Protecting Unmanned Privacy (PUP)

Brian L. Uhlhorn[*] and Katrina A. Gilmore[†]
Lockheed Martin, Advanced Technology Laboratories, Eagan, MN, 55121, USA

The rapid proliferation of small Unmanned Aerial Systems (sUAS) with an increasing reliance on software and network connectivity introduces new opportunities for cybersecurity attacks. Cyber attacks can disrupt or disable sUAS, leading to catastrophic consequences, particularly in scenarios involving commercial sUAS, or other applications where human lives depend on safe operation of aircraft. Traditional cyber protection methods such as physical separation of systems and software patches will not be sufficient for sUAS. New methods of cyber resiliency will be imperative to the FAA's mission of safe and successful integration of sUAS into the national airspace. The PUP Program demonstrated Lockheed Martin's Hardware Architecture Resilience by Design (HARD) cyber protection capabilities, developed and demonstrated under the DARPA System Security Integration Through Hardware and Firmware (SSITH) Program. HARD is a novel cyber protection method deployed at the hardware level and has applicability to architectures where the operator sits at the edge of, or outside of the loop, and for systems that have a complex path to software certification.

I. Introduction

Cyber security protections are a critical component of a safe national airspace. As the complexity of small Unmanned Aerial Systems (sUAS) increases, aviation operations will become increasingly at risk to cyber threats. Unmanned, autonomous operations are prime targets for cyber-attacks due to their reliance on software, increased connectivity, and distance of the sUAS from a human-in-the-loop who can identify and respond in real-time to cyber-attacks. Additionally, increasingly complex, networked systems provide additional attack surfaces for malicious actors. Disruptions caused by cyber breaches and malicious exploits can be devastating, and today's cyber protection methodologies will be insufficient protection for the future of aviation.

Understanding the changing landscape of cyber vulnerabilities is the first step in protecting our National Airspace System (NAS). Today, cyber protections are typically reliant on a layered approach as shown in Figure 1. The layered cyber security approach includes supply chain protections, physical security, secure networks, anti-tamper software, software protections (including cyber-supportive architectures such as physical separation between systems), intensive certification of aircraft and their systems, and now hardware protections. Many users of electronic systems are familiar

[*]Research Scientist, Spectrum Systems Laboratory
[†]Program Management Manager, Spectrum Systems Laboratory

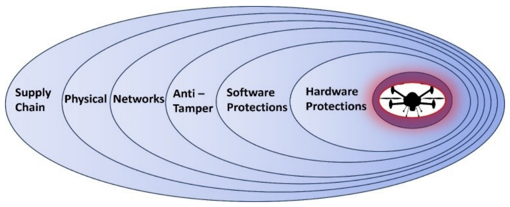


Fig. 1 A layered approach to cyber security supports protection of electronic systems.

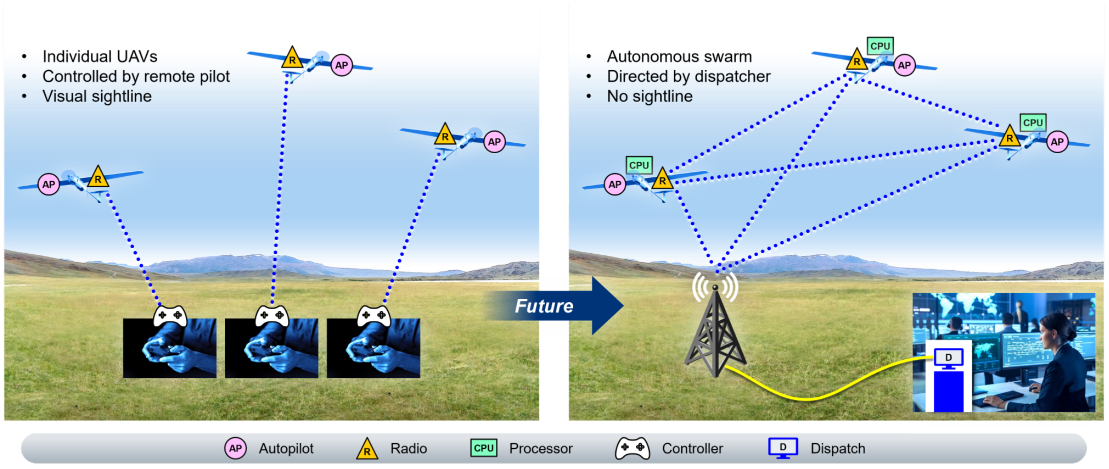


Fig. 2 Desired sUAS automation requires significantly more complicated software and networking.

with software protection or "anti-virus" software which is a single layer in the layered approach. Multiple layers increase the security of the overall system as each individual layer can focus on and protect against specific threats. Just like the "anti-virus" requires software updates, each layer also requires constant monitoring of threat evolution and consistent adaptation to threat advancements, meaning every security layer of a system typically requires updates in the form of physical security changes and software patches.

The costs, controls, and complexity of maintaining the current cyber protection policies used for manned aviation will not scale to the needs of unmanned aviation. As shown in Figure 2, advances in autonomy will necessitate an increased reliance on complex software and networked systems, with the operator moving further to the edge of the control loop. The current sUAS policy limits higher-complexity operations including operations Beyond Visual Line of Sight and Operation of Multiple Small Unmanned Aircraft that will increasingly rely on complex software and networked operations.

