JOURNAL OF UNMANNED AERIAL SYSTEMS

PEER-REVIEWED ARTICLE

CYBER ATTACKS AND DEFENSE FRAMEWORK FOR UNMANNED AERIAL SYSTEMS (UAS) ENVIRONMENT

Kevin Casagrande
Joshua Friederichs
Clarissa Gonzalez
Tanya Humphries
Zachary Tindall
Undergraduate Students, Electrical Engineering, University of North Dakota

Roger French
Principal Engineer, Rockwell Collins Corporation, Cedar Rapids, Iowa

Dr. Prakash Ranganathan
Assistant Professor, Electrical Engineering, University of North Dakota

ABSTRACT

The proliferation of privately-owned Unmanned Aerial Vehicles (UAVs) and the associated potential risks they pose to personal privacy and safe navigation has highlighted the need for an effective system to test UAV cyber security vulnerabilities and develop effective countermeasures. Towards those ends, we present a novel, customizable, and a universal UAV simulation environment utilizing an open-source Linux-based virtual machines (VM) for modeling cyber-attacks and defenses against ArudoPilot-MavLink-based fixed-wing and rotary-wing UAVs. To demonstrate the effectiveness of this simulation environment, the percentage of successful man-in-the-middle and re-broadcasting attacks conducted by a Kali Linux VM's against a stock ArduPilot UAV were examined. Additionally, the simulation environment was used to develop and test a unique defensive system based on an external SNORT intrusion detection system (IDS) to monitor communications between the simulated UAV and ground control system (GCS). Upon intrusion detection, this system automatically executed pre-planned countermeasures to defeat potential threats. The effectiveness of this system was evaluated based on the reduction of successful cyber-attacks and vulnerabilities and developing effective countermeasures without the time and cost required for physical prototyping using actual UAV assets was effectively demonstrated.

I. Introduction

With the rise in commercial and military use of unmanned aerial vehicles (UAVs or "Drones"), a lot of research has been conducted on the potential vulnerabilities inherent in autonomous technologies which operate geographically detached from human operators (Paganini, 2013). Most notably, Todd Humphreys and his team at the University of Texas have made considerable strides in both GPS spoofing and detection or erroneous GPS signals and the application of these technologies to UAVs (K, 2015).

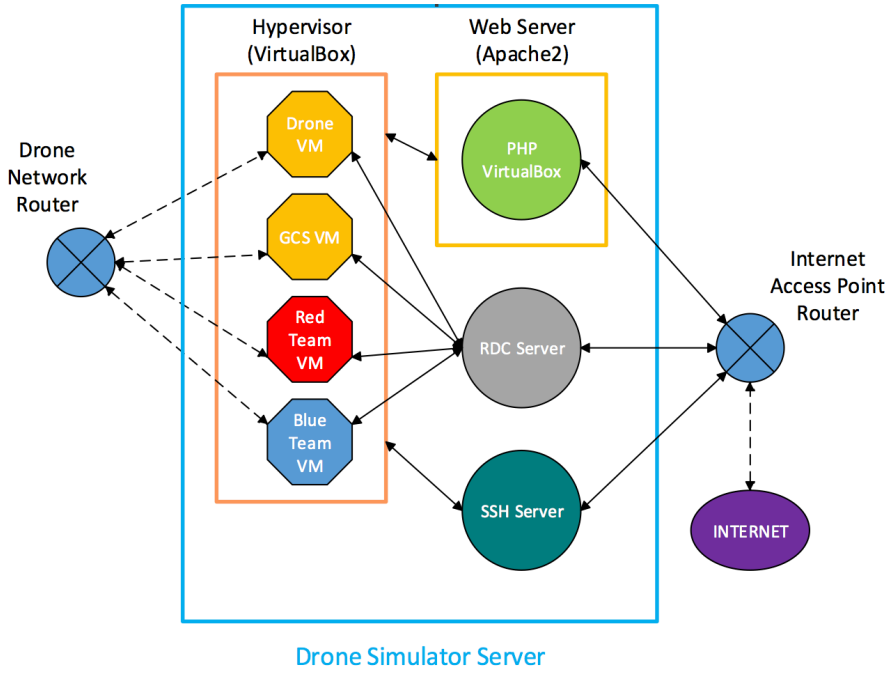

Fig. 1 – System Diagram of Drone Cyber Attack and Defend Simulator

Javaid et al in (Javaid, Sun, Devabhaktuni, & Alam, 2012) published a comprehensive analysis of vulnerabilities in drone design and the potential impacts on vehicle operation. However, with the commercial applications of drone technology beginning to dwarf the military applications, the likely targets of cyber-attacks are not large, military grade, hardened UAVs but rather the small, cheaply produced, and easily operated drones favored by commercial enterprise. Cyber security specialists and hacker collectives alike have devoted extensive analysis resources towards this area, as evident by frequent presentations at Def Con and similar conferences. In addition, some academic teams have started performing research on this topic. Hal Aldridge and a team from Purdue University developed a simulator and modelled the behavior of a UAV exposed to several forms of cyber-attacks (Kim, Wampler, Goppert, Hwang, & Aldridge, 2012). They proposed developing a method to detect and counter those attacks.

II. System Description

The project focuses on the development of open-source systems conforming to the specifications of the Dronecode project (www.dronecode.org) and ArduPilot (http://ardupilot.com/) to develop a VM-based drone simulation suitable for testing a wide array of cyber security tools and techniques. A basic block diagram of the system is provided in Fig 1. The system consists of a virtualization server which hosts the 4 Virtual Machines (VMs). These are the Drone VM, the Ground Control Station (GCS) VM, an attack VM and a defense VM. The main element of the system is a virtualization server composed of an Ubuntu 12.04 headless server running Oracle VirtualBox. A web interface for control and operation of the virtualization server is accomplished using PHPVirtualBox hosted through an Apache2 web server installed on the virtualization server base operating system.

The first element is the Drone VM, which is an Ubuntu 14.04 based host for the ArduPilot drone operating system. The ArduPilot code is fully compliant with the DroneCode standard and is the actual software which would run on physical drones conforming to this standard. The Drone VM includes both the fixed-wing and helicopter variants of this software. Additionally, the Drone VM is equipped with the JSBSim flight simulator software, which is used to provide the flight dynamics for the drone simulation. Finally, the Drone VM includes the MAVProxy software, which it uses to provide a UDP-based communication circuit for the drone. An overview of the drone VM can be seen in Fig. 2. Using these three elements in tandem allows for the accurate simulation of a wide-variety of fixed-wing and rotary type drones in user-programmable environmental conditions. Although the ArduPilot software would normally be constrained by the physical design of the drone it was running on, these parameters are user-programmable in the simulated environment. Thus any collection of drone parameters the user wishes to use can be simulated. This set-up also allows for follow-on efforts for hardware-in-the-loop testing utilizing a physical drone controller and actual live-testing of a complete drone system.

Simulation of the drone controller or operator is accomplished through the GCS VM. Like the Drone VM, the GCS VM is based on Ubuntu 14.04 and hosts the MAVProxy software. In the case of the GCS, however, the MAVProxy software is used as a ground control system which communicates with the ArduPilot software on the Drone VM via either Ethernet or Wi-Fi communications channels. The user inputs commands to the drone on the GCS VM, which are then communicated to the Drone over the network connection and the ArduPilot software executes these commands. The feedback to the ArduPilot is provided by the JSBSim software, which simulates the drone behavior. Through this communications link, the drone position, speed, and other characteristics are fed back to the GCS VM. This is in effect the same operation as if the operator were communicating remotely with an actual drone. The design of the system allows for any compatible ground control software and/or physical controller to be used to control the simulated drone.

To demonstrate the validity of this system, and for identification and correction of potential cyber vulnerabilities, this project includes additional VMs to demonstrate the use of offensive and defensive techniques to test and improve the simulated drone system. Currently two defensive VMs are included on the virtualization server: one based on Ubuntu 14.04 desktop and a second utilizing the SecurityOnion operating system. These VMs operate almost identically to a physical machine operating on the same network as the drone and ground control station. For testing offensive cyber tools, a fifth VM is currently installed which is a Kali Linux 2.0 operating system. Since these VMs operate on the same drone network as the Drone VM and GCS VM, and since the packets are physically routed from these VMs to a common router via separate adapters, the defensive and offensive VMs are able to interact with the drone and GCS identically as if they were physical machines. This means intrusion detection scans will trigger just as they would in an actual operating environment. Likewise, packet analysis and spoofing attacks will behave just as they would had they been conducted on an operational network.

III. Red And Blue Team Responsibilities

To accomplish the goals of this project, members were divided between three main task groups. Kevin Casagrande is responsible for the design, creation, and maintenance of the simulator environment. Kevin has created the virtualization server and web host as described above. Other responsibilities focus on improving this system and incorporating the hardware elements and physical capabilities required to accurately simulate the offensive and defensive cyber security capabilities utilized by the other groups.

