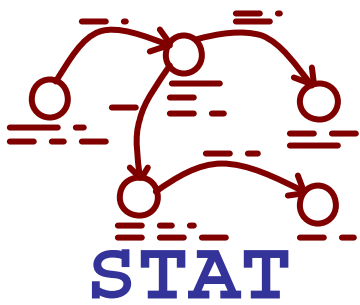
Advantages of the Approach

- Multiple Language Extensions and Event Providers can be used within the same sensor
- Responses can be associated with intermediate steps in attack scenarios
- Support for alert collection and distribution
- Third-party tools can be easily integrated through STAT Proxies



Future Work

- Web of sensors are usually associated with a single administrative domain
- Different Webs may require some sort of wide-area integration
- Use of the Siena content-based message delivery system to distribute alerts and control commands in wide-area networks
- Ultimate goal: Internet-scale coordination and control of intrusion detection capability
- Going beyond: re-configuring active attack scenario instances (load balancing, tracking mobile code, etc)



People Involved

- Richard Kemmerer
- Giovanni Vigna
- Per Blix
- Jacob Copenhaver
- Steve Eckmann
- Chris Kruegel
- Siva Sankaridurg
- Fredrik Valeur
- Jingyu Zhou