# Wireless and Physical Security via Embedded Sensor Networks[†**]

Michael J. Ocean
mocean@cs.bu.edu

Azer Bestavros
best@cs.bu.edu

Department of Computer Science
Boston University
Boston, MA 02215

## ABSTRACT

Wireless Intrusion Detection Systems (WIDS) monitor 802.11 wireless frames (Layer-2) in an attempt to detect misuse. What distinguishes a WIDS from a traditional Network IDS is the ability to utilize the broadcast nature of the medium to reconstruct the physical location of the offending party, as opposed to its possibly spoofed (MAC addresses) identity in cyber space. Traditional Wireless Network Security Systems are still heavily anchored in the digital plane of "cyber space" and hence cannot be used reliably or effectively to derive the *physical identity* of an intruder in order to prevent further malicious wireless broadcasts, for example by escorting an intruder off the premises based on physical evidence. In this paper, we argue that Embedded Sensor Networks could be used effectively to bridge the gap between digital and physical security planes, and thus could be leveraged to provide reciprocal benefit to surveillance and security tasks on both planes. Toward that end, we present our recent experience integrating wireless networking security services into the SNBENCH (Sensor Network workBench). The SNBENCH provides an extensible framework that enables the rapid development and automated deployment of Sensor Network applications on a shared, embedded sensing and actuation infrastructure. The SNBENCH's extensible architecture allows an engineer to quickly integrate new sensing and response capabilities into the SNBENCH framework, while high-level languages and compilers allow novice SN programmers to compose SN service logic, unaware of the lower-level implementation details of tools on which their services rely. In this paper we convey the simplicity of the service composition through concrete examples that illustrate the power and potential of Wireless Security Services that span both the physical and digital plane.

[†] This research was supported in part by a number of NSF awards, including CISE/CSR Award #0720604, ENG/EFRI Award #0735974, CISE/CNS Award #0524477, CNS/NeTS Award #0520166, CNS/ITR Award #0205294, and CISE/EIA RI Award #0202067.


*WiSec'08,* March 31–April 2, 2008, Alexandria, Virginia, USA.


## Categories and Subject Descriptors

K.6.5 [**Management of Computing and Information Systems**]: Security and Protection—*Unauthorized access*

## General Terms

Security, Management, Design

## Keywords

Sensor Networks, Wireless, Intrusion Detection, Physical Security

## 1. MOTIVATION

Wireless Network Security is a non-trivial problem and as such a variety of Wireless Intrusion Detection Systems (WIDS) have been created. WIDS deploy wireless probes/sensors to passively or actively monitor the MAC frames transmitted on the wireless medium and identify misuse by observing either suspicious characteristics of individual frames (*e.g.*, exhibiting characteristics imprinted by standard hacking tools) or a particular pattern in a sequence of frames (*e.g.*, sequences in violation of protocol standards). Wireless misuse includes illegitimate users attempting to gain access to the network (intrusion), man-in-the-middle attacks (*e.g.*, luring legitimate users into communication with a rogue access point), and various Denial of Service (DoS) attacks [5] (*e.g.* spoofing a legitimate wireless Access Point (AP) and sending a disauthenticate beacon to legitimate users).

While it is generally advantageous to secure a network at the lowest layer possible, the appropriate layer for incorporating security functionalities is highly dependent on the nature of the threat and whether such threat could be dealt with (*i.e.*, identified and managed) at that layer. For example, the threat of wireless intrusion is often dealt with using Layer-3 mechanisms (*e.g.*, content based packet filtering, IP address isolation), essentially ignoring the option of Layer-2 detection and prevention. Layer-3 IDSs are likely popular because there is far more data available at Layer-3, making it straightforward to respond to attacks, and because detection and response at Layer-3 is independent of the Layer-2 connection medium. On the other hand, wireless Denial of Service (DoS) attacks are much more difficult to deal with at Layer-3 as these attacks occur at Layer-2 and, even if detected, response is limited given that attackers will likely utilize fictitious or spoofed MAC addresses and may not have an IP address to retaliate against. Ultimately the only way to respond to these types of attack is to utilize information derived from the wireless medium (*e.g.*, received signal strength) to reconstruct physical location toward the goal of preventing further wireless transmissions from that user [10].

Wireless Intrusion Detection Systems provide mechanisms to identify, detect and locate DoS attacks, yet these systems are generally limited to logging or email alert response mechanisms. Many works ultimately recommend dispatching administration personnel to further analyze and respond to a detected attack – a costly and impractical solution in most situations. Instead, once the physical area of an attack has been derived it is possible to utilize automated responses from a variety of actuation hardware, if available; *e.g.*, embedded pan-tilt-zoom video cameras to gather an image, wireless detectors on pan-tilt motors to pin-point a signal, programmable robots to triangulate signal, a common message display (virtual bulletin board) in the environment informing users why their service has been interrupted and who is responsible. Additionally, there would be a clear benefit from including other, non-network centric *inputs* to the Wireless Network Security System (*e.g.*, driving a MAC whitelist from Bluetooth/RFID tracking, security camera images, passcard logs).

To achieve such cross-modal interaction within the context of a Network Intrusion Detection tool would require the generation of extensive, package and deployment specific software (modules, scripts, *etc*) that are, by their very nature, cumbersome to maintain. Indeed, such an approach is wrong headed. We observe that Wireless Network Security Services are specific, narrowly focused instantiations of an Embedded Sensor Network wherein sensory data includes the output of such monitoring tools. Rather than "hack" a Wireless Security System to include Sensor Network functionality, we advocate the inclusion of Wireless Security within a Sensor Network. Thinking differently about Network Security, the integration of new sensory data (*e.g.*, motion detection, face detection) and actuation responses expand Network Security beyond the digital plane and into the physical plane.