The defensive team (Blue Team) is composed of Tanya Humphries and Joshua Friederichs, is focused on defensive cyber security operations. This group is responsible for testing and implementing tools to prevent the control or destruction of the drone by malicious actors. Thus far, Tanya has focused on Snort and determining how this intrusion detection package can be leverage to identify unauthorized access to the drone and/or GCS and trigger defensive countermeasures. Snort is a sniffer program which captures network traffic and then analyzes the data (S, 2003).

Josh has focused on the Security Onion suite to determine which attacks it can protect against and if it could be suitably tailored to project. Together, Tanya and Josh will decide upon the best configuration for the defensive VM and integrate them into the simulated environment. A summary of the defensive tools is shown in Fig 3.

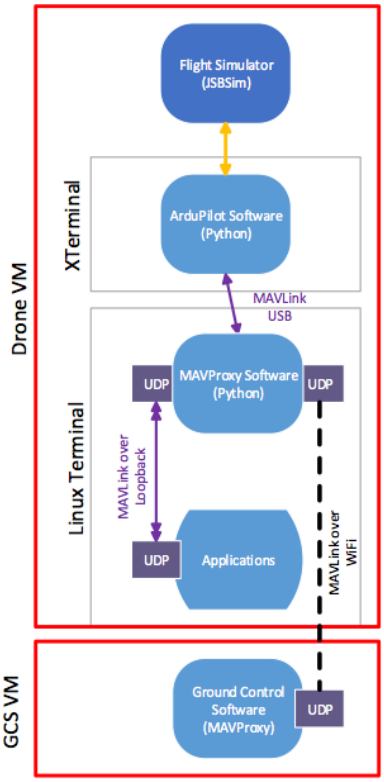

Fig. 2 – Drone Simulator VM Block Diagram

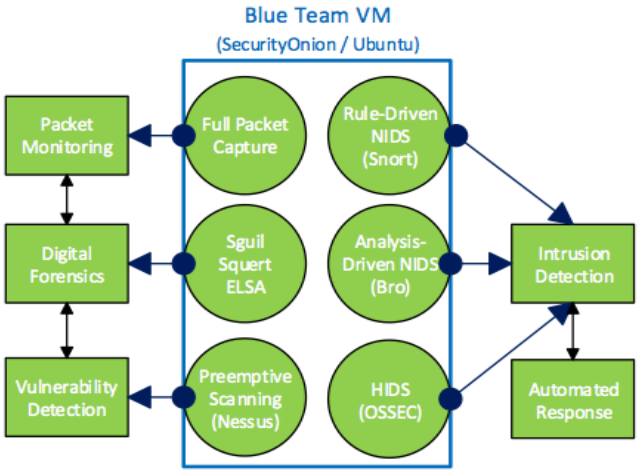

Fig. 3 – Blue Team Block Diagram

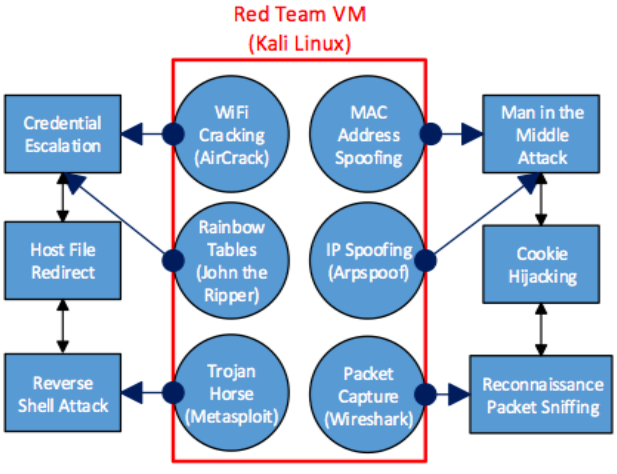

Fig.4. Red Team VM Block diagram

The Blue Team has been able to create a VM to serve as a defensive box that can detect network attacks and defeat them in an automated mode. Open source tools were used to accomplish this to allow for configuration and editing of these tools, as well as economic considerations. The tools making up the defensive VM include Snort, Kismet, Swatchdog, and SendIP.

The attack team (Red Team) is composed of Clarissa Gonzalez and Zachary Tindall. They are focused on offensive cyber security operations and will conduct attacks against the drone and/or GCS with the goal of seizing control or destroying the drone. Simply preventing the drone from executing its tasks will not be considered a successful attack for this project. Both Zach and Clarissa have thus far worked with Kali Linux, as this is likely the most suitable attack platform. Zach has been working on Wi-Fi attacks such as those presented in (Buchanan & Ramachandran, 2015) and (Lakhani & Muniz, 2013), including those aimed at severing the established link between the drone and the GCS and defeating WPA encryption. Clarissa has been working on IP and MAC address spoofing. Through the combination of these efforts, they plan to devise a mechanism for inserting the attack computer in the middle of the communications link between the drone and the GCS without alerting the operator or the Blue Team. Ultimately, Clarissa and Zach will determine the configuration and attack profile most likely to yield success and integrate them into the attack VM for testing. An overview of the tools being used by the Red Team can be seen in Fig. 4. So far the Red Team is executing basic Wi-Fi attacks using airmon-ng, a deauth message, wireshark, and aircrack-ng. The process for this type of attack is shown in Fig. 5.

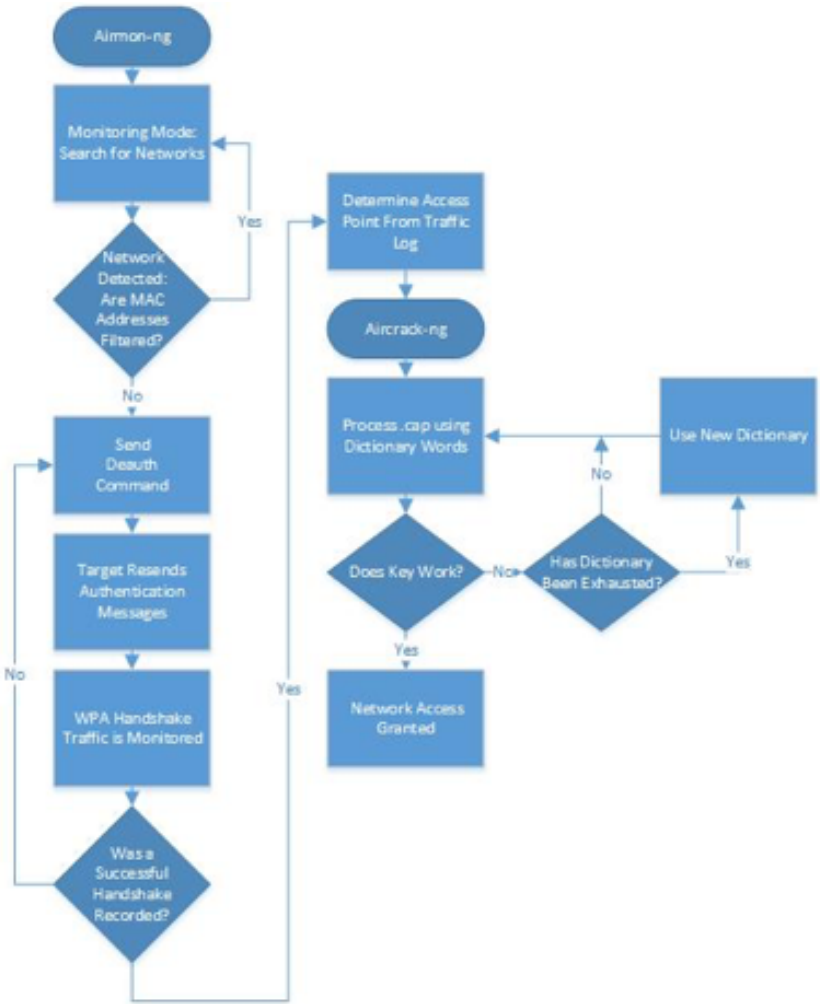

Fig. 5. Flowchart for Wi-Fi Attack

Once both the Red and Blue teams are ready, the project team will test the effectiveness of the defensive configuration against the offensive attacks. From the results of this testing, the group will attempt to improve the robustness of the drone's defenses and identify the likely threat vectors.

IV. Project Tasks

The following tasks have been identified and accomplished for this project:

1. Created a cloud-based VM
2. Developed a web-based interface for an Ardupilot to simulate drone behavior.
3. Ported the drone simulation to a physical device.
4. Developed an ad-hock 2.4 GHz Wi-Fi network interface between the controller VM, Penetration VM, and drone simulation.
5. Tested the effectiveness of attacks on the Wi-Fi connection developed by the Red Team and defenses of the Wi-Fi connection developed by the Blue Team.
6. Conducted the literature review of techniques for attacking Wi-Fi connections, Linux operating systems, controller computers, and drone control software
7. Selected offensive tools and develop familiarity and skill in their use.
8. Developed the attack box (VM or computer) and install required tools.
9. Tested the success of identified attacks, scans, etc. against the cloud-based simulation.
10. Identified Wi-Fi attacks to be tested against the physical Wi-Fi connection.
11. Researched techniques for intrusion detection and prevention for Wi-Fi connections, Linux operating systems, controller computers, and drone control software.
12. Selected defensive tools and develop familiarity and skill in their use.
13. Developed the defense box (VM or computer) and install required tools.
14. Evaluated the effectiveness of defensive tools against the red team attacks.