The need for additional cyber protections increase as changes in sUAS policy begin to support more complex commercial autonomous operations for sUAS and policy for optionally piloted aircraft and Advanced Air Mobility Aircraft move closer to reality. Cyber protections required for these newly emerging aircraft will necessarily be different due to the control of the supply chain, aircraft certification, and cost and quantity of the aircraft. The cost of implementing traditional cyber security protection methods must be weighed against the risk of deploying complex

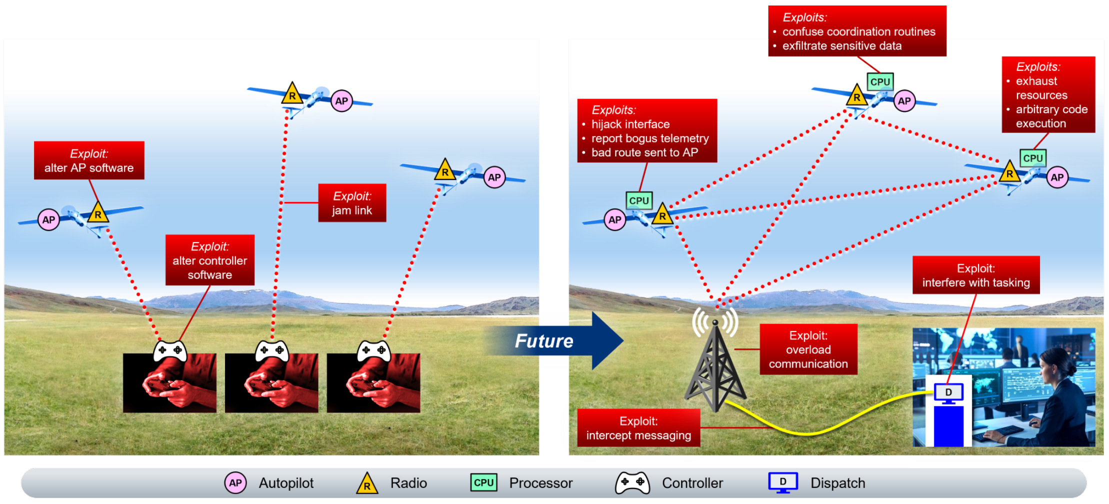


Fig. 3 The addition of processing and networking connections significantly increase the threat of cyber attacks.

software, processing components, and increased network connectivity that will result in additional available cyber-attack surfaces as displayed in Figure 3. Not only will the number of attack surfaces significantly increase, but the number aircraft will significantly increase, as will the number of types of aircraft, manufacturers, software developers, and operators, resulting in an exponential increase in cyber vulnerabilities. Necessarily, the aviation community's approach to cyber security must change in response to this increase in complexity.

The PUP program demonstrated Lockheed Martin's Hardware Architecture Resilience by Design (HARD) cyber protection capability, developed and demonstrated under the DARPA System Security Integration Through Hardware and Firmware (SSITH) Program. HARD is a new cyber technology that can provide hardware-integrated cyber security technologies for protection of sUAS systems. This foundational security mechanism could be a building block required for the safety assurances needed by the FAA and the public for safe sUAS operation. The DARPA SSITH program used operationally relevant aircraft, conducting live flight tests, with sUAS autonomy software that had been developed under separate programs to demonstrate the HARD cyber protection capability. The use of operationally relevant aircraft and independently developed software both support the validity of the demonstration.

Current regulations in the United States limit the ability to conduct Counter Unmanned Aerial System (CUAS) attacks, thus the PUP Team was restricted on where in the system architecture the cyber-attack could be conducted. For the PUP Program, the cyber-attack was focused on the aircraft dispatch system, whose hardware and software systems remain outside of the aircraft's control systems, consequently avoiding any crossover into CUAS actions. This methodology delivered a demonstration that showcased the technology while considering regulatory restrictions, the operation of the aircraft and the safety of the NAS as well as people and property on the ground. The detailed system architecture will be described later in this paper. It is important to note that to remain relevant, the cyber-attack used for the demonstration originates from a publicly known vulnerability in an existing, publicly available software

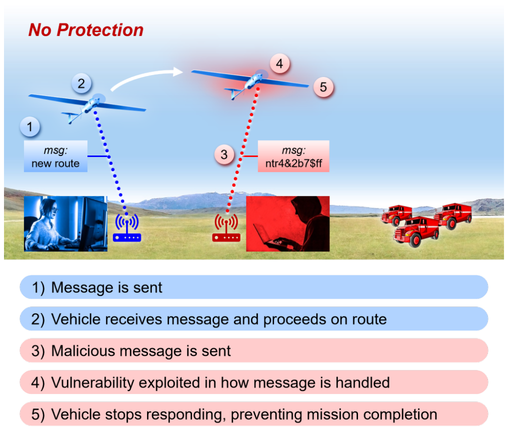

No Protection
1
2
3
4
5
msg: new route
msg: ntr4&2b7$ff
1) Message is sent
2) Vehicle receives message and proceeds on route
3) Malicious message is sent
4) Vulnerability exploited in how message is handled
5) Vehicle stops responding, preventing mission completion

Fig. 4 The PUP cyber scenario demonstrates an exploitation based on a message from a malicious actor.

library and was not artificially constructed or modified for this program. The approach of using a publicly identified vulnerability with an available patch avoids introducing new vulnerabilities into the aviation ecosystem that may occur when developing a custom "bug" solely for the purpose of the demonstration.

