Securing Commercial WiFi-Based UAVs From Common Security Attacks

Michael Hooper[1] ,Yifan Tian[1], Runxuan Zhou[1], Bin Cao[1], Adrian P. Lauf[2], Lanier Watkins[1], William H. Robinson[3], Wlajimir Alexis[1]
[1]Johns Hopkins University Information Security Institute, Baltimore, MD USA
[2]University of Louisville Computer Engineering and Computer Science Department, Louisville, KY USA
[3]Security and Fault Tolerance (SAF-T) Research Group, Vanderbilt University, Nashville, TN USA

Abstract— We posit that commercial Wi-Fi-based unmanned aerial vehicles (UAV) are vulnerable to common and basic security attacks, capable by beginner to intermediate hackers. We do this by demonstrating that the standard ARDiscovery Connection process and the Wi-Fi access point used in the Parrot Bebop UAV are exploitable such that the UAV's ability to fly can be disrupted mid-flight by a remote attacker. We believe that these vulnerabilities are systemic in Wi-Fi-based Parrot UAVs. Our approach observed the normal operation (i.e., ARDiscovery Connection process over Wi-Fi) of the Parrot Bebop UAV. We then used a fuzzing technique to discover that the Parrot Bebop UAV is vulnerable to basic denial of service (DoS) and buffer-overflow attacks during its ARDiscovery Connection process. The exploitation of these vulnerabilities could result in catastrophic and immediate disabling of the UAV's rotors mid-flight. Also, we discovered that the Parrot Bebop UAV is vulnerable to a basic ARP (Address Resolution Protocol) Cache Poisoning attack, which can disconnect the primary mobile device user and in most cases cause the UAV to land or return home.

Based on the literature and our own penetration testing, we assert that Wi-Fi-based commercial UAVs require a comprehensive security framework that utilizes a defense-in-depth approach. This approach would likely mitigate security risks associated with the three zero-day vulnerabilities described in this paper as well as other vulnerabilities reported in the literature. This framework will be effective for Parrot Wi-Fi-based commercial UAVs and likely others with similar platforms.

***Keywords*—UAV,ARDiscovery, network security, hobby, flight**

I. INTRODUCTION

Commercially-available and hobby-grade unmanned aerial vehicles (UAVs) have become a popular pastime for many enthusiasts [1, 2]. There is a growing demand to see UAVs employed for commercial purposes, but the realization of widespread commercial utilization has yet to be achieved due to complex regulatory environments and developments in sense-and-avoid technologies [3].

If these UAVs continue to grow in popularity or are even adopted for widespread commercial use, then it is important to consider the implications for the safety and privacy of the population. Imagine a future where commercial UAVs are employed by businesses in such a way that perhaps your next pepperoni pizza hovers outside your front door. Or, suppose that the United States Postal Service has grown weary of maintaining all of those delivery trucks, and your routine local mail is now airmail? These scenarios are not bound to our imagination, as companies such as Amazon, Google, and even NASA have experimented with commercial applications for the technology [3]. Many questions will need to be answered before UAVs can be commercially employed in the United States. In this paper we focus on the question: Are the devices secured in such a manner that they are safe to operate and therefore protected against technological exploitation?

The contributions of this work are: (1) the illustration of three zero-day vulnerabilities on the Parrot Bebop UAV and (2) a proposed security framework for Wi-Fi-based UAVs to guard against basic security attacks. It is our hope that by using our security framework, commercial UAVs can be securely manufactured from the beginning, which should limit the potential for abusing these platforms.

This work is organized into the following sections. Section II discusses related work and other technical research into the security vulnerabilities of commercial UAVs. Section III reviews the platform for Parrot Wi-Fi-based commercial UAVs. In Section IV, we analyze Parrot's ARDiscovery process. Section V introduces our multi-layer security framework. Section VI describes the experimental evaluation and introduces three Zero-Day exploits against the Parrot Bebop UAV. Results are presented in Section VII, while Section VIII provides conclusions and highlights future work.