V. SIMULATION ENVIRONMENT AND SET-UP

The initial part of the project was to design an actual simulation environment. The ultimate end product had to be accessible remotely and capable of hosting simultaneous access to multiple participants. Initially the use of commercial cloud services, such as Digital Ocean and Amazon Web Services, were considered as the primary deployment vehicle, mostly due to the commercial grade connectivity these services provide. However, as research into the requirements for the simulation itself progressed, it rapidly became apparent that the dedicated physical memory and processing required would make any commercial cloud-based service cost prohibitive for use in this project. Ultimately it was decided dedicated server hardware would be required. Privately-owned hardware was repurposed to provide the necessary physical back-bone for the virtualization environment. The end result of this effort was the server construct described above. Physical construction required 36 hours of assembly and configuration.

As commercial web-hosting was not feasible, a dedicated web-interface needed to be developed for hosting remove access to the hypervisor to allow group members to remotely create, start, and stop VMs as necessary for work on the project. As cost was a factor, open source options were favored. Towards that end, a headless Ubuntu 12.04 server was selected as the hypervisor operating system. On top of that OS, Oracle VirtualBox was compiled from source code for use as the hypervisor. Additionally, Apache2 Web Server was selected as the web interface host. Squid proxy server was installed to allow for cacheing of internet data which has the effect of speeding up updates to the multiple VMs by archiving frequently accessed material. To facilitate remove access of VirtualBox services, without having to resort to the command line, PHPVirtualBox was installed and configured as a script-based interface operating on the Apache2 web server. The set-up, installation, testing, and debugging of this set-up required and

additional forty-eight hours of dedicated development time.

Once the hosting environment was completed, development of the actual VMs for use in this environment began. To streamline operation of the VMs, each was initially prototyped on a separate physical computer. Once the configuration of each was finalized, rather than transport each VM over to the hosted environment, the VMs were re-created within the virtualization server environment. This ensured a more polished and streamlined deployment of each VM within the server environment, as the baggage and overhead from prototype development and tweaking was not transported over into the new environment.

Development of the Drone VM was particularly challenging, as the overall intent of creating a realistic system analogous to a physical drone was of primary importance. Various attempts at hosting a virtualized ARM-based machine within the VirtualBox environment were attempted with little success.Ultimately the conclusion was reached that in order to accommodate the processing power and virtual memory required to run the flight simulator software effectively, a full Ubuntu implementation would be required. In effect, this artificially increased the processing power and virtual memory allocated to the drone well beyond that typical for a physical model, as both the simulator software and drone operating system had to be installed on the same VM to allow flight dynamic data to be processed without excessive delay. The second major challenge in the development of the Drone VM was in the integration of the ArduPilot software and the JSBSim flight simulation program. Organically, JSBSim would function adequately with the ArduPlane software, but flight dynamics from JSBSim would not act compatibly with ArduCopter operating systems. As any extension of the simulation project to a physical drone would likely involve a multi-copter drone vice a fixed-wing UAV, and as the parameters of the project called for accurately simulating a significant range of potential drone designs, this was deemed to be too limiting of a condition to be considered acceptable. Custom scripts were programmed in order to allow for flight dynamic data from JSBSim to interact with ArduCopter software. While functional, this set-up is still significantly limiting. Therefore, alternative flight simulation systems were examined throughout the duration of this project in parallel with continued development of the primary system using JSBSim. None of these systems improved performance sufficiently to justify the cost of a commercial product.

Additional VMs for dedicated use as ground control, defensive cyber security testing, and offensive cyber security testing were also created as described in other sections of this report. While MAVProxy over MAVLink is the primary ground control software design being used in this project, a number of alternative configurations were also examined. It was determined, at a minimum, that QGroundControl and APM Planner interfaced effectively with the Drone VM, even when operating on Linux and Windows 7 VMs respectively. Some minor testing was conducted using iOS and Android based ground control applications with favorable results. As these are more suited for Hardware-in-the-Loop (HITL) applications, rather than the limited value they present in a virtualized Android development environment, little work was done beyond demonstration of theory-to-concept.

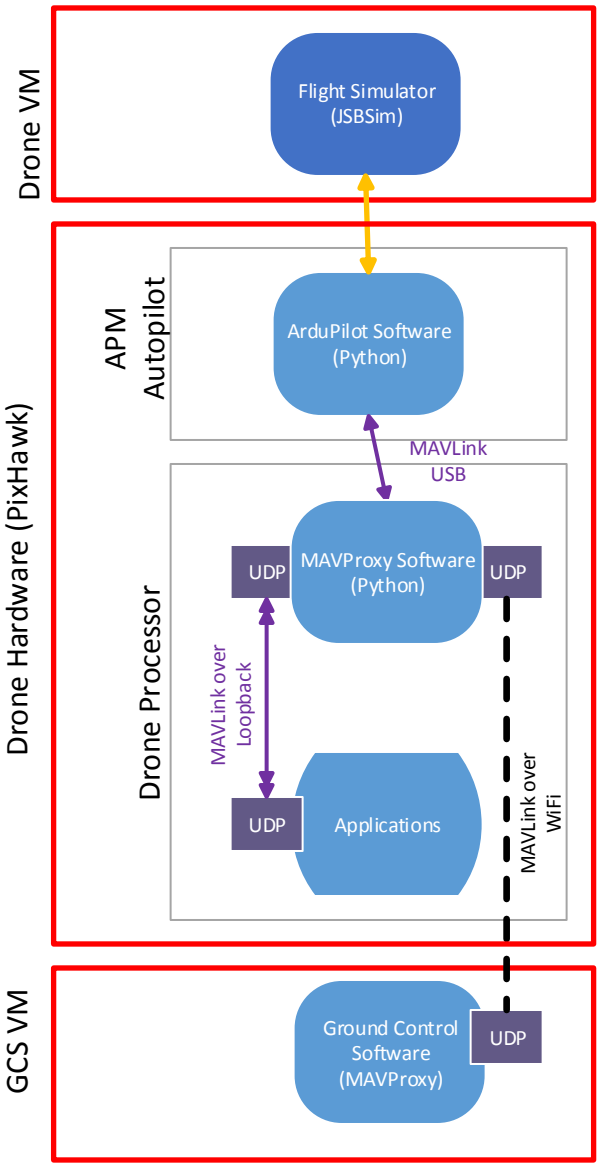

Fig. 6 – Hardware-in-the-Loop (HITL) Block Diagram

For sake of simplicity and standardization, the project has assumed MAVProxy to be the ground control system for testing purposes. This eliminated the need for group members to familiarize themselves with multiple ground control systems.

a) Remote Hosting.

As remote hosting of the environment had to be performed over a residential—vice commercial—internet service, there were a number of challenges in preventing the Internet Service Providers filtering of web traffic. As the type of dedicated traffic to a residential connection tends to trigger ISP anti-spam and anti-Denial-of-Service algorithms, a mechanism for authenticating traffic from group members was required. This was initially accomplished through a very basic PPTP virtual private network setup. However, as this was a crude solution, an improved connection was established using an alias on the Fully Qualified Domain Name (FQDM) associated with this project to forward standard HTTP traffic to a non-standard port number. Initially the interface software for remote access included in

PHPVirtualBox was used to access and control VMs directly. However, this method resulted in an unacceptably high failure of connectivity and furthermore required hard stops of the VMs to resolve the loss of synchronization each time, posing a high risk for corruption of the VMs themselves. Dedicated ports for Microsoft Remote Desktop Connect protocol access were established, using the RDP server services inherent in VirtualBox vice those in PHPVirtualBox. This provided for a very stable connection to the VMs, but still suffered from the lag due to the limitations of the residential internet connection.

An OpenVPN server operating on the virtualization server external router was attempted as a solution, but performance problems plagued this configuration during testing. A dedicated OpenVPN server VM was developed to overcome these limitations. While this configuration was successful for a period of time, ultimately the OpenVPN server VM failed to maintain a connection from outside the domain upon which it was hosted. Significant troubleshooting was successful at resolving these problems. As an attempted solution, the OpenVPN server was rebuilt from scratch several times, with careful attention on kernel configuration during each attempt. In each of these cases, the OpenVPN server functioned to provide a link to the internal drone network from external internet domains, but more advanced applications (such as remote VMs interacting on the network) were extremely limited. This was especially true when more than two remote machines attempted to connect to the network over the VPN connection simultaneously; under these conditions, the connection held but was all but useless from a practical standpoint. Ultimately it was concluded the combined limitations of the open source version of the OpenVPN software and the service provider limitiations placed on a residential internet connection were too great to be overcome by tweaking the OpenVPN configurations.