The attack deployed in the PUP field demonstrations follow the process as shown in Figure 4. In the first step, a "normal" dispatch message is sent to the Mosaic Warfare dispatch system which is processed correctly and forwarded to the vehicle. In step two, the vehicle proceeds on the route described by the message. In the third step, a red force actor sends a malicious message to the Mosaic Warfare dispatch system. The Mosaic Warfare dispatch system then processes the malicious message which reaches and exploits the vulnerability in the unpatched Mosaic Warfare library in step four. The exploit results in unplanned execution and the Mosaic Warfare dispatch system stops responding in step five, preventing mission completion. This scenario was demonstrated first on a Stalker military sUAS and then moved to a more commercially relevant airframe. This transition reduced the risk by starting with the airframe used under the DARPA program, and the moving to a more ubiquitous airframe more relatable to civilian unmanned aircraft. Additional details about this transition are described in the Section II on Methodology.

II. Methodology

Demonstration of the HARD cyber capability first took place on a swarm of two autonomous Stalker sUAS, and then transitioned to a flight of two quadcopters. This stepped approach provided two benefits: (1) Enabled demonstration on a military sUAS (for which the system was designed) and a commercial sUAS; and (2) Provided insight into how the system may transition to other diverse applications. The Stalker was chosen as the demonstration platform for the initial PUP event because the PUP team had deep knowledge of the Stalker software, ground control station, and autopilot. This prior experience reduced both engineering development time and operational risks.

As discussed previously, the team was not authorized to conduct a CUAS mission, and thus was limited to attack

Table 1 Stalker UAS Specifications

		The Lockheed Martin Stalker XE25 is a small, quiet, unmanned aerial system (UAS) that provides unprecedented long-endurance imaging capability in a variety of environments. The aircraft is powered by an electric motor and can travel up to 60 miles on 1 charge in optimal weather conditions	
Size	144 inches	Cruise Speed	34-45 mph
Height	16 inches	UAS Operator	Fly Precision
Maximum Takeoff Weight	28.5 lbs.	GCS Type	Laptop with Proprietary GCS system
Endurance	180 minutes	Autopilot	XCOM (Proprietary)
Line of Sight Range	2 kms	Maximum Altitude	15,000 ft MSL

surfaces outside of the aircraft system. As a result, the software mission system used for the demonstration was the LM ATL-developed Mosaic Warfare autonomous mission tasking or “dispatch” software. Mosaic Warfare dispatch system allows a user to deploy a swarm of sUAS which autonomously bid for target tasking based on attributes such as distance to target and sensor payload type. When an aircraft wins a bid, the aircraft tasks its own autopilot to route to the requested location, without requiring an operator in the loop. A detailed hardware and software architecture are discussed in future sections.

The second demonstration was performed using a commercial quadcopter, the FreeFly Alta-X. This airframe was chosen to validate the applicability to commercial sUAS which are more relevant to the FAA. The progression from the Stalker to the FreeFly Alta-X supported the demonstration of a commercially relevant platform and scenario; and supports the FAA’s mission of UAS Integration.

The following subsections provide overviews of the Stalker platform, the quadcopter platform, and how the cyber scenario described in Section I, Figure 4 applies to their usage. The payload hardware and software architectures used in the demonstrations for this program are described in the next subsections. A high level description of how the HARD approach is able to detect the exploit of software flaws is provided in the HARD Summary subsection. The next subsection delves into the public vulnerability and how a malicious actor may exploit it. Finally, the scenarios used demonstrate the capability and collect performance data are describe in the last subsection.

A. Stalker Demonstration Overview

The initial Stalker Demonstration was focused on demonstrating the HARD capability onboard the military aircraft used in SSITH and modifying the payload to accommodate PUP-specific requirements including accounting for the weather. Additionally, this demonstration provided additional data analysis that both supported a deeper understanding of the current capability and enabled the comparison under a future transition to a commercial platform. Specific details about the Stalker platform can be found in Table 1.

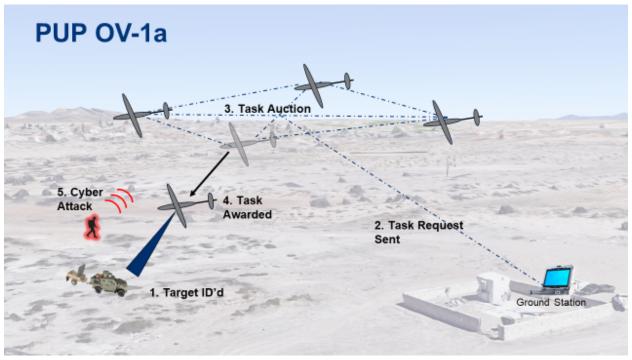


Fig. 5 A military intelligence, surveillance and reconnaissance scenario demonstrates the potential impact of a cyber-attack on a sUAS.

In this demonstration, two Stalker sUAS were launched in support of an intelligence, surveillance and reconnaissance mission. Figure 5 shows the steps of the scenario. First, the swarm of aircraft deployed to a loiter point and wait for tasking. The operator received an intelligence request for target identification and then injected a task into the swarm via the mission system, asking the swarm to deploy and aircraft to conduct a surveillance mission. In the third step, the swarm received the task and the aircraft within the swarm bid for the task based on attributes including distance to the target location, fuel remaining, and sensors required. Once the task was awarded to an aircraft, the aircraft proceeded to the task. In step 5, as aircraft were servicing tasks, a cyber-attack was initiated, causing the Mosaic Warfare dispatch system to permanently lose connection with the attacked aircraft. In this case, the pilot still had full manual control of the aircraft as the attack was purposefully limited to only the dispatch system. The result was an inability to complete the tasking or to re-task the attacked aircraft.