II. RELATED WORK

The following is a survey of vulnerabilities found in commercial UAVs. To our knowledge, the three vulnerabilities that we have found have never been reported before for the Parrot Bebop (and for its newest iteration, the Bebop 2).

A. UAV Vulnerabilities

The author in [4] revealed how Defcon 23 hackers were able to exploit open *Telnet* access and issue a de-authentication attack to force Parrot AR and Bebop UAVs, respectively from the air. In our work, we utilize the *Telnet* application as a delivery method for our fuzzing technique (i.e., we point *Telnet* at port 44444 to deliver fuzzed Java Script Object Notation (JSON) records [5]), but we do not actually login to the telnet server on the UAV or kill any processes.

Security researcher Samy Kamkar utilized a Raspberry Pi, a *Perl* script, and freeware tools to de-authenticate a Parrot AR Drone from its controller. This approach allowed his scripts to take command of the UAV in flight [6, 7]. This attack, called

978-1-5090-3781-0/16/$31.00 ©2016 IEEE

Skyjack, is dependent upon exploiting the UAV's onboard Wi-Fi through a de-authentication technique.

The author in [8] demonstrated that a Parrot AR UAV 1.0 and 2.0 are vulnerable to ARP (Address Resolution Protocol) poisoning, DHCP poisoning, and MAC cloning. Our ARP Cache Poisoning attack is similar to this approach; however, to our knowledge, we are the first to demonstrate the vulnerability of the newer Parrot Bebop UAV to ARP Cache poisoning.

These previous hacks demonstrate the vulnerability of Parrot Wi-Fi-based UAVs. Most of these vulnerabilities were demonstrated on the older ARDrone platform. Our hacks have been exclusively done on the newer Bebop platform. We have confirmed that the ARP Cache Poisoning vulnerability mentioned in [8] on the older UAV platform also exists on the newer Bebop platform. Also, we have found an entirely new area of vulnerability never before reported on any other Parrot UAV platform, the ARDiscovery process. These previous works, coupled with the zero-day vulnerabilities that we have found, verify our suspicions that commercially available UAVs are likely not being designed or manufactured in a secure manner, and that the public remains at risk if these devices are employed at scale without appropriate security protections.

B. UAV Security Frameworks

To our knowledge, no other related works propose a multi-layer approach to securing UAVs. Some authors address basic security concerns for UAVs [4, 6-8], and others even acknowledge that every facet of commercial UAVs lack security [9], but these papers stop short of proposing a solution.

There is a need for a security framework for commercial UAVs, but to date none exists. For example in [10], the authors focus on securing the UAV's access point (AP). While in [11], the authors focus on securing the UAV's GPS and its AP. There has been one step in the right direction where the author in [9] identifies that commercial UAVs lack security in every aspect of the device. Further, the author surmises that the lack of security of the Parrot ARUAV is due to it being a toy; however, he also concedes that some military grade UAVs suffer from the same security issues. We believe that [9] actually calls for the type of security framework in which we propose in this paper. To be clear, these researchers bring general awareness to the massive security issue with commercial UAVs, but do not propose a comprehensive solution.

III. PARROT WI-FI-BASED UAVS

Parrot S.A., a French company that specializes in audio equipment, began development of consumer multi-rotor aircraft in 2010 to fill a growing area in the hobbyist aviation field [12]. Its first UAV, the AR.drone, was a smartphone-pilotable aircraft utilizing 802.11 wireless technology to provide real-time video and easy-to-use on-screen controls with stable flight characteristics. Since this initial development, Parrot S.A. has produced multiple iterations of their AR.UAV software framework, most notably on the Bebop and Bebop 2 airframes. The Bebop and Bebop 2 airframes are Linux-based quad-rotor airframes, and differ from most video-capable aircraft in their implementation of a digitally-stabilized fish-eye camera module (instead of using heavier, more complex physical gimbal mounts) that can be "moved" mid-flight by digitally selecting a region of pixels from the camera sensor. In outdoor settings, a GPS provides relative positioning information, while accelerometers and magnetometers assist in both flight scenarios. The aircraft features "return-to-home" functionality, automated takeoff, and automated landing features. With additional software, the Bebop series of airframes can also perform autonomous waypoint navigation.