Given the time delays imposed by remote connections to the virtualization server and hosted VMs, any future iterations of this project will need to consider commercial grade internet hosting (via a university or other entity capable of providing such services), restricting access to a more local geographic area in the proximity of the server, or shifting to an entirely contained local network (i.e. eliminating remote access via the internet) in order to overcome these limitations.

As a potential work-around for the lack of an effective VPN system, archives of VMs were exported from the server into standard OVA files and distributed for individual use by group members. This way, it was believed, group members could work on their local hardware using these VMs, work on their individual segment, and then upload their work for incorporation on the virtualization server for incorporation into the larger project environment. It was hoped this would yield productivity advantages, as the internet lag from remote operation of the VMs would be limited to testing in the full environment. Towards that ends, automated scripts were developed which would archive VMs after significant changes were identified in those VMs. Archiving was accomplished by exporting the VMs to all-inclusive OVA files. A second set of automated scrips automatically synchronized with a dedicated DropBox folder, uploading revisions to the OVA files automatically. As group members had common access to this DropBox folder, they could then download the OVA files as needed for use on their local machines. This also provided a mechanism for establishing regular back-ups for the VMs, should any part of their testing or use cause irrevocable harm or corruption to them.

Extensive component and system testing was required to bring the system as described above to its ultimate state. The initial challenge was establishing the virtual drone construct, as the ArduPilot software is not designed to run on a standard operating system, let alone inside a virtual machine. Once a viable solution was found for these challenges, the resulting drone simulation had to respond appropriately to local commands. Testing therefore consisted of simulating both a quad copter and a fixed wing drone and entering appropriate commands directly into the local MAVLink communications protocol running on the simulated drone. Each of the appropriate commands was entered for each of the drone types a total of ten times in random order. The MAVLink PWM communication to the simulated motors was monitored to ensure expected response to each command. Additionally, the modeled drone behavior in JSBSim in response to each command was monitored for proper simulation as each command was executed. Testing was not considered successful until 100% of drone commands resulted in the expected response in both MAVLink PWM communication and JSBSim simulation results.

b. Communication between Drone and Ground Control Station (GCS).

The second phase of simulator testing involved communication between the simulated drone and a remote ground control station. Since ultimately the intent of the project was to allow for the GCS VM to act as the remote operating station, the test was established to monitor the successful communication between the simulated drone and the GCS VM. Both a quad copter and a fixed wing drone were simulated on the Drone VM. The MAVProxy GCS software on the GCS VM was established as the controlling station associated with each drone. Commands were passed from this controlling station to the drones, such that each command applicable to the drone type was initiated ten times over the course of a two-hour run. During this time, the drone was monitored for proper response to each command. It was observed that 100% of the initiated commands were executed by the drone. Additionally, each command was attempted when the drone was in a powered-down condition to verity negative response. In all such cases, the drones responded as expected. Additionally, throughout the 2-hour run time, packets transmitted/received by the drones and the MAVProxy software were monitored to determine the extent of packet loss. 99.9% of transmitted packets were received, which is appropriate for the UDP protocol used by MAVLink. In fact, this result may be someone unrealistically high, as environmental interference may be significantly higher in an actual drone, depending upon its location and proximity to the controlling station.

During the communication test described above, the synchronization between the drone and the MAVProxy plot of the drone to determine GPS error. At specified intervals, the drone's reported GPS position and the MAVProxy's determined position for the drone were recorded and compared. 73% of the time the two GPS locations were identical. And even when they differed, the difference was less than ½ of the level of accuracy possible given drone speed and update interval. Thus the drone position and MAVProxy plotted position never differed by more than the distance the drone could travel in a single update period, making the error solely a function of the frequency of communication between the drone and MAVProxy. Thus these results approximate the best case for actual drone performance, as the design of the system suggests some additional error between the predicted drone position and it's actual position is likely, especially when environmental factors such as wind are shifting rapidly or of high magnitudes (i.e. wind speed approaching 70% of drone speed at a 30 degree angle off drone course).

c. Interface Testing and Interoperability.

Three major interfaces between the server environment and outside operators (via the internet) were tested for security and operability. First, the web hosting of the PHPVirtualBox interface was tested by attempting to access each of the established accounts from different device types (Windows computer, Ubuntu computer, iPhone, iPad, and Android tablet) from different external domains (Starbucks hosted WiFi, Verizon hot spot, AT&T data services, and GoWifi). For each device, on each internet provider, each account was used to logon to the PHPVirtualBox interface and then used to start, freeze, resume, and stop each VM. In all cases, no abnormalities or errors were observed during these tests. Similar testing was performed on the Secure Shell Server, using a variety of applications from each of these external internet hosting services. For example, PuTTY was used for the windows machine, Juice SSH on the android tablet, and SSH Term on iOS devices. The internal SSH command line interface was used for the Ubuntu machine. Again, in all cases, each account was able to remotely logon to the virtualization server via SSH. Finally, the ability to remotely connect to each VM via Remote Desktop Connect was tested using the RDC program on a windows machine, Remoter Pro on iOS devices, and RDC Connect on the android tablet. As before, these tests were made from the various internet hosting services listed above. While the delay was more pronounced for public hotspots, due to a combination of high loading from multiple users and lower quality WiFi standards due to maximizing operability with all kinds of devices and users, connections were manageable in all cases. However, it was noted from other group members that increased geographic separation greatly amplified the observed delay. The testing was repeated from a 3000-mile geographic separation and the resulting delay was well below the 60 Hz refresh rate considered acceptable (observed refresh was as low as 0.1 Hz on a public network at 3000-mile geographic separation). However, mean refresh rate was 10 Hz when more ideal hosting environments were selected, which is manageable but far from ideal. As previously discussed, increased performance will require either reducing the geographic separation or increasing the bandwidth for the server well beyond that typical of residential internet services.

Security of the server was tested using a Nessus scanner and OpenVAS 8.0 automated vulnerability scanning tools. Some vulnerabilities were initially discovered, but all were successfully patched. Subsequent scans with each subsequent update were performed; no critical vulnerabilities were identified during any of these subsequent security scans.

d. Red Team Goals.

The goal of the Red Team was to demonstrate an attack using the various tools researched for this task. Research into the tool suite in Kali was key to understanding different approaches in analysis of a target, intrusion of the target, and taking over the target. After familiarization of the tools was achieved, Zach was able to start working within the virtual environment that was set up. This environment included the Kali Virtual Machine (VM), the Ground Control Station VM (GCS) and the Drone VM (UAV). Knowing and understanding this environment was key to the following steps that were designated as goals to take over the drone:

- Find IP addresses of Ground Control Station (GCS) and Drone (UAV)
- Find likely ports of communication that command packets were using
- Perform Man in the Middle attack using information above, taking control of UAV.
- Command UAV

The methods of gaining an IP address and router information were initially very useful as it provided a starting point to work with. To cut down on initial work, certain control variables were set. These variables were the IP addresses of the UAV and GCS, the port numbers they were working on, and the protocol they were using. Using these variables, team was able to develop a script that takes command of the UAV. The tools used for this are, in the order of usage, as follows:

- IP forwarding "on" (Kali inherent functionality)
- Scapy ARP spoofing (scapy (P, 2010) is a python-based packet manipulation program)
- IP forwarding "off"
- ifconfig Kali settings set to GCS settings
- mavproxy started and control is taken

The goals of finding the IP addresses of the GCS and UAV and their ports that the command packets are sent were achieved through a combination of the tools researched. To find the UAV and GCS IP's was a matter of scanning for UDP packets given a range. The special part about the UDP packet scan is that it reveals otherwise unknown IP addresses exclusively using the network for UDP messages. The range of IP addresses was the factor that determined how long the scan would take and could be anywhere from a few seconds to hours long. In a real-world situation, reconnaissance would be done before the attack would take place so this would be a low to medium issue.

The way the scan works is that it sends UDP command packets to every IP address listed in the range specified. This scan also attempts to send the UDP packets on a closed port, selected intuitively. This scan then waits for return packets. Upon packets returning with a fail, because the port would be closed and the packet rejected, the IP is revealed to the scan and shown to the user. The same method applies for port scanning as well but this time it is scanning for ports that are in use by the GCS and UAV. This port scan could take a very long time as the ports being used by the GCS and UAV could be changed. However, an unsuspecting client or someone in the unknown would not know that the default port number is specified on their website thus narrowing down the range significantly. This is done manually thus far to incorporate this scanning process into the script designed below.

e. Monitoring Packets through Scapy.