B. Quadcopter Demonstration Overview

The Quadcopter Demonstration scenario was designed to mature the demonstration to be representative of a sUAS operation more relevant to the FAA. In this demonstration, the scenario was based on a Drone as a First Responder (DFR) mission, where emergency services deploy a UAV to a scene prior to the arrival of uniformed officers. The FreeFly Alta-X was used as the commercial demonstration vehicle for this exercise, details about the FreeFly Alta-X can be found in Table 2.

An operational view of the Scenario is shown in Figure 6. Two DFR assets are launched in response to a first response call. The two airborne assets are dispatched to two positions, a primary position located at DFR #1 (the front entrance of a building), and a secondary position at DFR #2 (a rear entrance). A cyber-attack at DFR #1 results in Dispatch losing access to that aircraft. DFR #1 is a higher priority location and as a result, when that aircraft loses connection, the secondary aircraft manuevers to the primary position. This cause/effect was known to the attackers, so as DFR #2 moved into the first position, attackers entered through the unprotected door and breach the entrance of the building. The malicious attack resulted in the inability to communicate with the exploited aircraft from the Mosaic

Table 2   FreeFly Alta-X Quadcopter Specifications

		The FreeFly Alta-X is an unmanned aerial system is designed to satisfy the needs of professional and industrial partners. The Alta-X was designed from the ground up to achieve the perfect blend of efficiency, agility, and reliability while keeping the pilot workflow simple and fast	
Width	89 inches	Cruise Speed	60 mph
Height	15.2 inches	UAS Operator	Lockheed Martin
Maximum Takeoff Weight	55 lbs.	GCS Type	Laptop QGC
Endurance	30 minutes (10lb payload)	Autopilot	Custom PX4
Line of Sight Range	2 kms	Maximum Altitude	Varies See OEM Specs

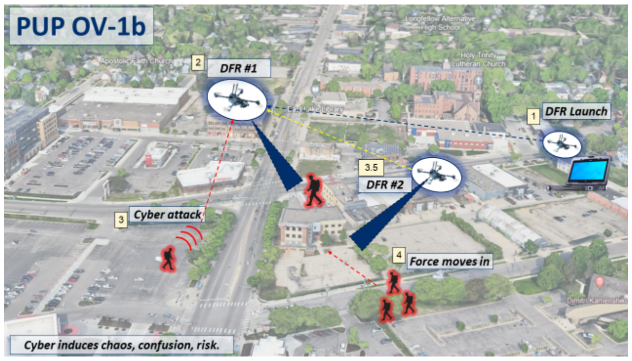


Fig. 6   Drone as a First Responder Scenario demonstrates the potential impact of a cyber-attack on a UAS.

system, the failure of the video feed, and DFR #2 leaving its position, culminating in a failure to protect part of the building. This scenario demonstrates the described software vulnerability is present in the software and exploiting the vulnerability can disrupt operations. As with the Stalker demonstration, the remote pilot in command retained full manual control of the aircraft throughout the scenario, including during and after the cyber-attack.

C. Hardware Architecture

Both the Stalker and the FreeFly Alta-X aircraft use a similar hardware architecture. The relevant aircraft system components consist of an onboard computer hosting the aircraft flight controller and autopilot, and a network switch connecting the aircraft computer to peripherals. Neither aircraft hardware architecture were modified from their original state.

The Mosaic Warfare dispatch capability resides in hardware that is physically separated from the Stalker and FreeFly Alta-X the aircraft system components. A nominal or "unprotected" aircraft uses an NVIDIA Jetson Xavier processor to host the Mosaic Warfare software with a power conversion board to power the processor from the aircraft power if necessary. A protected aircraft has an additional processor to host the protected capability. In the case of the Stalker, a

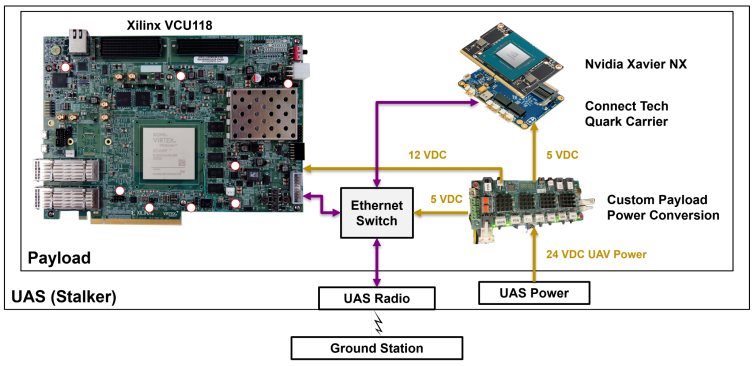


Fig. 7 The Stalker payload supporting the Mosaic Warfare system consists of 2 compute boards (Xilinx FPGA and Jetson Xavier), an Ethernet switch, and a custom power conversion board.