Onboard 802.11 Wi-Fi modules can permit a client controller, either in the form of a smartphone or tablet computer, or a dedicated physical controller, called the SkyController, to bind and control the aircraft. Smartphones and tablets bind to either the aircraft directly, or to the SkyController (which provides long-range telemetry and physical joystick controls) by connecting to an open wireless access point on either the aircraft or the SkyController. The method by which devices are discovered is detailed in Section IV. In either configuration scenario, both the aircraft and SkyController maintain their open access points active. Control between a smartphone and the aircraft is implemented on the software side via a mobile application, in its current iteration called "FreeFlight 3," and runs on both Android and iOS platforms [13].

Because of the ease-of-use, utility, and price points of the Bebop series, as well as those offered by other manufacturers, consumers appear to be buying such aircraft in large quantities – up to millions of units, which possess the ability to bind to smartphones and tablets through Bluetooth and WiFi interfaces. Due to their popularity, and the open network architecture through which they communicate, the Bebop and Bebop 2 UAV from Parrot is a strong candidate for studying network and device security.

IV. ARDISCOVERY PROCESS

The process by which networked devices identify and connect to nearby nodes, particularly in an ad-hoc configuration, is known as a discovery protocol. Discovery protocols vary in implementation and details of operation, but their purpose remains the same: to identify and permit the connection of a wireless device to an existing network.

Parrot utilizes a discovery method known as ARDiscovery to negotiate the establishment of a connection between an aircraft running the AR.UAV 2.0+ software and a controller. As an example, this method can include the Bebop (1 or 2) aircraft and a smartphone, as well as a SkyController if present. The discovery protocol is limited to devices connected to its open Wi-Fi access point, and works over a combination of TCP and UDP channels, establishing a handshake over a known TCP port.

The controller initiates a TCP handshake from UAV-to-controller (d2c) and controller-to-UAV (c2d) in order to establish the necessary communication channels for a successful flight. Data sent between the connected devices is managed through JSON records sent via UDP. Once powered up, the FreeFlight3 application begins the ARDiscovery

process, which is required in order to find and negotiate communications between an aircraft and its controller. Association is typically done either manually, in the mobile device's Wi-Fi configuration system, or initiated by the software.

The Discovery Port shared by the controller and the UAV is specified in the ARDiscovery protocol as port 44444. For example, the controller would send the JSON record in Table 1 to port 44444 on the UAV and the UAV would respond with the JSON record in Table 2 if it does not already have a controller, otherwise it would respond with the JSON record in Table 3. Note, these JSON records give meta-data about the controller or communication channel between the UAV and the controller.

Table 1. Controller-to-UAV JSON record

<From the controller to the UAV>
{"d2c_port":54321,"controller_name":"HTC One M9","controller_type":"htc_himaulatt","device_id":"PI040338AA5B037455" }

Table 2. UAV-to-controller JSON record, controller accepted

<From the UAV to the controller>
{ "status": 0, "c2d_port": 54321, "arstream_fragment_size": 65000,"arstream_fragment_maximum_number":4,"arstream_max_ack_interval":-1,"c2d_update_port": 51, "c2d_user_port": 21 }

Table 3. UAV-to-controller JSON record, controller rejected

<From the UAV to the controller>
{ "status": -3999, "c2d_port": 0, "arstream_fragment_size": 0, "arstream_fragment_maximum_number": 0, "arstream_max_ack_interval": -1, "c2d_update_port": 51, "c2d_user_port": 21 }

V. MULTI-LAYER SECURITY FRAMEWORK

One of the most practical security frameworks to date is defense-in-depth [16]. It involves a multi-layered approach to computer network security. We believe that a similar approach should be taken when securing remotely controlled flying computers or commercial UAVs. Specifically, we are focusing on Wi-Fi-based commercial UAVs like the Parrot Bebop, DJI Phantom, or 3DR Solo. The challenge for security in these types of UAVs is the fact that they are wireless-network-capable computers that fly. Therefore, security must be considered for a computer, a navigational system, and a wireless network.