Scapy is a new tool that red team was working with and was chosen on account of more control over what packets are sent, the number of packets sent, python compatibility, and less commands and memory used by the python script. Knowing the UAV and GCS's IP address and the GCS command port, team was able to develop a python

script that uses scapy IP spoofing, ifconfig settings, and the mavproxy commands to take control of the UAV. Mavproxy, the protocol used for two years by the PIXHAWK MAV project ("MAVLink Micro Air Vehicle Communication Protocol – QGroundControl GCS," 2015), is used to send commands to the UAV. The UAV initially starts with its output directly forwarded to the GCS's IP and port. This establishes a 'master' via wireless or wired using mavproxy. When this happens the user at the GCS can send commands to the UAV and the UAV follows them as if someone entered the command directly from the UAV's microcontroller.

The script takes the GCS's IP, the UAV's IP, and the interface used for the communication, limited to 'eth0' for now, and starts forwarding the IP address immediately. This gives the Kali user the ability to be the man in the middle. From here, the script automatically goes through the steps to insert itself as the 'master' user for the UAV. This includes enabling and disabling IP forwarding, severing the link between the UAV and GCS. While the GCS and UAV are trying to reestablish a link, the script sets the Kali computer as the 'master' using a subprocess call to mavproxy. Once this is done the user will be able to see heartbeat messages from the UAV and is able to send commands to the UAV. This whole process is automated through Python scripting and tools within python.

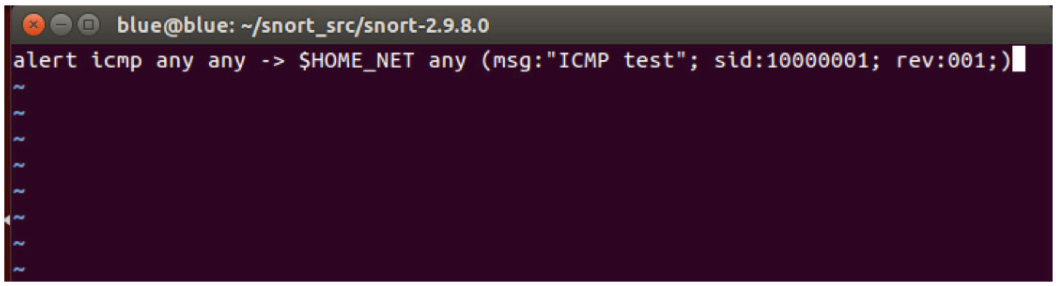

Fig 7. Snort rule file

The tools and code used to script in Python are as follows:

- Importing scapy.all for scapy features
- **echo** command that inserts and saves to a file
- **send** command that sends a packet (scapy tool)
- **subprocess** call that starts a subprocess

These are the main commands that were used along with different parameters. From these tools and commands, three functions were built that accomplish the tasks outlined above. The first is the spoofing function that uses **send** to send ARP packets to the GCS and UAV to actively spoof them. The second is the restoring function that, once the program is closed, restores targets and the system back to default using the same command. The third and final function is the main function where the various other functions are called and the actual attack occurs.

The first task the script does is ask for the IP addresses and the Interface. Once the script has these, the process of attacking begins by port forwarding using the **echo** command, spoofing using the first function, severing the link by disabling port forwarding, and taking control with the **subprocess** call to **mavproxy**. This achieves the goal of demonstrating a man in the middle attack.

Further future development is required in the form of target analysis and wireless simulation. Because the simulation is using a wired, Ethernet connection the method of gaining control of the UAV is different than what it would be in real-world applications. There is also much to be said in terms of "fast" target analysis. The processes of uncovering the GCS and UAV's IP addresses and ports being used have been developed. The only problem with these processes is the length of time it could potentially take to scan for all possible targets. There is also further work to be done in automation of the analysis, but that task is a stretch goal. Overall, the tasks and goals set by the Red Team have been met.

f. Blue Team Goals.

The goals for the Blue Team involved researching the potential defensive tools. Team explored the possibilities of using Snort on Ubuntu tools for defense. Snort is a sniffer application that will monitor all network packets and will compare them to user generated rules. The initial set up of Snort is intensive and required several attempts to get the configuration files edited correctly and get the program running.After successful installation of Snort it was necessary to learn how to write rules and test them. The basic format for a Snort rule is:

action protocol src_ip src_port direction dst_ip dst_port (rule options)

A sample rule can be seen in Fig 7. This rule was used, with some modifications at different times in the testing to verify the system responses. The actions available in Snort are alert, log, pass, activate, dynamic, drop, reject, and sdrop. So far the actions the team is using are alert, log, and pass. The alert function will generate and alert on the system and then log the packet. Log will log the packet without alerting the user. The log files from both these options can be monitored by other programs to take further action. The pass action will allow a packet to be ignored so that false positives are not generated. This will be used to allow the traffic between the UAV and GCS to be passed through without spending system resources monitoring all the packets transmitted between them.

The protocol section for this has the choices of TCP, UDP, IP, and ICMP. During initial testing rules were written to monitor for ICMP packets to detect a system ping. The rules used for the final set up will monitor for UDP protocols since this is the type of packet used by the MAVLink commands.

The src_ip, src_port, dst_ip, dst_port, and direction all serve to specify where the traffic is coming from and going to. The rules were are using are bi-directional, and will detect traffic from all source ip addresses to all destination addresses. This is specified by using ANY for the addresses. Since we have pass rules for the legitimate traffic between the GCS and the UAV the traffic between those two will be ignored. At this point we are not using any custom rule options.

In the final setup, the system will utilized the PulledPork application to keep the rules up to date with the latest network defenses. The Pulled Pork rule files will assign a priority level to the types of attack detected and log all suspicious traffic. If it is noticed that some of the routine traffic between the GSC and the UAV is causing alerts then the local rules files will be edited with pass rules that will allow that traffic through without further analysis.

The log files are monitored by a separate application. The team chose to use Swatchdog to monitor the log files and act on the alerts. The current setup will has Swatchdog run sendip to send a Return to Launch (RTL) signal to the UAV in the event that a Snort alert of priority 2 or higher is found. Swatchdog has the ability to only respond to an attack once in a given period of time to limit the number of alerts, emails, etc. Since the UAV will not be harmed by the RTL command being sent repeatedly it was decided that we will respond to every attack and send the RTL command as often as necessary to ensure recovery of the UAV. These files must be edited everytime the GCS and UAV are paired to ensure that the correct ip address and port are included in the RTL command.

The RTL command was captured by Josh by logging the traffic between the GCS and the UAV while the commands were being sent. By sending the RTL code the drone will be placed into a flight mode that will cause it to fly back to the GPS coordinates that it was initially armed at. This will prevent the loss of the drone.

Since Snort did not have the capability to detect a man in the middle attack in progress, the decision was made to implement an additional application to monitor the ARP map to watch for this type of attack. The ARP map contains a listing of the MAC address and IP address of all devices connected to the network. Arpwatch works by making a copy of that map and then comparing the current version to that original to detect changes that could be caused by arp poisoning attacks or by a man in the middle attack. The Arpwatch program will normally send an email notification of any changes. These alerts are also written to the syslog file. By running a second occurrence of Swatchdog to monitor the syslog file the RTL command can be sent based on those alerts.

Running the virtual machines for the UAV, GCS, and the defensive box locally showed the expected responses when a ping triggered the local rule. The rule for the ping response was then commented out of the configuration files and the modified VM was saved for transfer to the server. Once running on the server it will probably be necessary to update the rules and the response code to reflect different IP addresses.

To allow the sendip to function correctly it was necessary to run it as the root user (sudo). This action normally requires the root password to be entered, but by modifiying the sudoers file, sendip was given permission to run as root without the password. Snorby was implemented to provide a simple method of viewing snort alerts for testing purposes. This application pulls all of the Snort alerts from the log files and displays them in a web-based GUI to allow for easy analysis of the system.

g. Security Onion Vs. SNORT for Intrusion Detection System.

The team explored Security Onion for Intrusion Detection System (IDS) tools. Security Onion was found to be useful for familiarizing oneself to a variety of IDS tools and their functions and operation. However, it also included a number of tools not useful for the goals of this project, and used excessive storage space and memory. Also, Snort requires extensive configuration which made a pre-installed version of it no more useful than a manual install on a clean Linux OS. The decision was made to create a defensive suite built upon a clean version Ubuntu 14.04.3 with Snort as the primary defensive tool that would interface with other tools to provide IDS and IPS.

Snort is a highly effective and popular tool used as an IDS and Intrusion Prevention System. It allows for configurable rule sets to detect attacks and then act upon them. After installation of Snort, the rule sets and configuration file were edited to provide for alerts to the System Log files when an attack was detected. These alerts would then be detected by the tool Swatchdog. Swatchdog monitors the Linux log files, and then performs a pre-determined action when it detects a specified alert. In our case, Swatchdog can be used to detect the text string 'Alert', or the name of the type of the attack that is included in the alert log. Once the text string was detected, Swatchdog executed a custom Bash script to send a command to the UAS. Josh built a VM using these tools and saved it as an OVA file. Tanya continued work on this portion of the project creating an Intrusion Prevention System on the Blue Box.