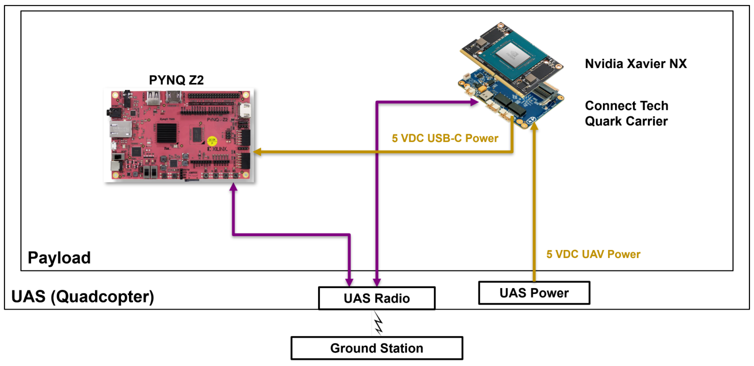


Fig. 8 The quadcopter payload supporting the Mosaic Warfare system consists of 2 compute boards (PYNQ Z2 and Jetson Xavier) and an Ethernet switch.

VCU118 FPGA hosts the protected capability. In the case of the FreeFly Alta-X, a PYNQ Z2 FPGA demonstrates the protection on a commercial-off-the-shelf component. The rationale for the Mosaic Warfare software separation between the Jetson and the FPGAs is defined in greater detail in Section II.D. The Quadcopter also includes a camera to support the Scenario described in subsection II.B.

A system diagram of the hardware components for each aircraft type is documented in Figure 7 and Figure 8 with the Mosaic Warfare system in the block on the top, and the aircraft system in the containing block. The Mosaic Warfare hardware interfaces with the aircraft hardware system in two ways: (1) the Mosaic Warfare hardware is powered by the host aircraft (Stalker or FreeFly Alta-X) through a direct connection to the aircraft's power source or through a custom power conversion board connected to the aircraft's power source; and (2) the Mosaic Warfare communicates through Ethernet either through a switch or direct connection depending on the aircraft configuration. The cyber-attack does not travel through either of these interfaces to the aircraft system.

The Mosaic Warfare payload for the Stalker UAV resides in both an external payload bay on top of the nose cone as shown in Figure 9 and in the internal bay in the body of the vehicle. The Mosaic Warfare payload for the quadcopter resides in the cargo net volume of the FreeFly Alta-X as shown in Figure 10.

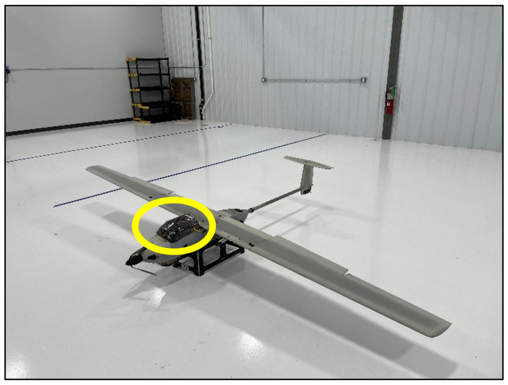


Fig. 9 The Lockheed Martin Stalker UAV supports external payload bays, shown as the black compartment on top of the nose cone.

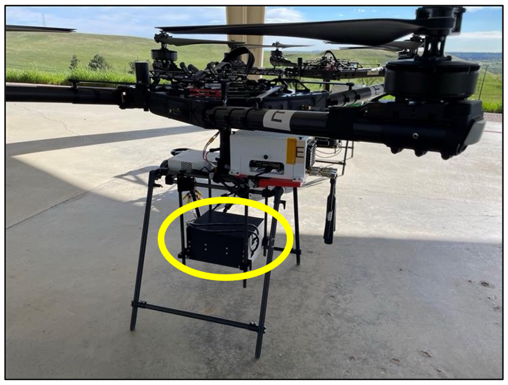


Fig. 10 The FreeFly Alta-X cargo attachment supports rapid payload integration.

D. Software Architecture

The software architecture mirrors the hardware architecture and consists of two independent systems: the aircraft software and the Mosaic Warfare software. Both systems operate independently with a few exceptions discussed in the following sections. As was the case with the hardware architecture, this demonstration showcases the applicability of the HARD technology to UAVs, but the software exploited in this exercise resides completely in the Mosaic Warfare software architecture, separate from the aircraft software system. The software architecture for both the Stalker and FreeFly Alta-X UAVs consists of two major components, the aircraft autopilot and the Mosaic Warfare dispatch system. The aircraft autopilot supports the function of the aircraft including control and providing telemetry data. The Mosaic Warfare dispatch system enables a user on the ground to control the UAV through a user interface, supporting both manual assigned missions and automated mission assignment. The Mosaic Warfare dispatch system may be bypassed entirely to let a pilot on the ground take full control of the aircraft.

The Mosaic Warfare Software System includes four software modules plus a ROS Master node, and a user interface co-located with the aircraft ground control station. The user interface (UI) for the Mosaic Warfare software system is co-located with the aircraft ground station. The Mosaic Warfare UI allows an operator to inject the task requests into the aircraft swarm. The UI also receives telemetry data from all of the participating aircraft and displays it on its map, providing situational awareness of the aircraft and their tasking. The Mosaic Warfare modules are implemented on the the Jetson Xavier NX and either the VCU118 or PYNQ Z2 hardware as shown in Figure 11.