In this paper we detail our work to include wireless network monitoring devices as sensors in our Sensor Network infrastructure, the SNBENCH (Sensor Network Workbench). The SNBENCH provides a high-level programmatic interface to the resources of a Sensor Network (SN) and thus the inclusion of wireless network sensors enables intrusion detection and response services to be written quickly and easily. The SNBENCH has been designed with extensibility and modularity as a central tenet and therefore the changes required to include these new sensing modalities are quite modest. Moreover, the framework's modular nature allows a user to swap in any improved emergent wireless surveillance tool or technology (be it algorithmic or a physical turn-key device) with nominal effort and such changes would be transparent to their dependent services. We submit that our programmable, adaptable SN framework is the ideal foundation on which to compose Wireless Network Security services and physical security services alike, providing reciprocal benefit to each. The example programs given provide some insight into the highly customized, cross-modal Wireless Security behaviors that are possible in this context.

## 2. RELATED WORK

While many Network Intrusion Detection (Security) Systems exist (both commercial and open-source), we are presently unaware of any other work that leverages a programmable Sensor Network framework toward joint physical and Wireless Network Security, and thus believe we are unique in this regard. We present works that are related in three major thrusts; We distinguish between works that provide detection on a single wireless source (probe) as *Wireless Intrusion Detectors* (WIDs), those works that detect events across multiple detectors simultaneously as *Intrusion Detection Systems* (IDSs) and finally those that determine attack location as *Wireless Intrusion Detection Systems* (WIDSs). Although WIDSs contain a WID component, these works are not necessarily proper subsets of each other, as IDSs may not provide wireless detection.

### Wireless Intrusion Detection

Kismet [14] is the *de facto* open-source Layer-2 Wireless Intrusion Detector. Kismet passively scans 802.11 channels for activity and can be used for a variety of uses including finding hidden access points, mapping access points in a geographic region via GPS, or generating alert events when suspicious frames are detected. A Kismet deployment may consist of three distinct components, (1) a light-weight Kismet Drone that passively captures the wireless frames from its local interface and sends them to (2) a Kismet Server that processes the frames from one or more drones to detect either fingerprint or trend based suspicious activity and (3) an optional remote Kismet client that connects to the Server to receive notifications and render the results. The Kismet server may drive external wireless event notification by providing custom clients that communicate using the published Kismet protocol. Kismet may be configured as an Intrusion Detection System by associating several drones with a single server process to build a single, central wireless event log file. Kismet is not considered a Wireless Intrusion Detection System by our definition however, as the Kismet server does not indicate which physical drone is responsible for an alert which prevents spatial intrusion tracking. Kismet-newcore, a rewrite of the Kismet project, does preserve which drone generated a wireless alert event yet lacks a stable build at this time. Other tools have existed in the WID space prior to Kismet (*e.g.*, WIDZ [16]) but have been largely unmaintained in recent years. Similarly, AirIDS [17] set out to be the first open-source Intrusion Detection System aimed at 802.11 attacks, however the project never reached a stable release, is no longer available for download, and appears to have been abandoned.

### Intrusion Detection Systems

While Kismet is the *de facto* Layer-2 WID and IDS, Snort [19] is the *de facto* standard IDS for Layer-3 (IP traffic analysis). Snort is a mature IDS with a large user base and comprehensive set of detection rules for detecting malicious content in IP packets for a wide range of attacks. Snort also offers very basic response mechanisms (*e.g.*, logging or email alert mechanisms) however the Barnyard project not only aims to increase performance of logging output but also claims to enable the creation and use of custom output plug-ins. As Snort is aimed at Layer-3, it offers no support for wireless monitoring, however the Snort-wireless [15] project adapts the Snort rule engine for Layer-2 wireless use by adding wireless frame capture and replacing IP addresses with MAC addresses in rule processing. Unfortunately the Snort-wireless project has not been updated since late 2005 and plans for integrating Snort-wireless into Snort appear to have been abandoned.

In many ways, our vision is similar to that of modular IDSs (*e.g.*, [24], [23]). These modular (or so-called "Hybrid") systems are designed to allow various Intrusion Detection Software packages to be integrated as "Sensors" in the IDS. These works share the modular approach which is similar in spirit to the cross tool integration that we hope to provide to the Network Security community, yet these works are narrowly focused on issues of traditional Network Security. Our work enables the composition of sense and respond programs that manipulate both network data and physical sensory data (*e.g.*, image processing on embedded video cameras) in a manner that would be impossible on these IDS platforms without significant changes.

**Wireless Intrusion Detection Systems**

Tracking MAC addresses alone is insufficient for a WIDS as they may be easily spoofed by malicious tools [5]. The requirement that a WIDS must determine attack location is sensible, considering that Layer-2 DoS attack response generally requires physical intervention [10].

Nominally we expect we know the Cell of Origin (COO) of a detected wireless transmission (*i.e.*, the user's distance must be within the detection range of the physical location of the detection point), however while this might be useful for short range media (*e.g.*, Bluetooth, RFID) we'd like to obtain higher accuracy than the range of 802.11 (ranging from 200 to 25meters, depending on the physical layout). Many approaches to derive location from Signal Strength Information (SSI) of RF transmissions have been undertaken, including Microsoft Research's RADAR [4] which uses readings from multiple sensors to perform on-line triangulation, compared against off-line training data. Many works since have tried to loosen the off-line training needs of this work attempting to dynamically overcome issues of transmission reflection, diffraction and interference (*e.g.*, [25]). Work presented in [21] frames these goals directly in terms of reconstructing wireless entity location particularly for security purposes and a similarly goaled architecture is described in [12].

The work in [2] offers an architecture for a Wireless Intrusion Detection System that breaks from the norm slightly, in so far as it establishes wireless sensors that form a perimeter around an access point (or area) to be protected and uses directional antennae (opposed to the typical, omni-directional antenna found on WAPs) that would sweep the region to better pin-point the location of a particular wireless user. This work provides detailed analysis of particular directional antennas and is able to pinpoint wireless intruders accurately. Our work is compatible with the use of sweeping directional antenna; and perhaps more so than what is envisioned in [2] as the SNBENCH can address and direct the servos that control antenna movement explicitly, enabling on-demand target tracking. In particular, we envision either making the directionality of the wireless sensors explicit to be controlled within the service logic, or mounting the wireless antennae to the pan-tilt-zoom camera network that sweeps the perimeter of our SN testbed. Additionally we note that requiring a perimeter defense may be useful in some installations, but perhaps impractical in most. The ability to use modalities other than directional antennae for identification (*e.g.*, the use of cameras) extends the applicability of our approach.