A. Common Security Attack Threat Model

The threat model that we employ in this paper is based on three basic attacks: (1) Denial-of-Service (DoS), (2) Buffer-Overflow, and (3) ARP Cache Poisoning. We developed this threat model by doing penetration testing on the Parrot Bebop UAV and discovering three zero day vulnerabilities. Then we tested and retested exploits derived from these vulnerabilities to verify their effectiveness in compromising the inflight behavior of the UAV. These three common security attacks are capable of disrupting the inflight behavior of the Bebop UAV. The Bebop UAV and others are likely vulnerable to several other common security attacks; however, this was not readily apparent in our security analysis. For the purpose of this paper, we will confine our multi-layer security framework to defend against these specific security attacks. We believe that this security framework is extendable and could be altered to accommodate defense against other threats as well.

B. The Controller-to-UAV Inter-networking Problem

In Section V.A and in Section VI.B, we identify and describe our threat model. The Multi-layer Security Framework is a direct consequence of our penetration testing efforts. Based on our security assessment of the Parrot Bebop UAV, we denote the issue to be *the Controller-to-UAV Inter-networking Problem*. In other words, we demonstrate that the processes and protocols used to internetwork the controller and UAV are insecure. We believe that this problem can be mitigated by adding: (1) a **watchdog timer** to limit the time the CPU can be used for non-navigational processing, (2) **hardline input data filtering** of all data sent to the UAV's embedded system, and (3) **anti-spoofing mechanisms** to the UAV's access point. We believe that this approach will harden the UAV enough to defend against common security attacks like the ones in our threat model. These security mechanisms are designed to operate in the network stack and operating systems (OS) domains. An illustration of the Multi-layer Security Framework is illustrated in Figure 1.

The proposed watchdog timer is designed to guard against DoS attacks and will operate in the OS domain. It will ensure that each non-navigational process is low priority in the scheduler (Algorithm 1: line 2) and only allowed access to the CPU for a specified time period τ within a given timeframe γ (Algorithm 1: lines 3 - 8). The offending processes will be restored after having waited two γ time periods (Algorithm 1: lines 9 - 12). This will be monitored at the process and scheduler level and enforced in the interrupt core of the OS. In other words, all non-navigational processes will have a lower priority than navigational processes and processes that attempt to use too much CPU time (i.e., greater than τ) will be interrupted and restored after having waited twice the observational time period (i.e., γ). The functionality of the proposed watchdog timer is illustrated in Algorithm 1. Both τ and γ should be set based on experimental testing to ensure that these parameters prohibit non-navigational processes from overwhelming the CPU, thus stopping DoS attacks.

The proposed hardline input data filtering security mechanism will operate in the network stack domain, but it will be capable of interrupting and terminating non-compliant processes. At OSI layer 1 [14], this mechanism will guard against Buffer-Overflow attacks and ensure that any alphanumeric data sent to the UAV will conform to a length of at most λ_A characters. Monitoring will also be done at the OSI layers 2 (data link) and 3 (network) to ensure that maximum lengths are not exceeded for packets and data frames. This will ensure that legitimate applications that are running on the UAV but have remote connections via the OSI layers 2 and 3 are compliant with the input character length specified by λ_A.

In other words, no processes running on the UAV's embedded system that receive input data remotely, can accept payload data with a length longer than λ_A. The functionality of the proposed hardline input data filtering mechanism is

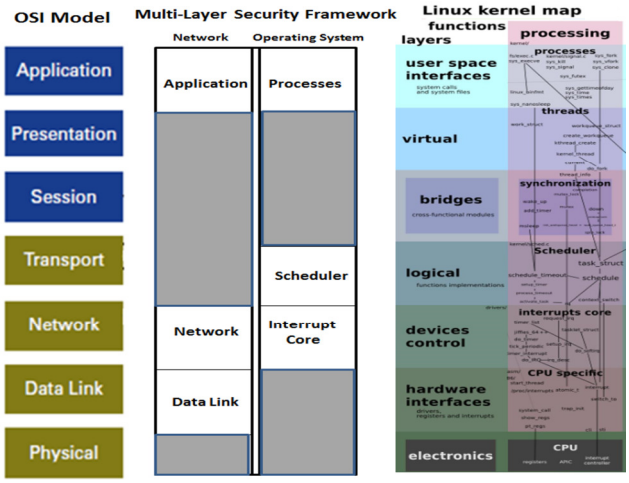

Figure 1. Multi-layer Security Framework

illustrated in Algorithm 2. Further testing should be done on the Bebop UAV to determine the character length to be specified in λ_A.