Kismet is an IDS that is designed to detect traffic over Wi-Fi networks. It is possible to interface Kismet with Snort to provide a more effective coverage of Wi-Fi networks than Snort alone can provide. Snort and Kismet were interfaced to test this configuration and successfully worked together. However, the current stage of the project in simulation only, does not require the Wi-Fi IDS capability. Therefore Kismet will not be used until the project moves to a real world setup.

h. Honey Pot Attack Defense.

The team then began working on the development of a 'Honeypot' defense. This would be an Intrusion Prevention System (IPS) to help deter and confuse attacks. A traditional Honeypot will create a fake serve or a computer on a network for a hacker to detect and being to attack. The honeypot will then often be able record the attempted attacks for later analysis. The Honeypot for this project would need to be similar, but with some unique requirements. Rather than simulating normal computer/server activity, it would need to simulate the activity of a UAV and it's Ground Control Station (GCS). After researching a number of open source and commercial honeypot solutions, it was determined that none were available to simulate the traffic between a UAV/GCS, so the decision was made to create a custom honeypot. The honeypot would serve two roles. Firstly, it would confuse and deter attacks. A hacker observing the traffic between the GCS/UAV would see both legitimate traffic and the honeypot traffic, and would be unable to determine which traffic was genuine without extensively analyzing it. The second purpose of the honeypot is to draw attempted attacks on the honeypot traffic, so that the defensive measures using Snort/Swatchdog can be activated. The more attacks the hacker makes on false traffic, the more opportunities would exist to detect these attacks and apply preemptive defensive measures.

The first suggested architecture of the honeypot was simply to use MavLINK as shown in Kevin's portion of the project to create a false GCS/UAV. This was found to be effective at creating false traffic, however it required additional Virutal Machines and/or actual computers to complete.

A less resource intensive solution was then determined to be creating the GCS/UAV traffic directly. This was accomplished using the SendIP tool. SendIP allows for the creation of arbitrary IP packets over a network. Wireshark was used to analyze the IP traffic created by the actual GCS/UAV traffic. Through careful analyzation of the collected packets it was possible to determine a rough pattern of packets and included data. This pattern was then recreated though SendIP. The goal was not for an exact replication of the packets, as each session between a GCS/UAV would

be unique, but simply to approximate the traffic an attacker would expect to see with Wireshark.

The SendIP program includes a looping function. This was used in conjunction with the Linux 'watch' command to create 2 levels of looping with varying times. By adjusting the timing of these loops, different pseudo random patterns of packets could be created to make any attempt at analyzing these packets difficult and time consuming. Multiple concurrent SendIP commands also added to the complexity and randomness, again making it difficult for any attacker to detect the false traffic. After completion of the honeypot script, Zach assisted by encoding it within a Bash script to make it easier to start up and begin sending the code. Blue team also assisted the Red team by capturing and decoding common MavProxy messages so that they could be used in Red Team attacks. For instance, SendIP messages were created that could be used to send Mode Return to Launch, Mode Auto, Mode Guided, Mode Loiter, and Mode Land MAVProxy messages without the need for MavLINK.

VI. System Performance and Testing

Extensive testing of the sub-systems involved in the project were performed in order to ensure these sub-systems operated correctly at both the level of the individual sub-system and the interactions between sub-systems. A full description of the testing plan and a table showing the test results can be seen in Table 1.

First and foremost, testing of the virtualization server itself was performed. This testing included the testing of the PHPVirtualBox web hosted interface (which includes the Apache2 web server, the VirtualBox hypervisor, the password database LDAP directory, and the PHPVirtualBox web interface) to ensure all user accounts could log on to the system and manipulate VMs, first from the local network and then from an external internet host via the RogueNuke.com domain. All tests initiated from the local network were successful from the beginning, however none of the attempts from an external internet connection yielded positive results. Troubleshooting revealed the issue was due to ISP blocking external requests on standard HTTP ports. Initially this problem was solved using a P2PTP-based VPN configuration, but that failed after two days, as the internet-facing IP address for the server was not static. Ultimately the RogueNuke.com domain was re-configured to forward HTTP traffic to the server via non-standard ports combined with a dynamic domain name association. Once this was accomplished, testing yielded successful results from external internet hosts. Server testing also included the testing of the Secure Shell (SSH) server and Remote Desktop Connect (RDC) services. SSH testing involved remote server logon from both internal and external internet hosting via each authorized account and utilizing the command line interface for the VirtualBox hypervisor via this remote connection. RDC testing involved logging on to each VM remotely via the RDC connection, editing a text file on the VM desktop, performing an update to the VM, and successfully shutting down the VM via the RDC connection. Initial testing for both SSH and RDC were successful from the local network, but suffered the same problems identified for the PHPVirtualBox interface noted above. Fortunately, the same solution was successful in resolving these problems; ultimately both the SSH and RDC testing was successful.

Drone VM sub-system and GCS sub-system individual component-level testing included testing of the MAVLink communications protocol configuration. The ability of the ArduPilot software (both ArduCopter and ArduPlane) to receive and respond to MAVLink communications generated locally on the Drone VM was tested to ensure no errors with communications protocol configuration on the Drone VM. Testing was accomplished by sending MAVLink commands to the ArduPilot software which conform to each of the standard commands applicable to both the ArduCopter and ArduPlane variants generated by MAVProxy software installed on the Drone VM. Testing was considered successful when messages generated locally by the ArduPilot software were received and properly decoded by the ArduPilot software. Any instances of lost or misinterpreted MAVLink messages would have constituted test failure, but no instances of this were noted.

Similarly, the MAVLink configuration on the GCS VM was tested. The ability of the GCS software to receive and respond to MAVLink communications generated locally on the GCS VM was tested to ensure no errors with communications protocol configuration on the GCS VM. Testing was accomplished by sending MAVLink commands to the GCS which conform to heartbeat messages and drone status reports generated by ArduPilot software installed on the GCS VM. Testing were considered successful when all messages generated locally by the ArduPilot software are received and properly decoded by the MAVProxy GCS software. Any instances of lost or misinterpreted MAVLink messages would have constituted test failure, but as with the Drone VM, no such failures were observed.

Synchronization between MAVProxy GCS software and Drone Location was also tested at the individual sub-system level. The ability of the Ground Control Station to synchronize GPS location with a simulated GPS location input and display proper map data was tested to ensure the GCS software was capable of detecting and responding to a known location input during system initialization. This was a "go-no go" test. A single initial GPS input was provided via text file generated internal to the GCS. When the GCS initializes with the drone marker in on the map position corresponding to that GPS location the test was considered successful.

Data exchange between flight simulator program and ArduPilot was also tested at the component level. The communication between the simulated PWM signal inputs between the ArduPilot autopilot and the JSBSim flight simulator to provide correct aerodynamic inputs and GPS position information based on simulated engine operation and navigation controllers was examined to ensure the drone performance for a given set of input data conformed to the commanded behavior. Testing was accomplished through 25 trials of all standard commands for both a fixed-wing and quad-copter drone types (ArduPlane and ArduCopter autopilots respectively). Each command applicable to that aircraft was tested during each trial. No performance abnormalities observed during this testing. Parameters such as battery life which do not directly impact response to navigation commands were not tested. Performance in response to environmental conditions, such as wind shear, likewise were not tested.

Upon the completion of individual sub-system testing, the interactions between the GCS and Drone sub-systems were examined. Specifically, MAVLink communication between the Drone VM and GCS VM were tested, the objective of which was to ensure MAVLink messages generated external to the Drone VM and transmitted using the UDP communications protocol were properly received and processed by the ArduPilot software. Simultaneously, the testing ensured MAVLink messages generated by the Drone were properly transmitted and processed by the MAVProxy GCS software on the GCS VM. Testing was accomplished by using the MAVProxy GCS software on the GCS VM to transmit commands to the Drone VM and observe drone appropriate response given those commands. Ten iterations of each command applicable to both the ArduCopter and ArduPlane variants of the were tested. The criteria for a successful test was proper drone response for each individual command with zero losses. Likewise, all status messages sent by the drone must have been received and properly interpreted by the MAVProxy GCS. Finally, for the duration of the testing, the drone position, speed, altitude, and heading must have at all times been synchronized between the Drone and the GCS, as evident by simultaneous observation of drone location and behavior on the map status pages of both the Drone VM and GCS VM. Any observed loss of synchronization would have constituted test failure. However no such failures were noted.

Testing of the intrusion detection system was performed initially to validate that the programs all worked as designed. Local rules were written for Snort which caused an icmp ping to be logged as a priority 1 and then priority 2 alert. A ping was sent to the VM and Snorby was used to verify the correct priority alert was generated. Then the network traffic was viewed to ensure that the RTL command was sent by Swatchdog. All iterations of the testing performed as expected.