In the commercial spectrum, the Wireless Intrusion Detection System AirDefense Enterprise [3] integrates the industry standard signal strength positioning system, Ekahau [9] to provide location tracking of wireless intruders[1]. IBM's Internet Security Systems' Wireless Products [11] are also popular, but lack location tracking. Both tools attempt to provide complete, turn-key detection and response systems for corporate wireless networks. Given their single-solution nature, these tools do not provide integration with third-party Intrusion Detectors or tools. Responses to wireless attack detection in these systems are more pro-active, for example, disauthenticating a malicious user to effectively kick him or her off the network, yet they do not offer an accessible programming interface to adjust the sense and respond behavior. While these solutions represent the upper echelon of commercial sense and respond WIDSs, their lack of extensibility makes cross-modal monitoring solutions (*e.g.*, utilizing video frames) unattainable.

[1]Popular turn-key hardware solutions to wireless device tracking also exist, *e.g.*, Cisco's 2700 Series Wireless Location Appliance [20].

# 3. SNBENCH OVERVIEW

To orient the reader to the platform to ease further discussion, in this section we briefly highlight the salient features of SNBENCH. The vision, goals and high-level overview of the SNBENCH infrastructure have been reported elsewhere [6] and implementation details may be found in [18].

SNBENCH consists of programming support and a runtime infrastructure for Sensor Networks comprised of heterogeneous sensing and computing elements that are physically embedded into a shared environment. We refer to such a physical space with an embedded SN as a Sensorium [7]. The SNBENCH framework allows Sensorium users to easily program, deploy, and monitor the services that run in this space while insulating the user from the complexity of the physical resources therein. We liken the support that SNBENCH extends to a Sensor Network to the support that higher-level languages and operating systems provide to traditional, single machine environments (language safety, APIs, virtualization of resources, scheduling, resource management, *etc*). SNBENCH is extensible by design such that new hardware and software capabilities may be painlessly folded into the infrastructure by its advanced users and those new capabilities easily leveraged by its novice users.

SNBENCH provides a high-level programming language with which to specify programs (services) that are submitted to the resource management component which in turn disseminates program fragments to the run-time infrastructure for execution. At the lowest level, each sensing and/or computing element hosts a Sensor eXecution Environment (SXE) that abstracts away specific details of the host and attached sensory hardware. SXEs are assigned tasks by the resource management components of SNBENCH the Sensorium Service Dispatcher and Sensorium Resource Manager in tandem monitor SN resources, schedule (link) and deploy (bind) tasks on to available SXEs. A graphical representation of this end-to-end support is shown in Figure 1.

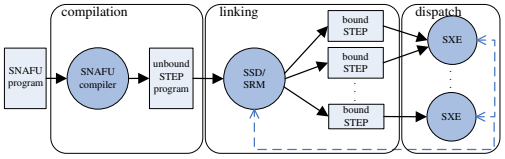


**Figure 1: The SN program life-cycle as enabled by SNBENCH. Rectangles represent data, circles represent tasks/processes, and the dashed lines represent control communication (dependency).**

The Virtual Instruction Set Architecture of SNBENCH is the Sensorium Task Execution Plan (STEP), a tasking-language used to describe complete programs and fragments alike. A STEP program is a graph of a SN programs's data-flow and computational dependency with the nodes of a STEP graph representing the atomic computation and sensing operations and edges representing data flow. In execution, demand for evaluation pushed down from the root of the graph to the leaves and values percolate up from the leaves back to the root. STEP nodes describe data, control flow (*e.g.*, repetition, branching) and computation operations that we refer to as STEP Opcodes and the SXE maintains implementations of the Opcodes with which it may be tasked.

Opcodes do not directly manipulate sensors, but rather manipulate SNBENCH typed data. Specific details of the sensor hardware of the SXE are abstracted away by a SensorHandler module that is capable of communicating with and reformatting the data from a specific sensor to produce to SNBENCH typed data; Support for

new sensor device types require the addition of new SensorHandler modules[2]. In SNBENCH there is a distinction between a SN Service Developer who uses high-level programming languages to compose Services by gluing together Opcodes and sensors (generally without regard for how the Opcodes are actually implemented beyond their type signature) and the SNBENCH "engineers" who are responsible for expanding the Opcode and SensorHandler libraries to enable new functionalities.

# 4. ENABLING WIRELESS MONITORING

SNBENCH is extensible by design insofar as support for new sensing devices may be added to the Sensor eXecution Environment (SXE) by providing implementations of two relatively small interfaces; a *SensorHandler* translates SNBENCH requests to interact with a specific device and a *SensorDetector* module must provide facility to detect new devices of this type and inspect their state. The SensorHandler is akin to a device driver, abstracting away the specific idiosyncrasies of the particular device's interface and enabling the device to be accessed by higher-level programming constructs. As far as the SNBENCH framework is concerned the abstracted device becomes just another managed input device/event generator only different from a video camera or motion sensor insofar as the datatype of its output.

To enable wireless network security service composition on SNBENCH, we have added two new sensors and a new actuator; The WifiAlertSensor reports wireless alert detection events, the WifiActivitySensor reports MAC addresses and Received Signal Strength Indication (RSSI) for any passively observed wireless activity, and the WifiResponder actuator sends a disauthenticate flood to a particular MAC address. Rather than implement wireless Layer-2 tools from scratch we opted to leverage several existing open-source software packages.