Finally, the anti-spoofing security mechanism will be implemented at the OSI Network and Data Link layers in the access point, and will be capable of dropping ARP replies with spoofed IP or MAC addresses. This will ensure that potential controllers do not corrupt the network with misinformation via an ARP Cache Poison Attack. Once the access point detects the JSON record (i.e., Table 2) sent by the UAV to enable the controller to access its control ports, it copies the UAV's ARP cache and uses the entries as the template for the UAV's IP address and MAC address and the valid controller's IP address and MAC address (Algorithm 3: lines 2-4). Now the access point can drop any ARP reply packets that contain the IP or MAC address of either the controller or the UAV (Algorithm 3: lines 5-17). This will ensure the integrity of the UAV and controller inter-networking. In other words, the OSI Layer 2 and 3 aware access point could record the IP and MAC addresses of the UAV and controller and ensure that no device on the network can spoof them. The functionality of the proposed anti-spoofing mechanism is illustrated in Algorithm 3.

VI. EXPERIMENTAL EVALUATION

In our experimental evaluations, we focus on the penetration testing of the controller-to-UAV inter-networking. Because we were able to find three zero-day vulnerabilities for the Parrot Bebop, it implies that there is a security issue with this aspect of the UAV and likely other Wi-Fi-based UAVs and underscores the need for this research.

A. Experimental Testbed

The Parrot Bebop is a P7 dual-core CPU and quad-core GPU embedded system running a Linux-based OS with a built in Wi-Fi AP, GPS, and camera. This UAV can be controlled by a smartphone. A break-away depiction of this UAV appears in Figure 2.

B. Experimental Procedure

In this work, our threat model is three common security attacks. We performed penetration testing on the UAV and developed exploits that allowed us to perform DoS, Buffer-Overflow, and ARP Cache Poisoning attacks against it. Therefore, our approach was to: (1) look for open ports in the UAV's embedded system using NMAP, (2) conduct routine flights of the UAV using the controller and capture the network traffic from the *attack laptop* using wireshark (Figure 3A), (3) analyze the captured network traffic to develop an initial fuzzing strategy, and (4) fuzz the controller-to-UAV inter-networking

Algorithm 1: Watch dog timer

Input: P_i - Non-navigational processes
Output: t - Runtime per P_i in timeframe γ
01: t=0
02: P_i -> low priority
03: **for** each P_i in timeframe γ **do**
04: **set timer t**
05: **if**($t > \tau$)
06: **interrupt** P_i
07: **end if**
08: **end while**
09: **for** each interrupted P_i **do**
10: **sleep** $2*\gamma$
11: **restore** P_i
12: **end while**
13: **return** t

Algorithm 2: Hardline input filtering

Input: F_i - Field from application, packet or frame.
Output: f - Number of characters per field
01: f=0
02: **for** each F_i **do**
03: **count each character** in F_i
04: f++
05: **if**($f > \lambda_A$)
06: **interrupt** P_i
07: **end if**
08: **end while**
09: **return** f

Algorithm 3: Access point anti-spoofing

Input: I_U & M_U – IP and correlated MAC address for UAV
I_C & M_C – IP and MAC of current controller
Output: 1 if I_i or M_i are spoofed
01: e=0
02: **if JSON c2d_port record** contains (status == 0)
03: **copy** UAV ARP cache
04: **end if**
05: **for** each I_i **do**
06 **if ARP Reply** contains ($I_i == I_U$ Or $I_i == I_C$)
07: **drop** corresponding packet
08: e=1
09: **end if**
10: **end for**
11:
12: **for** each M_i **do**
13 **if ARP Reply** contains ($M_i == M_U$ Or $M_i == M_C$)
14: **drop** corresponding packet
15: e=1
16: **end if**
17: **end for**
18: **return** e