Next the IDS system was run with the GSC and UAV to verify that no false positives occured. The UAV and GCS were run to battery shutdown 10 times, providing approximately 300 minutes of flight time. During these flights there were no Snort alerts generated.

Honeypot testing was initially conducted locally with success. The honeypot was able to create streams of data, which when observed through Wireshark, closely matched the actual traffic. The rate of packet creation by the honeypot was found to match the rate of creation of a sample of the actual GCS/UAV traffic with an accuracy of 91.9%. The packet lengths of the honeypot matched the packet lengths of the GCS/UAV with an accuracy of 81.5%. This was deemed to sufficiently simulate actual GCS/UAV traffic and to prove viability of the honeypot. The SendIP tool can also be easily modified to create more or less traffic in the event of a GCS/UAV combination with different characteristics.

The honeypot testing was then attempted using the virtual setup as discussed above. An error was discovered in the results, as Wireshark on a third party system would not detect the SendIP traffic. After some research, it was discovered that the simulation's reliance on using the eth0 port (simulating a wired connection) would not fully simulate the effects of a WiFi system. In particular, when eth0 is used to transmit SendIP messages, the message leaves the host computer with 'fake' IP address and is caught by the router. The router will then reject these messages as it does not know the location of said fake IP addresses. This would not be a factor or problem in a wireless setup, so this error was ignored for our purposes. If running the simulation using eth0, it is recommended to analyze the

honeypot data from within the Blue VM in order to observe all expected traffic.

The Red team performed attacks against the GCS and UAV without the defenses to provide a metric that the later results could be compared to. The attacks were ran 100 times against the system. The man in the middle attack was 100% successful and the rebroadcasting attack was 96% successful.

These attacks were repeated with the Blue VM running. Because of the simulated network using a wired configuration, and the Red team attacks already set up against the correct IP addresses, the effectiveness of the honeypot and Snort were not tested. Arpwatch was able to detect 100% of the attacks by the Red team, but the RTL command was not received by the UAV. It was discovered later that the swatch configuration files were not edited to contain the correct port and ip addresses, so the commands may have been sent but were not acknowledged by the UAV. Figs 8, 9, and 10 show some of the arpwatch alerts extracted from the syslog file after the testing was completed. The alert can be seen for when the Red Team VM initially connects to the network in Fig. 8, an ip address change can be seen in Fig. 9, and Fig 10 shows an address 'flip flop'.

```
Apr 29 17:46:14 blue arpwatch: new station 192.168.103.128 08:00:27:23:16:2a eth0
```

Fig. 8 – Red VM connected to network

```
Apr 29 17:46:25 blue arpwatch: changed ethernet address 192.168.103.113 08:00:27:23:16:2a (08:00:27:f9:54:3d) eth0
```

Fig. 9 – Red VM change IP address

```
Apr 29 17:46:36 blue arpwatch: flip flop 192.168.103.113 08:00:27:f9:54:3d (08:00:27:23:16:2a) eth0
```

Fig. 10 – Flip Flop of assigned IP addresses

This final alert was repeated 100s of times throughout the man in the middle attack. If the RTL was sent correctly, the UAV would have returned. The test plan is based on progressive testing of sub-systems independently, testing of the interactions between sub-systems, and finally testing of the complete composite system. For purposes of this testing, sub-systems are identified as the Drone Simulation VM, Ground Control Station Simulation VM, Red Team VM, Blue Team VM, and Virtualization Server. The purposes of testing are to demonstrate both verification and validation of the simulator and all sub-systems therein.

By utilizing a "bottom-up" approach, we can effectively ensure the above sub-systems conform to both the design parameters and functional requirements before integrating them into the composite whole. This process will allow us to identify problems with either verification or validation at the lowest possible level before progressing on to the more complex interactions between sub-systems and finally to the complete system.

VII. Sub-Systems Testing

i) Part I. Sub-system testing

This testing phase tests the operation and performance of vital functions organic to each sub-system independently. During this phase of testing, any failures detected will be addressed by modifying the system under test and repeating testing until no failures occur.

Drone Sub-System Testing. The ability of the ArduPilot software (both ArduCopter and ArduPlane) to receive and respond to MAVLink communications generated locally on the Drone VM will be tested to ensure no errors with communications protocol configuration on the Drone VM. Testing will be accomplished by sending MAVLink commands to the ArduPilot software which conform to each of the standard commands applicable to both the ArduCopter and ArduPlane variants generated by MAVProxy software installed on the Drone VM. Testing will be considered successful if all messages generated locally by the ArduPilot software are received and properly decoded by the ArduPilot software. Any instances of lost or misinterpreted MAVLink messages will constitute test failure.

Ground Control Station Sub-System Testing. The ability of the Ground Control Station to synchronize GPS location with a simulated GPS location input and display proper map data will be tested to ensure the GCS software is capable of detecting and responding to a known location input during system initialization. This is a "go/no-go" test. A single initial GPS input will be provided via text file generated internal to the GCS. If the GCS initializes with the drone marker in on the map position corresponding to that GPS location the test will be considered successful.

The ability of the GCS software to receive and respond to MAVLink communications generated locally on the GCS VM will be tested. Testing will be accomplished by sending MAVLink commands to the GCS which conform to heartbeat messages and drone status reports generated by ArduPilot software installed on the GCS VM. Testing will be considered successful if all messages generated locally by the ArduPilot software are received and properly decoded by the MAVProxy GCS software. Any instances of lost or misinterpreted MAVLink messages will constitute test failure.

Virtualization Server Sub-System. The sub-system for this testing includes the Apache2 Web Server, the PHPVirtualBox interface, the VirtualBox hypervisor, and the password database. Testing is accomplished by successfully logging into the PHPVirtualBox interface, first from a web browser installed on the local network and then from an external internet connection. Testing is successful if each of the user accounts can be logged into and the VMs may be started, stopped, and modified via the PHPVirtualBox web interface.

SSH Server Connectivity testing examines the ability to remotely log into the server operating system via a Secure Shell (SSH) connection. Testing is considered successful if the sever can be logged on to successfully from each account. Testing will include the command-line interface to start, stop, reconfigure, and export VMs through VirtualBox, and the navigation and editing of flies related to the project on the server, as demonstrated by the ability to edit the PHPVirtualBox configuration file.

Remote Desktop Connect testing examines the ability to remotely operate VMs using the Microsoft Remote Desktop Connect software. Testing will be accomplished by establishing a RDC connection to each VM, logging in successfully to the VM via the RDC connection, editing a text file on the VM desktop, performing system updates via RDC, and successfully shutting down the VM via the RDC connection. Additionally, testing will not be considered successful unless these functions are able to be accomplished from both a computer on the local network and from an external internet connection via the RogueNuke.com domain.

Blue (SNORT and HoneyPot) Sub-System. The SNORT sub-system will be tested by running the box and sending test attacks that will be recognized as the different priority levels to ensure the correct responses are sent. Testing of this phase will be successful if the system send the correct swatchdog response. The test will be considered successful if the expected response to all of the attacks is executed.

Testing for the 'HoneyPot' will be conducted using WireShark, Wireshark's statistical analysis tool, and Minitab. The HoneyPot creates traffic that appears to come from multiple UAV's and GCS's using the SendIP tool. This traffic can be captured by WireShark, and then compared to actual UAV and GCS traffic. Resident within WireShark are statistical tools that allow for the ability to determine a number of properties of this captured data, such as packet length and frequency of traffic from individual IP's. A comparison in amount of traffic will be conducted using Minitab, with a statistical analysis resulting in percentage of 'HoneyPot' traffic compared to actual traffic. A graphical and numerical comparison of packet length will be conducted using data from 'WireShark' analyzed by Minitab. The 'HoneyPot' will be considered successful if it can create multiple false UAV/GCS's that generate 90-110% of the number of packets as the actual UAV/GCS, and if the false traffic packet length has a similar (90%) normal distribution to the actual.

ii. Part II. Sub-system Interactions Testing

This phase of testing tests the performance of key seams and interactions between individual sub-systems. The focus is on the response of individual sub-systems in response to inputs into the sub-system and the resulting outputs generated by the sub-system.

Drone Sub-System Testing / GCS Sub-System Testing will be accomplished by using the MAVProxy GCS software on the GCS VM to transmit commands to the Drone VM and observe drone response is correct given those commands. Ten iterations of each applicable command will be tested. The criteria for a successful test is proper drone response for each individual command with zero losses. Likewise, all status messages sent by the drone must be received and properly interpreted by the MAVProxy GCS. Finally, for the duration of the testing, the drone position, speed, altitude, and heading must remain synchronized between the Drone and the GCS. Any observed loss of synchronization will constitute test failure.

Red (Kali Linux) Sub-System / Drone VM Sub-System. To properly demonstrate the interaction between the Drone VM and the Red Sub-System ARPSpoofing is used. This will demonstrate the ability to read traffic being sent back and forth between the Drone VM and GCS VM, identifying key elements such as the Drone's IP address as well as its MAC address. This will show the interaction between all three sub-systems without actually attacking and a key process in what happens in a real-world situation.