**WifiAlertSensor**

The WifiAlertSensor is a SensorHandler implementation that leverages the Kismet [14] wireless intrusion detector via a self-contained customized Kismet client. The Java based WifiAlertSensor class is hosted by a "non-lightweight" SXE and translates the proprietary Kismet client-server protocol into structured, typed SNBENCH objects (tagged XML) that encapsulate notifications from the Kismet server. The decision to use Kismet stems from its passive scanning ability, wide range of hardware support, and modular design (described in Section 2). While the decision to use this package in particular may be debated, the inclusion of any another functionally-equivalent Wireless Intrusion Detector would be equally straightforward.

A Kismet client may request to receive several types of Kismet messages from a Kismet server/drone pair (client traffic, AP detection, suspicious activity alerts, *etc*.). In the case of the alert sensor, we request notification of all wireless alerts supported by the current stable build of Kismet (detailed in Table 1). Whenever the Kismet server detects an Alert condition from its corresponding drone's data feed, an alert is sent to the WifiAlertSensor client which translates and buffers the alert message. In addition to translating the Kismet protocol, the WifiAlertSensor adds additional fields to alert message; a local timestamp to measure buffer service delay, a sensor source to identify the physical sensor (drone) that produced the message, and a severity field that indicates the relative threat of the particular attack (this corresponds to the values we have specified as the "danger" column in Table 1).

The WifiAlertSensor's message buffer is configurable in length (where length is measured in either size or time) and alert messages are retrieved from the buffer by Opcodes requesting data from this sensor. Implementation of the retrieval Opcode may impose a blocking or nonblocking semantic, as needed. In our experimentation we implemented a single Alert-centric Opcode, `sxe.core.wifi.get`, that performs a non-blocking read from the alert sensor's buffer to populate and return a WifiAlert. The WifiAlert datatype is a subtype of snStruct, with tagged fields corresponding to the fields populated by the WifiAlertSensor and thus accessing the data within a WifiAlert reuses the existing snStruct manipulation opcodes. A Service Developer retrieves WifiAlerts via the high-level function `DetectWifiAlert()` that is compiled into a call to the Opcode `sxe.core.wifi.get` with a WifiAlertSensor (or set of sensors) as a parameter. Complete examples of high-level service logic are given in Section 7.

**WifiActivitySensor**

The Activity sensor provides data regarding wireless transmissions that have been detected by a passive, promiscuous-mode wireless sensor. In particular we are interested in the MAC address of a transmission, the observed signal strength (RSSI) and the mode of the transmission (*i.e.*, Access Points, Clients, Ad-hoc participants). While determining physical location from RSSI is imperfect (as RSSI readings themselves may not be entirely accurate depending on the driver implementation and other physical factors) the use of RSSI readings can better pinpoint a MAC address' physical location than the simple cell-of-origin data alone. WifiActivitySensor maintains a hash-table of the detected wireless activity (keyed by MAC address), which can be used to either report new/updated wireless activity similar to the Alert sensor or to query the activity log to find information about a particular MAC address. Like the alert sensor the activity sensor also communicates with a remote sensor "server" process responsible for gathering data.

In practice we actually support two different physical implementations for the activity sensor server as the Kismet drone/Server does not retrieve RSSI on our preferred target platform. For Kismet RSSI supported hardware we use a client derived from the WifiAlertSensor implementation that requests and parses NETWORK and CLIENT messages from the Kismet server rather than ALERT messages. For the OpenWRT platform, we use custom code that sends ioctl's to the wireless device to put the device in passive, monitor mode, accept frames, and retrieve data from the frames and device (including the RSSI). Our program is based on code from the open source WiViz [22] package for OpenWRT, which contains the ioctl codes needed to achieve the proper device state and interaction. Like the Kismet server, this program provides notifications of activity messages which are received and hashed by the WifiActivitySensor.

The high-level Opcode `DetectWifiActivity()` is compiled into `sxe.core.wifi.get` with a WifiActivitySensor as a parameter and blocks until a new activity message is available from that sensor. Additionally `QueryWifiActivity()` (compiled into `sxe.core.wifi.find`) searches the WifiActivitySensor's hash table for the latest reading associated with the specified MAC address. Like the WifiAlertSensor, the samples returned from this sensor are encapsulated in an snStruct derivative type.

[2]SXEs can retrieve Opcode implementations at run-time, however support for loading new sensing devices at run-time is not currently supported. Such functionality is not difficult to support, and is analogous to dynamically loading device drivers to support new hardware.

| Name | Danger | Type | Description |
|---|---|---|---|
| PROBENOJOIN | none | Trend | A user is probing periodically without joining any AP. Windows XP clients cause this alert as normal behavior. |
| LUCENTTEST | none | Fingerprint | A site survey package is in use |
| NETSTUMBLER | low | Fingerprint | An attempt to discover the SSID of a hidden AP |
| WELLENREITER | low | Fingerprint | Dictionary based attempt to discover hidden AP's SSID |
| DISASSOCTRAFFIC | medium | Trend | A user is transmitting data shortly after being disassociated (likely an indication that this user is victim of an attack) |
| BSSTIMESTAMP | medium | Trend | An AP's timestamps are out of sequence and thus may indicate a spoofing attempt |
| AIRJACKSSID | high | Fingerprint | An AP is broadcasting an SSID that is the default of the hijacking too Air-Jack |
| DEAUTHFLOOD | high | Trend | Disassociate or deauthenticate are being repeatedly sent (flooded) from a non-AP node |
| CHANCHANGE | high | Trend | An AP is now advertising a different channel than previously detected (probably a man-in-the-middle attack) |
| BCASTDISCON | high | Fingerprint | A disassociate or deauthenticate message has been broadcast. May be used to disclose a hidden SSID, perform a man-in-the-middle attack, or denial service. |
| NOPROBERESP | high | Fingerprint | A response to a probe containing a 0 length SSID, which is an attempt to exploit a bug in some AP firmware |
| MSFBCOMSSID<br>MSFDLINKRATE<br>MSFNETGEARBEACON | high | Fingerprint | A packet crafted to exploit a particular Windows driver fault that allows arbitrary code injection |
| DISCONCODEINVALID<br>DEAUTHCODEINVALID<br>LONGSSID | high | Fingerprint | A packet crafted to exploit a driver or AP firmware fault that might allow arbitrary code injection |