In the same way that physical structures can receive additional security protections based on vulnerabilities and potential impact if exploited, software can also receive additional security protections. The PUP Program chose to split the Mosaic Warfare modules across the two compute boards to accommodate the capabilities of the various processors, and to physically demonstrate the ability to protect mission critical modules separately from less critical modules, in an efficient and effective manner. The Mission Executive, Replan Monitor, Clusstar Wrapper, and ROS Master modules

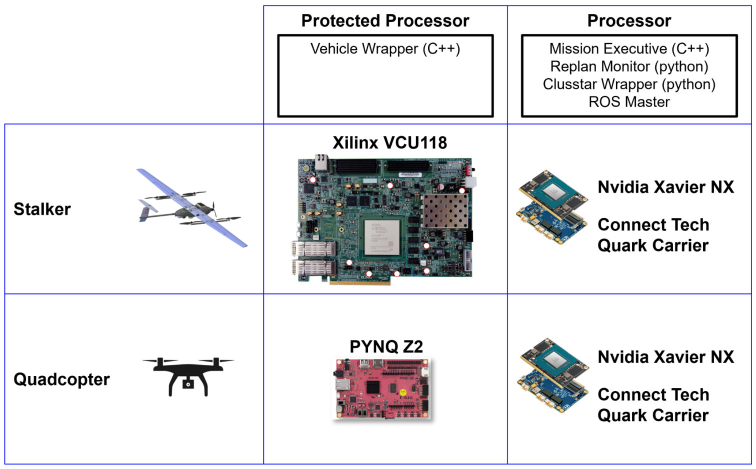


Fig. 11 Mosaic Warfare software modules are split across the VCU118/PYNQ and the Jetson Xavier NX to support protection of safety-critical modules.

perform administrative tasking related to bidding for tasks and managing communications which are necessary for operation of the system. In contrast, the Vehicle Wrapper module sends the autopilot the dispatch request once the tasking bid has been awarded to the aircraft. The Vehicle Wrapper's ability to send altered and potentially dangerous messages (e.g. controlled flight into terrain) identifies the Vehicle Wrapper as a safety critical module to be protected by HARD. As previously mentioned, the exploit selected for the demonstration will not alter the message to the autopilot or cause a safety-of-flight concern for the aircraft. The exploit is further discussed in a following section.

E. Cyber Exploit

The planned exploit is the final component of the PUP cyber scenario demonstration. The Mosaic Warfare software system was made vulnerable through the addition of a feature from an unpatched, vulnerable library taken from a real-world example, a Network Timing Protocol Query (`ntpq`) bug in the publicly available Network Timing Protocol (`ntp`) library. The `ntp` library used in the cyber scenario demonstration is version 4.2.8p11 which contains the CVE-2018-12327 vulnerability, currently available on the Internet at `https://www.ntp.org/downloads/`. As of the date of this document, publicly available `ntp` library versions 4.2.8p17 and later have patched this known vulnerability. The PUP Team used an unpatched version of the library for its unprotected systems. The team elected to use a known bug with an available patch to avoid introducing new cyber concerns into the broader ecosystem.

The `ntp` library enables logging a timestamp originating from a network time protocol server. This feature is important and often used because individual vehicles in the swarm may have slightly different clock settings either due to initial starting values or due to clock drift. Using a single time source available on the network allows the logs from the different vehicles to be accurately synchronized for post mission analysis.

The vulnerability exists because a string with a limit of 256 characters is not checked before the string is passed to the next routine. This vulnerability exists in the openhost routine used by `ntpq` of the unpatched, vulnerable `ntp` library. The openhost routine allows hostname strings to be passed with or without opening, [, and closing,], square brackets. The openhost routine strips off these enclosing square brackets if they are present and then treats the stripped hostname string as a hostname string bare of these enclosing brackets. In the unpatched version of the `ntp` library, the code stripping the square brackets does not check the size of the hostname string, which was previously defined as a maximum of 256 characters. A string longer than 256 characters will result in a buffer overflow and will overwrite the existing values on the stack, potential altering overhead operating parameters. For example, an attacker may overwrite the return address with an address chosen to execute an inappropriate, unexpected, or planted portion of code. The unintended code could send the aircraft into controlled flight into terrain or force the aircraft to land.

In the case of the demonstrations conducted under the PUP Program, the buffer overflow was crafted such a way that the effect on the system was contained within the Mosaic Warfare dispatch software. An exploit on an unprotected system will cause the onboard Mosaic Warfare system to lose connection with the UI, resulting in Mosaic Warfare operator being unable to task the aircraft or track it on the Mosaic Warfare UI. The remote pilot in command will continue to retain control of the aircraft through their ground control system.

F. Exploit Methodology

It is important to reiterate that the demonstration was constructed such that the core capability could be exercised within current regulations, and without causing undue risk to the operation, to the UAV, or to the broader aviation ecosystem. As such, the exploit and its effect were limited to only the Mosaic Warfare system, and the scenario assumed the attacker already had network access to the software system

The vulnerability in the `openhost` routine used by `ntpq` of the vulnerable `ntp` library discussed in Section II.E is reached in the Mosaic Warfare software by passing a specifically crafted ROS message to an unprotected Mosaic Warfare vehicle. An example of a malicious ROS message is shown in Table 3. In this example, the value of the timehost field is set on line 6. For ease of display, the full number of filler characters are not shown and instead represented by "<filler characters>" in the table. In this case, there are a number of characters beyond the 256 character limit discussed in the previous section. Depending on compiler and the overwrite target, the attacker may use more, or fewer filler characters to land on the desired portion of the stack. In this case, approximately 610 filler characters are used to line up the remaining characters with the return address of the openhost routine. The remaining characters, `LO\u005cu0003`, were selected by the attacker to write a specific target address into the return address location on the stack.