From our initial penetration testing of the Bebop UAV, we discovered that the smartphone performs an ARP lookup for the MAC address of the device with the IP address of 192.168.42.1. Then, the smartphone sends a JSON record to IP address 192.168.42.1 on port 44444 on the UAV (See Table 1), which contains meta-data about the smartphone and setup parameters. Next, the UAV responds with a JSON record

either pointing the controller to port 54321 (See Table 2) or denying its request to become a controller with a different JSON record (See Table 3). In Sections VI.B.1 – VI.B.3, we discuss the three zero-day vulnerabilities that we found in the controller-to-UAV inter-networking.

1) Exploit #1: Buffer-Overflow Attack

Our fuzzing approach involved launching the UAV using a smartphone controller, and using a script on our *attack laptop* to one JSON record (with up to 1000 characters in the first field, i.e., port) requesting to become the controller for the UAV using the command: *telnet 192.168.42.1 44444 {Small to Very Large JSON Record}*. During the course of this experiment, the attacking computer captured network traffic and embedded system statistics from the */proc/stats* directory. We incremented the length of the entry in the first field in the JSON record by 1 starting with 926 until the UAV crashed (from repeated fuzzing we knew that the UAV would crash around 1000 characters), see Figure 3B.

2) Exploit #2: Denial of Service (DoS) Attack

Our fuzzing approach involved launching the UAV using a smartphone controller, and using a script on our *attack laptop* to send up to 1000 JSON records in parallel requesting to become the controller for the UAV using the command: *telnet 192.168.42.1 44444 {Small Replayed JSON Record} &*. During the course of this experiment, the attacking computer captured network traffic and embedded system statistics from the */proc/stats* directory. We incremented the number of simultaneous processes, which requests to establish a controller relationship with the UAV, by 1 (starting with 1) until the UAV crashed. (See Figure 3C).

3) Exploit #3: ARP Cache Poison Attack

Our ARP Cache Poisoning approach involved launching the UAV using a smartphone controller, and using a python library called *Scapy* in a script on our *attack laptop* to continuously send spoofed ARP replies to trick devices on the UAV's wireless network that our *attack laptop's* MAC address was at IP address 192.168.42.1. We executed this script until the smartphone controller was disconnected from the Bebop, see Figure 3D.

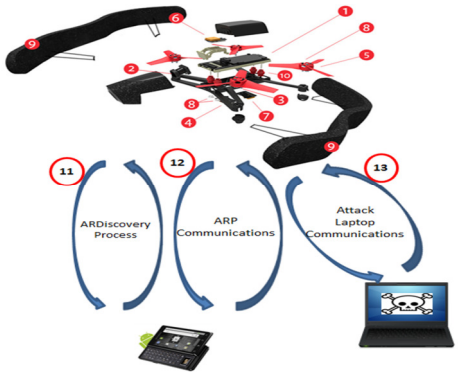

Figure 2. Experimental testbed: (1) CPU, (2) camera, (3) motors, (4) frame, (5) propeller, (6) GNSS radio, (7) Wi-Fi antennas, (8) motor controller, (9) bumpers, (10) anti-vibration rubbers, (11) ARDiscovery process, (12) ARP communications, and (13) attack laptop.

Figure 3. (A) Capturing data from normal flight, (B) Testing Buffer-Overflow Attack, (C) Testing Denial-of-Service Attack, and (D) Testing ARP Cache Poisoning Attack.

VII. RESULTS AND DISCUSSION

Our penetration testing of the controller-to-UAV inter-networking for the Parrot Bebop successfully demonstrated three zero-day vulnerabilities which allowed us to develop three common security attacks [17]. These results are a good indicator that Wi-Fi-based commercial UAVs are likely vulnerable and require the type of comprehensive security framework that we propose.