**Table 1: Alert events detected by Kismet and reported to SNBENCH**

**WifiResponder**

In addition to the wireless network sensing described above, we have also implemented a Layer-2 wireless actuator (*i.e.*, output device) WifiResponder that may be used as a retaliatory action against a detected attacker. The WifiResponder invokes a script on a trusted (whitelisted) device running Linux with a compatible 802.11 interface and the airreplay-ng [8] tool. The Opcode `APDeauth()` takes as arguments a WifiResponder that will send a flood of deauthenticate messages to a particular MAC address (the second argument) from a particular MAC address (the third argument).[3] An actuator is nearly identical to a Sensor in its implementation within the SNBENCH. The Handler for WifiResponder invokes the remote common gateway interface (CGI) script to initiate the deauthenticate "attack" against the specified host.

# 5. DEPLOYMENT ENVIRONMENT

Our test-bed deployment contains several OpenWRT[1] Linux enabled Linksys WRT54GL Access Points, each with the kismet-drone, airreplay-ng, and signal strength monitor packages installed. The access points are configured to use their wireless interface in client mode, and are connected to our gigabit research LAN by its on board 100Mbit Ethernet port.

To support the WifiAlertSensor, we run a Kismet drone process on each of the access points while the Kismet Server process runs on the same host as the SXE. Although the Kismet Server process could also be run directly on the AP, the RAM and CPU limitations of these devices lead to a less responsive system than if the Kismet Server process is hosted on a separate host. As the Kismet Server does not presently distinguish the results from different Kismet Drones [13], we run one Kismet server per drone and each WifiAlertSensor connects to a unique Kismet Server process thus allowing SNBENCH to distinguish which drone generated a wireless event. Running one Kismet Server per Drone also carries the advantage of minimizing the impact of a Kismet server process hanging, or failing to process updates from its drones (admittedly a fairly uncommon occurrence).

In our tests of the WifiAlertSensor we were able to simulate and detect all relevant attacks detected by Kismet (Table 1) and were unable to measure any significant induced delay on event detection in the SNBENCH infrastructure. Analysis confirmed the expectation that the amount of time a single Kismet message spent in the Sensor buffer was directly related to the computation load on the SXE host and the alert generation rate. In general the observed buffer service delay oscillated between 0 and 15ms per alert under moderate load with unrealistically high message flooding arrival rates (in practice, Kismet can and will throttle alert notification rates, however this was disabled for our performance tests). Under heavy load conditions with alert message flooding we experienced queuing delay as long as 300ms. This gives us a good indication as to the maximum acceptable workload for an individual SXE before it is no longer a viable host for wireless sensing tasks. Ultimately we believe that any response detection under 1 second is reasonable as it is unlikely that the attacker would, say, flee the premises (or video frame) within that amount of time.

# 6. SERVICE PROGRAMMING PRIMER

To understand the example Wireless Security Services implemented on SNBENCH it is important to understand the key concepts and

[3]Readers may readily note that this opcode is a loaded weapon and may gasp or recoil in horror. In fact, this is not the first Opcode that requires special user privileges to ensure correct use.

unique constructs of SNBENCH programming.[4] The Sensorium Task Execution Plan (STEP) language has a functional-style, high-level sibling called SNAFU (Sensor Network Applications as Functions). SNAFU serves as a readable, accessible language that is compiled into the graph-centric STEP for execution. Broadly speaking, functions in SNAFU correspond to the computational nodes of a STEP graph while terminals represent nodes that convey sensors, actuators, and constant values.

SNAFU provides symbolic assignment and function definition, however it forbids explicit recursion by reference. Instead SNAFU provides iteration constructs called triggers.[5] A trigger takes two arguments, a predicate and a response clause. The predicate is repeatedly evaluated until it evaluates to true, at which point the response clause is evaluated and returned as the result of the trigger expression. For example consider the expression `Trigger(P,Q)` in which `P` is the detection of an AP being compromised and `Q` is an expression that shuts down the AP. The `While-Trigger(P,Q)` is similar to the previous trigger, except that it evaluates `Q` every time `P` evaluates to true and when `P` eventually evaluates to false returns the last value of `Q` (or *NIL* if `P` was initially false).

Persistent triggers extend the basic triggers in that they return a stream of values over their persistent evaluation. A `LevelTrigger` evaluates the predicate `P` indefinitely (or for some specified length of time or conditional termination) and evaluate *and return* a value of `Q` every time `P` evaluates to true. In practice `P` may be the detection of a particular MAC address being used in the network and `Q` is the recording of an image at the detected locale. An `EdgeTrigger` continually evaluates the predicate, but will only evaluate and return the clause `Q` when ever the predicate `P` transitions to be true (*i.e.*, on the edge of the signal `P`). In use, if the expression `P` represents detection of two deauthenticate beacons (indicating the start of a deauthenticate flood) and `Q` is an SMS pager alert, we do not want to generate a separate notification for every consecutive deauthenticate beacon during the duration of the flood.

SNAFU also allows a programmer to refer to an expression by symbolic reference (*e.g.*, `let X = Y in Z` wherein `X` stands for the complete expression `Y` in the expression `Z`) or refer to a computational result by symbolic reference (*e.g.*, `let_const X = Y in Z` wherein `X` stands in for the result of the expression `Y` in the expression `Z`). Finally the trigger construct begs for the creation of a unique reference that allows the symbol to be recomputed once per iteration of the trigger. The "`let_once`" binding (*e.g.*, `let_once X = Y in Z`") provides exactly that facility, ensuring the expression `Y` is evaluated once per iteration of the trigger (`Z`) at the first occurrence of the symbol `X`, while all latter instances of the symbol `X` in the same iteration of `Z` are evaluated by reference to the previous evaluation.