Once the unpatched `openhost` overflows the buffer with the filler characters, these final three values are written in reverse order over the return address, changing the return address from the appropriate value to the value desired by the attacker. This target address could be some arbitrary function or instruction within the program, or other values in memory may be reached. As discussed elsewhere for this demonstration, the impact of the attack was limited to just the

Table 3 JSON listing of a malicious ROS message for the cyber scenario.

```
{
   "name": "Tasking",
   "type": "Route",
   "geometry": ["x": 0.0, "y": 0.0, "z": 0.0],
   "metadata": "{
      \"vehicle_id\": 1,
      \"timehost\": \"[aaaa< filler characters >aaaaLO\u005cu0003]\"
      . . .
   }",
   "uuid": "0"
}
```

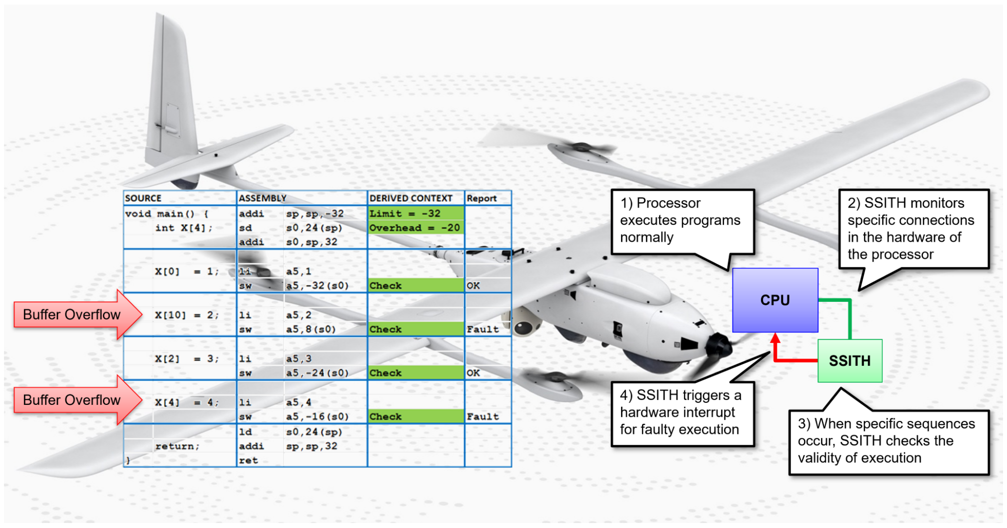
operation of the Mosaic Warfare dispatch system, but those not bound by regulatory restrictions may further craft the values written to the stack for more elaborate and advanced attacks

In the SSITH Program, the team used all available HARD techniques to monitor and respond to cyber attacks. However, due to the nature of the `ntpq` vulnerability, it was determined that the HARD technique could be adapted to only need to monitor the level 1 activity on the PYNQ board. Essentially to exploit the vulnerability, the attack forces the program to jump through a specific sequence of the code. This sequence would be revealed by monitoring the program counter. The level 2 and 3 information would further reenforce the detection but would require more resources. Hence, the monitoring pipeline was constructed to only use level 1 data but approach the reporting threshold cautiously, saving significant resources. Depending on the platform requirements, resources available and the larger cyber system implemented on the platform, the level of information used for monitoring may be customized as well as the aggression on determining the reporting threshold.

G. HARD Summary

Lockheed Martin's HARD cyber protection capability deploys light weight out-of-band modules to monitor the state of the processor against sequences of expected and problematic instructions on relevant hardware, as displayed in Figure 12. HARD monitors specific information about the execution of the processor core which allows the HARD modules to run in tandem with the processor and not interfere with normal operation. Each module filters incoming states for a predefined pattern consisting of program counters, instructions, registers used, register data, and write back data. This information is used to identify whether an exploit is being attempted or if the operation is within normal parameters. In the simplistic example shown in Figure 12, the system detects a Buffer Overflow. Upon fault detection the process is interrupted, and a report is generated providing the program counter, instruction, and pattern that triggered the interrupt. When the process is interrupted, the UAV is placed in stable operation while the HARD implementation brings the offending process back into a safe state.

The VCU118 board hosts a Xilinx XCVU9P-L2FLGA2104E FPGA in which a custom soft Arm Cortex-A55 was implemented. As the Arm Cortex-A55 was implemented in an FPGA, the processor was modified to expose all information of the processor state to the HARD detection techniques. A pipeline was created using SSITH HARD techniques specifically for the `ntpq` bug. The NTPQ pipeline monitored for a sequence of 19 processor states which


Buffer Overflow
Buffer Overflow
1) Processor executes programs normally
2) SSITH monitors specific connections in the hardware of the processor
CPU
SSITH
4) SSITH triggers a hardware interrupt for faulty execution
3) When specific sequences occur, SSITH checks the validity of execution