A. Exploit #1: Buffer-Overflow Attack

The results of fuzzing the ARDiscovery process, by increasing the number of characters in the first field in the JSON record sent by a potential controller, revealed that the Parrot developers never considered this scenario.

Therefore, while a valid controller was already flying the UAV, the UAV was incapable of handling JSON records sent from potential controllers with the first field larger than 931 characters UAV. Because we captured system statistics from the */proc/stats/* directory on the UAV, we were able to.

calculate the CPU and memory load of the UAV while our experiments were running. In Figure 4, we illustrate the baseline CPU usage of the Bebop as compared to the CPU usage of the Bebop during our fuzzing experiment. The UAV's CPU usage during our experiment is almost the same as the baseline except when the UAV accepts the large JSON record, which causes it to crash. At this point, the CPU usage drops below 10%. We assume this is because the UAV's navigational application has crashed. In Figure 5, which illustrates the memory usage, similar behavior to the baseline also occurs. Once the UAV receives the large JSON record, the memory usage drops below 10%, which probably means the navigational application has crashed. We believe that the input filtering capability we have identified in Algorithm 2 will defend this UAV against Buffer-Overflow attacks.

B. Exploit #2: Denial of Service (DoS) Attack

Fuzzing the ARDiscovery process with simultaneous requests from potential controllers revealed that the Parrot developers did not account for this scenario and thus the UAV

was incapable of handling up to 1000 simultaneous requests from other potential controllers while a valid controller was already flying the UAV.

Because we captured system statistics from the */proc/stats/* directory on the UAV, we were able to calculate the CPU and memory load of the UAV while our experiment was running. The fuzzing tests for this experiment was different from the fuzzing experiments in Exploit #1, but the CPU and memory usage behaviors were very similar to Figures 4 and 5 respectively for both the baseline and the fuzzing experiment. From our observations, it appears that as long as the UAV is flying, it appears to operate (from a CPU and memory usage perspective) the same, but when intrusive tests attempt to cause the UAV to deviate from the baseline CPU and memory usage, failure occurs. Shortly after our exploits are initiated against the UAV, we visually verify that the UAV ceases to fly and crashes, and both the CPU and memory usage drops below 10%. Based on these observed behaviors from the UAV, we infer that our exploits crash the UAV's navigational application. We believe that the watch dog timer capability we have identified in Algorithm 1 will defend this UAV against DoS attacks.

C. Exploit #3: ARP Cache Poison Attack

The result of our ARP Cache Poisoning experiment was a successful disconnection of the primary smartphone controller. When the *attack laptop* spoofs the UAV's IP address, there is a conflict on the UAV's wireless network, and thus no communication can occur between the primary controller and the UAV. After one minute of disconnection between the UAV and the controller, the UAV lands as stated in the manual [15]. We believe that the access point anti-spoofing capability we have identified in Algorithm 3 will defend this UAV against ARP Cache Poisoning attacks.

VIII. CONCLUSION AND FUTURE WORK

We believe that our security framework is a good step in the right direction of hardening the attack surface of Wi-Fi-based commercial UAVs. While this entire framework is in development, we believe that our descriptions of the security mechanisms for the controller-to-UAV inter-networking aspect of these UAVs is likely sufficient to address intermediate to medium security threats in these types of UAVs. Our security framework was driven by knowledge gained from our own penetration testing as well as the results from the penetration testing by other researchers from the literature. We discussed how Multi-layer Security Framework would defend UAVs from all of the zero-day vulnerabilities we found in this study. In future work we plan to perform penetration testing on other aspects of the Parrot Bebop UAV (i.e., GPS, the embedded system of the UAV itself, and its access point) and provide further details on how our Multi-layer Security Framework could be used to defend commercial Wi-Fi-based UAVs against future zero-day vulnerabilities.