# 7. WIRELESS SECURITY SERVICES

### Simple Detection

An example SNAFU program that provides simple logging is given in Program 1. A `level_trigger` is used to assign an event handler to the detection of a high severity wireless alert. The `storage.-append` Opcode modifies a named storage entity (*i.e.*, table) specified by the first argument by inserting a data object and its corresponding unique key. The storage table is keyed by timestamp and includes entries for each detected violation containing the recorded MAC address, the sensor from which the alert was detected, and the type of alert. Unlike the logging provided by Kismet as an IDS, this service records which sensor has detected the event and is backed by an SQL server. The logged data is available programmatically via storage access Opcodes or direct SQL queries, or through a standard web browser via the SXE host's web service that performs XSL translations to render the local data storage.

This sample SNAFU program could easily be extended to establish a log of all observed wireless activity (not just attacks) by adjusting the predicate of the trigger from `DetectWifiAlert` to `DetectWifiActivity` and removing the severity check. Another simple example service logic is given in Program 2; This program will automatically email an administrator when a specific wireless attack is detected.

### Attack Response

The previous examples are essentially the status quo for a response to the detection of a breach in a Wireless Network – an entry into a log file or an email alert. The advantage of employing the SNBENCH in the wireless security domain is the wider range of responses possible. Nominally, the email operation in Program 2 could be replaced with any number of response mechanisms including sending an explicit deauthorization to the detected MAC address[6] using the WifiResponder and APDeauth opcode described in Section 4. Instead we explore the unique cross section of the network plane (*e.g.*, wireless data frames) with the physical plane (*e.g.*, signal strength and signal loss of signal over distances). For example, an embedded, cross-modal Sensor Network such as the Sensorium can utilize both wireless network sensors (*i.e.*network plane sensors) and a pan-tilt-zoom video camera network (*i.e.*physical plane sensors) to catch an image of the attacker "in the act."

### Logging Physical Evidence

Any user detected engaged in wireless network intrusion is clearly within a bounded distance from the detecting sensor. This coarse, cell of origin based physical location of wireless users is available, imprinted in all wireless data returned from the WifiSensors (determined by which sensor has detected the user). A very simple wireless cell of origin location example is specified in Program 3. The program's content is very similar to the previous examples and introduces some Pan-Tilt-Zoom sensor (PTZCamera) specific opcodes, the function of which should be clear from context. This sample streams images of a region where an attack has been detected. The location logic is explicit in the service logic, selecting an image from the camera that best covers the physical space within the signal coverage of the relevant Wifi sensor, using "hard coded" location logic that is specific to the sensor configuration of the particular deployment. The `case` expression is used for readability as syntactic sugar (shorthand) for nested conditionals and takes the same syntax as in StandardML. Connecting this program fragment to either of the previous examples would enable the logging or emailing of images that correspond to the attack location.

Alternatively, the reconstruction of user location could be implemented within an Opcode resulting in intrusion detection service logic that is agnostic to the particular location resolution mechanism used. Such an approach makes sense if the deployment environment already contains a wireless location infrastructure (*e.g.*,

[4]We refer the reader to [18] for a more thorough treatment of the SNAFU language and its evaluation.

[5]Tail-recursion can be emulated with triggers and the token "`LAST_TRIGGER_EVALUATION`" that refers to the previous evaluation of the trigger within the body of the predicate or response clause.

[6]A MAC address is far from the best way to uniquely identify an attacker as the attacker will likely use a fictitious MAC address or worse, clone a legitimate user's MAC during an attack.

**SNAFU Program 1** Adds an entry to a central log on detection of a wireless alert.

```
let_once ALERT = DetectWifiAlert(sensor(WifiAlert,"ALL")) in
    let_once KEY = concat(ALERT.TIMESTAMP,ALERT.SOURCE) in
        level_trigger(
            equals(ALERT.SEVERITY,"HIGH"),
            storage.append("ALERTLOG",KEY,ALERT)
        )
```

**SNAFU Program 2** E-mail an administrator whenever a specific wireless alert is detected.

```
let_once ALERT = DetectWifiAlert(sensor(WifiAlert,"ALL")) in
    level_trigger(
        equals(ALERT.TYPE,"DEAUTHFLOOD"),
            email("mocean@cs.bu.edu",
                concat($NOW$,
                    ": Deauth flood detected from MAC ", ALERT.MAC,
                    " at time ", ALERT.TIMESTAMP,
                    " by sensor ", ALERT.SOURCE
                    )
                )
            )
```

[20], [9], all knowing oracle) that could be accessed from within an Opcode call. An example of this approach is given in Program 4. `WifiLocateMac` encapsulates the physical location of MAC addresses and (`PTZLocate`) determines the best PTZ Camera (and corresponding angle) to capture an image of that location. The implementation of `WifiLocateMac` is functionally similar to `BestPTZForViewOf` in the example in Program 3, yet uses a received signal strength from multiple sensors to estimate the target's location between the sensors.

Recall the SNBENCH not only eases the composition of such alert services, it also eases deployment by automated re-use of existing computation/deployments to improve resource utility. All the examples given thus far share the same predicate logic and could share a single instantiation of that portion of the logic.

# 8. FUTURE WORK

As briefly touched on in the previous Section 7, computational complexity can either be placed explicitly in the service logic or pushed into the Opcodes on which the service relies. Naturally, there is a resource flexibility and performance trade-off in this decision. In the most degenerate case any program can be implemented as one giant Opcode that eliminates any performance penalties incurred by the STEP interpreter. Such an approach yields a complex Opcode implementation that eliminates SNBENCH's inherent resource management benefits (*i.e.*, computation and resource sharing becomes unlikely as there will be few common subgraphs between STEP programs, individual opcodes can not be split across multiple SXEs, and there may be few SXEs available to accommodate such a large computation). On the other extreme a program composed entirely of very basic STEP Opcodes may pay a high overhead for the SXE interpreter, yet has maximum flexibility to be split across any compatible partitioning of the SXE space. Finding the optimal balance between STEP and Opcode complexity largely depends on the particular needs of the given service.