Blue (SNORT) Sub-System / GCS Sub-System Testing / Drone VM Sub-System. The snort sub-system will be tested with the GCS and the Drone operating to ensure the rules do not generate false positives under normal operation. During this phase a 30 minute mission will be flown 10 times and the number of priority one and priority two alerts will be monitored. Any false alerts will constitute a failure and will require modification of the rules files.

iii. Part III: Composite System Testing

This final phase of testing examines the performance of the composite system, during both unprotected and protected attacks against an operating drone and GCS during routing commands.

Undefended Composite Attack Testing. This testing is accomplished without the defensive (Blue) VM in operation to establish a baseline of performance for an undefended system. 100 iterations of both a man-in-the-middle attack and a rebroadcasting attack against the operating Drone and GCS VMs and the success rate of these attacks will be examined. For the man-in-the-middle attack, an ARP poisoning attack using ARPSpoof will be used to establish the man-in-the-middle scenario. From there WireShark will be used to identify the Drone and GCS IP addresses and ports.
Finally, MAVProxy/MAVLink will be used to take control of the drone. An attack is considered successful if the attacker is able to issue a command to the drone to which the drone responds. For the rebroadcasting attack, commands issued by the GCS to the Drone will be captured by the attacker using WireShark and then rebroadcast at a later time using Sendip. An attack is considered successful if the drone responds to the rebroadcast command. Finally, the LowOrbitIonCannon (LOIC) will be used to generate a denial of service attack. The attack is considered successful if the LOIC successfully swamps the network router. Drone response to the LOIC attack will be observed and the time required to re-establish communications between the GCS and the Drone once the attack is ceased will be measured.

Intrusion Detection Composite Testing. Testing of the entire system will be accomplished by initializing the UAV, GCS, Snort system and honeypot virtual machines and then launching the red team VM. The tests from the undefended attack testing will be repeated and the rate of success from these attacks will be compared to the testing results without the defensive packages running and will be used to determine the effectiveness of the defensive software. Table 1 show the system results of sub-systems of the project with a testing metric of pass or fail status.

Table 1. System Results

Test	Goal	Pass /Fail
	Individual Sub-Systems	
Drone	Target - All messages received and decoded	
	Actual - All messages received and decoded	Pass
GCS - GPS Testing	Target - Drone marker initializes in correct position	
	Actual - Drone marker initializes in correct position	Pass
GCS - MavLink Communications	Target - All messages received and decoded	
	Actual - All messages received and decoded	Pass
Server - User Account Access	Target - All users able to log in and access VMs	
	Actual - Using document put out by Kevin all users can log in	Pass
Server - SSH Connectivity	Target - All accounts can log in through SSH connection	
	Actual - Accounts can log in through SSH connection.	Pass
Server - Remote Desktop Connection	Target - Access and use of all VMs successful using RDC	
	Actual - Access and use successful but slow	Pass
IDS box	Target - Correct response to all priority attacks	
	Actual - Correct response sent to all priority attacks	Pass
HoneyPot - Number of Packets/second	Target - 90-110% of actual UAV/GCS packet rate	
	Actual - 1-(HP 11.94-UAV 11.04)/UAV 11.04=92.44%	Pass
HoneyPot - Packet Length	Target - 80% or higher of normal distribution to actual packets	
	Actual - 1-(UAV 21.0 -HP 17.14)/UAV 21.03=81.5%	Pass
	Sub-system Interactions	
Drone - GCS Command response	Target - Correct responses and zero command losses	
	Actual - Correct responses and zero command losses	Pass
Drone - GCS Synchronization	Target - No observable loss of synchronization	
	Actual - Synchronization losses occur if left idle for too long	Pass
Red box - Drone	Target - Able to read traffic and identify IP and MAC address	
	Actual - Able to read traffic and identify IP and MAC address	Pass
IDS - GCS - Drone	Target - No false positives during normal drone operation	
	Actual - No false positives of any alert level	Pass
	Composite System	
Undefended Rebroadcast	Target - none, used to establish baseline	
	Actual - 96%	
Undefended Man-in-the-Middle	Target - none, used to establish baseline	
	Actual - 100%	
Defended Rebroadcast	Target - Detect and respond to all attacks	
	Actual - 0% undetected - NO RTL command received by UAV	FAIL
Defended Man-in-the-Middle	Target - Detect and respond to all attacks	
	Actual - 0% undetected - NO RTL command received by UAV	FAIL

viii. Ethical Considerations

Any research into possible malicious exploits of security vulnerabilities in software systems carries with it an inherent ethical concern for how that research may be used. Guidelines for this type of research are provided in the Federal Information Security Management Act (FISMA) and assorted industry publications. In general, it is the responsibility of an ethical penetration tester to inform manufacturers of security vulnerabilities identified in their products and allow them sufficient time to address these vulnerabilities—should they choose to do so—before disclosing vulnerabilities to general public.

With respect to this project, no new vulnerabilities were identified in the hardware or software being examined. Some of the attacks conducted as part of this project may be novel in the detailed mechanisms by which they are employed, but the vulnerabilities themselves are not new and have been well documented in academic research publications and presentations at various cyber security conferences. Therefore, nothing presented in this paper poses an increased risk to the public good, but demonstration of these vulnerabilities as presented in this project, as well as potential novel solutions, will hopefully further build on existing efforts and encourage industry and academia to respond to these security issues.

Additionally, as one of the primary goals of this research was to create a simulation environment that could be used to test offensive and defensive cyber security techniques without fear of creating actual hazards to the public, the product developed by this project serves to provide a safe and effective mechanism for accomplishing ethical testing without potential harm to existing networks and systems.

ix. Performance Metrics

Metrics for use in evaluation of this project included are: percentage of packet loss between the Drone and the Ground Control Station, percentage of successful Drone mission accomplishment, percentage of attacks/intrusions detected, mean time before attack/instruction detected, percentage of successful Drone return to start, percentage of malicious commands executed by the Drone, and frequency of unrecoverable Drone error.

x. CONCLUSION

Through the use of open-source Linux-based virtual machines, a fully customizable UAV simulation environment can be created for both software-in-the-loop and hardware-in-the-loop cyber security testing of fixed and rotory-wing ArduPilot over MavLink UAVs at a fraction of the cost required for testing with physical UAV assets. This project demonstrated how both offensive and defensive cyber security operations could be tested within such an environment. Utilizing the Kali Linux software suite, man-in-the-middle attacks and rebroadcasting attacks had 100% and 96% success rates respectively against a stock ArduPilot UAV; these results are consistent with those found in the literature for physical UAVs. Furthermore, the simulation environment developed in this project was used to design, develop, and test a novel UAV security scheme based on an automated SNORT-based intrusion detection system external to both the UAV and the ground control station. Initial tests of the individual systems have been successful and final testing of all areas is in process. The simulation environment developed in this project is effective for both evaluating the potential vulnerability of existing UAV control and communications systems, as well as for developing and testing potential solutions to those vulnerabilities.

Acknowledgement

This project is sponsored and made possible by Rockwell Collins Charitable Grant Allocation and Wells Fargo Foundation grants in 2015-2016.

REFERENCES

Buchanan, C., & Ramachandran, V. (2015). *Kali Linux: Wireless Penetration Testing Beginner's Guide*. Birmigham, UK: Packt.

Javaid, A. Y., Sun, W., Devabhaktuni, V. K., & Alam, M. (2012). Cyber security threat analysis and modeling of an unmanned aerial vehicle system. *2012 IEEE Conference on Technologies for Homeland Security (HST)*, 585–590. https://doi.org/10.1109/THS.2012.6459914

K, H. (2015). Todd Humphreys' Research Team Demonstrates First Successful GPS Spoof. Retrieved from https://www.ae.utexas.edu/news/features/todd-humphreys-research-team-demonstrates-first-successful-gps-spoofing-of-uav

Kim, A., Wampler, B., Goppert, J., Hwang, I., & Aldridge, H. (2012). Cyber Attack Vulnerabilities Analysis for Unmanned Aerial Vehicles. *AIAA Infotech at Aerospace 2012*, (June), 1–30. https://doi.org/10.2514/6.2012-2438

Lakhani, A., & Muniz, J. (2013). *Web Penetration Testing with Kali Linux*. Packt Publishing.

MAVLink Micro Air Vehicle Communication Protocol – QGroundControl GCS. (2015). Retrieved from http://qgroundcontrol.org/mavlink/star

P, B. (2010). About Scapy. Retrieved from http://www.secdev.org/projects/scapy/doc/introduction.html#about-scapy

Paganini, P. (2013). Hacking Drones - Overview of the Main Threats. Retrieved from http://resources.infosecinstitute.com/hacking-drones-overview-of-the-main-threats

S, F. (2003). Detection Tools: Snort. Retrieved from http://www.informit.com/guides/content.aspx?g=security&seqNum=48