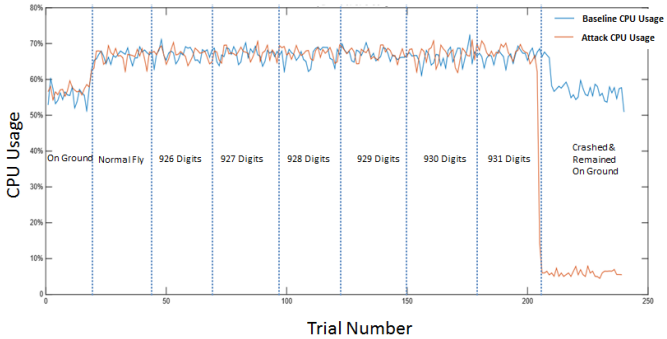

Figure 4. Baseline and attack CPU usage of Parrot Bebop

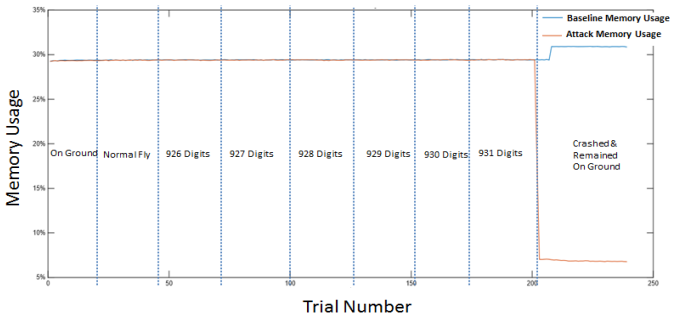

Figure 5. Baseline and attack memory usage of Parrot Bebop

IX. REFERENCES

[1] C. Anderson. (2012) "How I Accidentally Kickstarted the Domestic UAV Boom". *Wired.com.* Available: http://www.wired.com/2012/06/ff_UAVs/

[2] J. Booton, "UAV demand sizzles heading into the holidays," in *MarketWatch*, ed. [Online], 2014.

[3] "FAA needs to make thoughtful safety rules before UAVs deliver our packages or pizza," in *Los Angeles Times*, ed. [Online], 2015.

[4] S. Gallagher, "Parrot UAVs easily taken down or hijacked, researchers demonstrate," *Ars Technica,* 2015.

[5] D. Crockford, "The application/json media type for javascript object notation (json)," 2006.

[6] S. Kamkar. samyk/skyjack [Online]. Available: https://github.com/samyk/skyjack

[7] K. Moskvitch, "Are UAVs the next target for hackers?," in *BBC*, ed, 2014.

[8] J. Rinke, "Take Control of Paired ARUAV," ed. YouTube.com, 2013.

[9] E. Deligne, "ARDrone corruption," *Journal in Computer Virology,* vol. 8, pp. 15-27, 2012.

[10] J. Pleban, R. Band, and R. Creutzburg, "Hacking and securing the AR.UAV 2.0 quadcopter - Investigations for improving the security of a toy," in *The International Society For Optical Engineering*, 2014.

[11] S. M. Giray, "Anatomy of unmanned aerial vehicle hijacking with signal spoofing," in *Recent Advances in Space Technologies (RAST), 2013 6th International Conference on*, 2013, pp. 795-800.

[12] B. Brock and K. Rajamani, "Dynamic power management for embedded systems," in *IEEE International Systems-on-Chip (SOC) Conference*, 2003, pp. 416-419.

[13] "FreeFlight 3," ed: Parrot S.A., 2015.

[14] H. Zimmermann, "OSI reference model--The ISO model of architecture for open systems interconnection," *Communications, IEEE Transactions on,* vol. 28, pp. 425-432, 1980.

[15] Parrot Bebop User Guide. (2016) Available: http://www.parrot.com/usa/support/parrot-bebop-drone/.

[16] S. Institute, "Defense In Depth: SANS Institute Reading Room," SANS Institute, [Online] 2015.

[17] J. Burns. (2016) "Johns Hopkins Team Hacks, Crashes Hobby Drones To Expose Security Flaws" Forbes.com. Available: http://www.forbes.com/sites/janetwburns/2016/06/13/johns-hopkins-team-hacks-crashes-hobby-drones-to-expose-security-flaws/#23e0dc271fa1.