### Advanced Location Reconstruction

Program 5 tries to find a balance between the extremes of the last two examples in the prior section. This program also produces an image whenever an attack is detected, but specifies much of the location logic within the STEP program while still leaving the significant computation to Opcode implementations (*e.g.*, `rssiToDist`, `PTZFovCover`). The function `Fold`, commonly referred to as reduce in some languages, is a higher-order function that applies a given function accumulatively across a given list. In this application, the function `BetterView` is applied across the list of all Pan-Tilt-Zoom sensors to determine which sensor has the best coverage of the area in which an attack was detected in order to take a snapshot from that sensor. In compilation Fold is syntactic sugar that is expanded by the compiler via substitution; its implementation is an expression that uses a `WhileTrigger`, the `LAST_TRIGGER_EVAL` token, lists and pairs to provide iterative application over the list of elements.

### Wireless Access Lists from Physical Data

The last example program (Program 6) also leverages the natural connection between wireless network security and physical site surveillance, however does so in the other direction, using information detected on the physical plane to (re-)configure the wireless network. An embedded camera network and face detection Opcodes are used to detect the identities of individuals entering or leaving a secured space as a trigger to enable the detected user's wireless MAC address for service in that physical area. Put simply, when we see Jane enter the lab we want to allow Jane's MAC address to be used in the lab (placed in the whitelist), and we want to remove her MAC address from the whitelist when she leaves the lab. The goal is to make it slightly more difficult for a malicious user to find an unused, authorized wireless MAC address to abuse for great lengths of time. Modification of the WLAN's access control list in this example is performed by assuming the presence of a `WifiWhitelist`; This implementation would be straightforward on OpenWRT enabled Access Points, requiring a CGI script to modify the device's maclist configuration file. One may easily imagine other sensors that may be used in tandem with face detection as the trigger predicate in this expression, *e.g.*, a magnetic ID card or RFID reader.

### SNBENCH as a Turn-Key Network Security Solution

There is no reason to limit SNBENCH's Network Security to Wireless attacks. Other Intrusion Detection Tools and Network interfaces could easily be added to further improve the SNBENCH as a

**SNAFU Program 3** Whenever a wireless alert is detected, pan a PTZ camera to that region and return its image.

```
def BestPTZForViewOf(alert) =
  case APName(alert.SOURCE) of
      ''CS Grad Lab West'' => List(45,0,0,sensor(PTZCamera,"PTZ1"),
    | ''CS Grad Lab East'' => List(15,0,0,sensor(PTZCamera,"PTZ1"),
    | ''CS Grad Lab Lounge'' => List(0,0,0,sensor(PTZCamera,"PTZ3"),
    | ''CS UGrad Lab'' => List(0,0,0,sensor(PTZCamera,"PTZ4")

let_each ALERTSENSORS = sensor(WifiAlert,"ALL") in
  let_once ALERT = DetectWifiAlert(ALERTSENSORS) in
    level_trigger( not(isNull(ALERT)),
      PTZSnapshot(BestPTZForViewOf(ALERT))
    )
```

**SNAFU Program 4** Functionally equivalent to Program 3, but uses "black-box" opcodes.

```
let_each ACTSENSORS = sensor(WifiActivity,"ALL") in
   let_each PTZSENSORS = sensor(PTZCamera,"ALL") in
      let_once ALERT = DetectWifiAlert(sensor(WifiAlert,"ALL")) in
         level_trigger( not(isNull(ALERT)),
            PTZSnapshot(PTZLocate(QueryWifiAlert(alert.MAC,ACTSENSORS)),PTZSENSORS))
         )
```

**SNAFU Program 5** Whenever a wireless attack is detected return an image from the camera that has the best coverage of the region containing the attack.

```
def BetterView(T,X,Y) =
  let L = Pair(locate(T.SOURCE),rssiToDist(T.RSSI))
    in cond(greater(PTZFovCover(X,L),PTZFovCover(Y,L)), X, Y)

let_once ALERT = DetectWifiAlert(sensor(WifiAlert,"ALL")) in
  let_once ALERTINFO = QueryWifiActivity(WifiActivity,ALERT.SOURCE) in
     level_trigger(
        not(or(isNull(ALERT),isNull(ALERTINFO))),
           PTZSnapshot(
              Fold(
                BetterView(ALERTINFO,-),
                sensor(PTZCamera,"ALL")
              )
           )
     )
```

**SNAFU Program 6** Use the detection of a user's face to enable their associated MAC address for use on the AP.

```
let_once SNAP = snapshot(sensor(Camera,"Lab door in","Lab door out")) in
   let_once WLAN_MAC_ADDR = storage.lookup("MACMAP",FaceDetect(SNAP)) in
      level_trigger( not(isNull(WLAN_MAC_ADDR)),
         case SNAP.SOURCE of
            "Lab door in" =>  WifiWhitelist(WLAN_MAC_ADDR, sensor(AP,"CS Lab*")),
          | "Lab door out" => WifiBlacklist(WLAN_MAC_ADDR, sensor(AP,"CS Lab*"))
      )
```

complete, cross-layer Network Intrusion Detection System. Integrating Layer-3 detection (*e.g.*, Snort) as a sensor would enable the detection of misuse from IP contents that could be used to drive isolation or removal responses at Layer-2. Additionally including port scanning and other fingerprinting tools would greatly increase confidence in user identification enabling more confident automated response.

In our vision of SNBENCH we view the Sensor Task Execution Plan (STEP) as a thin-waist language that may be a compilation target from other domain specific languages (*e.g.*, a Structured Query Language). Ideally we would like to move our own campus IT department to the use of SNBENCH over their current Network Security and Intrusion tools. As such we might consider the development of a STEP compiler for a declarative/rule-oriented language that is similar to existing network rule specification languages to ease the staff's transition to SNBENCH.

Related to this goal, our work on a lightweight SXE for embedded devices will be used to explore deployment of the Sensor eXecution Environment directly on OpenWRT enabled access points to provide SNBENCH as a turn-key solution for Wireless Network Security services.

## 9. CONCLUSIONS

Many excellent Wireless Intrusion Detection tools exist and they achieve their specialized goals well. These tools should not (and rarely attempt to) provide complete Network Security Systems as they are generally focused on detecting a particular kind of attack (denial-of-service, intrusion, *etc*) at a particular scope (Layer-2 or Layer-3). A complete Network Security solution should integrate multiple tools to cover a superset of possible attacks at all possible layers and should bridge the divide between cyber and physical identities. In practice this allows NID tools to focus on what they do best (*i.e.*, detection) and yet be woven into a comprehensive Network Security solution. Network Security (specifically, wireless security) is not a problem that exists in a vacuum detached from the physical space in which the network is deployed. We promote an approach to unify physical site surveillance and network security security under the umbrella of SNBENCH — a general purpose sensing infrastructure we have developed. In that regard, we have demonstrated how SNBENCH enables the rapid development and deployment of cross-modal security services. We have shown that with SNBENCH (1) detection of wireless anomalies can be correlated with other sensory inputs providing reciprocal benefit to merging security on the physical and cyber planes, (2) detection and response services may be easily composed and modified without technical knowledge of the specific protocols or implementations of the underlying sensory tools, and (3) adding additional intrusion detection tools as input or other devices for response is straightforward given SNBENCH's modular architecture. The illustrative example programs provided in this paper range from the status quo (simple logging and email alerts) to beyond (enabling MAC addresses based on face detection) in a hope to spark the reader's imagination to more elaborate services and responses that are currently possible with the SNBENCH platform (*e.g.*, locking doors, turning on a siren, ...).

## 10. REFERENCES


[1] *OpenWRT Project Homepage*, http://openwrt.org/.

[2] Frank Adelstein, Prasanth Alla, Rob Joyce, and Golden G. Richard III, *Physically Locating Wireless Intruders*, ITCC '04: Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC'04) Volume 2 (Washington, DC, USA), IEEE Computer Society, 2004, p. 482.

[3] AirDefense, Inc., *AirDefense Enterprise Product Homepage*, http://www.airdefense.net/products/enterprise.php.

[4] Paramvir Bahl and Venkata N. Padmanabhan, *RADAR: An In-Building RF-Based User Location and Tracking System*, INFOCOM (2), 2000, pp. 775–784.

[5] John Bellardo and Stefan Savage, *802.11 Denial-of-Service Attacks: Real Vulnerabilities and Practical Solutions*, SSYM'03: Proceedings of the 12th Conference on USENIX Security Symposium (Berkeley, CA, USA), USENIX Association, 2003, pp. 2–2.

[6] Azer Bestavros, Adam Bradley, Assaf Kfoury, and Michael Ocean, *SNBENCH: A Development and Run-Time Platform for Rapid Deployment of Sensor Network Applications*, IEEE International Workshop on Broadband Advanced Sensor Networks (Basenets), October, 2005.

[7] Boston University, Department of Computer Science, *Sensorium Research Homepage*, http://www.cs.bu.edu/groups/sensorium/.

[8] Christophe Devine, *Aircrack-ng Homepage*, http://www.aircrack-ng.org/.

[9] Ekahau, Inc., *Ekahau Positioning Engine 4.0 Product Homepage*, http://www.ekahau.com/products/positioningengine/.

[10] Jamil Farshchi, *Wireless Intrusion Detection Systems*, http://www.securityfocus.com/infocus/1742, 2003-11-05.

[11] IBM Internet Security Systems, *Wireless Products Homepage*, http://www.iss.net /documents/whitepapers/wireless_LAN_security.pdf.

[12] James Goddard Joshua Lackey, Andrew Roths, *Wireless Intrusion Detection*, http://www-935.ibm.com/services/-us/bcrs/pdf/wp_wireless-intrusion-detection.pdf, 2003.

[13] Mike Kershaw, *Kismet User Forum*, http://www.kismetwireless.net/Forum/General/Messages/-1142522037.4893529.

[14] ———, *Kismet (version 2007-01-r1b)*, http://www.kismetwireless.net/documentation.shtml.

[15] Andrew Lockhart, *Snort-wireless Homepage*, http://snort-wireless.org/.

[16] "loud-fat bloke", *WIDZ (Wireless Intrusion Detection System) Homepage*, http://freshmeat.net/projects/widz/.

[17] Michael Lynn, *AirIDS Project Homepage*, http://airids.sourceforge.net/.

[18] Michael J. Ocean, Azer Bestavros, and Assaf J. Kfoury, *SNBENCH: Programming and Virtualization Framework for Distributed Multitasking Sensor Networks*, VEE '06: Proceedings of the 2nd International Conference on Virtual Execution Environments (New York, NY, USA), ACM Press, 2006, pp. 89–99.

[19] Martin Roesch, *Snort - Lightweight Intrusion Detection for Networks*, LISA '99: Proceedings of the 13th USENIX Conference on System Administration (Berkeley, CA, USA), USENIX Association, 1999, pp. 229–238.

[20] Cisco Systems, *Wi-Fi Based Real-Time Location Tracking: Solutions and Technology*, http://www.cisco.com/application/pdf/en/us/guest/products/-ps6386/c1244/cdccont_0900aecd80477957.pdf, 2006.

[21] P. Tao, A. Rudys, A. Ladd, and D. Wallach, *Wireless LAN Location Sensing for Security Application*, 2003.

[22] Nathan True, *Wi-viz: Wireless Network Environment Visualization*, http://devices.natetrue.com/wiviz/.

[23] Giovanni Vigna, Fredrik Valeur, and Richard A. Kemmerer, *Designing and Implementing a Family of Intrusion Detection Systems*, SIGSOFT Softw. Eng. Notes **28** (2003), no. 5, 88–97.

[24] Yoann Vandoorselaere, et. el., *Prelude Hybrid IDS*, http://www.prelude-ids.org/.

[25] Moustafa Youssef, Ashok Agrawala, and Udaya Shankar, *WLAN Location Determination via Clustering and Probability Distributions*, March 2003.