SOURCE	ASSEMBLY		DERIVED CONTEXT	Report
void main() {	addi	sp,sp,-32	Limit = -32	
int X[4];	sd	s0,24(sp)	Overhead = -20	
	addi	s0,sp,32		
X[0] = 1;	li	a5,1		
	sw	a5,-32(s0)	Check	OK
X[10] = 2;	li	a5,2		
	sw	a5,8(s0)	Check	Fault
X[2] = 3;	li	a5,3		
	sw	a5,-24(s0)	Check	OK
X[4] = 4;	li	a5,4		
	sw	a5,-16(s0)	Check	Fault
	ld	s0,24(sp)		
return;	addi	sp,sp,32		
}	ret			

Fig. 12 Lockheed Martin's HARD capability addresses computer weaknesses at the hardware level, only interrupting when a problem is detected.

occur just after the `ntpq` routine has stripped off the square brackets. The instructions and write back data of this sequence is examined, revealing the size of the hostname string. If the size of the hostname string is bigger than the what the code can handle, the HARD pipeline reports the event as problematic. Another pipeline was creating using SSITH HARD techniques which monitors more generally for the effect of the `ntpq` bug. The Generalized pipeline monitors for the entrance and exit to the `openhost` function in the `ntp` library. When the entrance is detected, the signature of return address pushed to the stack is noted. When the exit is is detected, the signature of the popped return address is compared to the noted signature of the pushed signature. If the signatures compare favorably, then the return address has not been altered. If the signatures are different, the return address has been altered, most likely by the buffer overflow contained within the `openhost` function, and the HARD pipeline reports the event as problematic.

The PYNQ Z2 board uses a commercially available system on a chip (SoC) for its processor, a Xilinx ZYNQ XC7Z020-1CLG400C. As this SoC is in production, no further modifications at the gate level may be made to the embedded processor itself. This means monitoring the activity occurring on the processor requires reassembling the information available on the implemented ports (trace, coherency, AXI, etc.). From these implemented ports, it is easiest to recover the program counter (level 1), more difficult to recover the instruction (level 2), and most difficult to recover the register file data (level 3). A pipeline was created based on SSITH HARD techniques after inspection of the `ntpq` bug revealed that level 1 information would be sufficient for detection. A unique set of branch program counters indicate the execution has reached the `openhost` function within the code. The section of code stripping off the square brackets has a particular branch program counter indicating a single character of the hostname string has been processed. The NTPQ(PYNQ) pipeline monitors the program counters until it determines the `openhost` function has been reached and then counts the loops through the single character processing code. Once the count has exceeded the number of characters allocated in the code, the HARD pipeline reports the event as problematic. The PYNQ Z2 runs modern multitasking operating system meaning that the Mosaic Warfare system program may be swapped in and out of the processor to allow other programs to execute. This means that the level 1 information may capture this process swap in progress. The reaction of the HARD pipeline to this swap process could be aggressive or cautious. In the aggressive case, the HARD pipeline would immediately flag any swap as hostile, making the assumption that a malicious attacker has forced the swap to mask the attack on the `ntpq` bug. This aggressive approach may result in additional false positives, reporting more detections than actual malicious attacks which would require either additional effort to sort through the detections or additional delays as all detections are addressed. In the cautious case, the HARD pipeline would reset the loop count upon a process swap, making the assumption that the swap happened by chance within the loop of the `ntpq` bug. This cautious approach may result in some false negatives as there would be some non-zero probability that a malicious attack occurred at the same time as a process swap. These two reactions are the extreme cases which have the most straight-forward and simplistic implementation, but the aggression may be tuned with more complicated pipelines or with a higher level of information. The NTPQ(PYNQ) pipeline was engineered to be the most cautious case to emphasis the extreme reaction and least impactful physical implementation.

H. Demonstration Scenarios

The scenarios for the Stalker and FreeFly Alta-X based demonstrations are detailed in this section with a overview of the behavior expected. The scenario without protection is used as a baseline to compare the results from the scenarios with protection: patched and HARD protected.

1. No Protection

In this scenario, the PUP team demonstrated exploiting the Mosaic Warfare system by executing a cyber-attack that resulted in the disruption of the Mosaic Warfare system. This demonstration used the unprotected `ntp` library with the `ntpq` bug as described in Section II.E. The result was the inability to communicate with the exploited aircraft from the Mosaic Warfare UI system. In the case of the FreeFly Alta-X, this also resulted in the failure of the FreeFly Alta-X's video feed, and DFR #2 leaving its position, resulting in a failure to protect part of the building. This scenario demonstrated that the described software vulnerability is present in the software and that it can be exploited to disrupt operations. Once this scenario was complete, the aircraft's Mosaic Warfare system was reset and the aircraft either proceeded to the next scenario or returned to land.

2. Patched

In this scenario, the PUP team demonstrated mitigation of a cyber-attack via a software patch. The patch onboard the aircraft changed the software code in the system, removing the weakness and resulting in an inability to exploit the software. When the exploit attempt was initiated there was no impact to aircraft operations. This scenario demonstrates the option of changing the software to resolve vulnerabilities, once the vulnerability is identified and a patch is available. After the completion of the scenario, the aircraft either proceeded to the next scenario, or returned to land.

3. HARD Protected

In this scenario, the PUP team demonstrated mitigation of a cyber-attack through a HARD pipeline that specifically addressed the `ntpq` vulnerability. When the exploit was initiated, the cyber capability onboard the aircraft detected and halted the operation in progress, stopping the exploit. The result was a momentary (approximately 2-3 second) pause in the aircraft reporting position information to the Mosaic Warfare UI system. In the case of the FreeFly Alta-X, this also resulted in a momentary disruption to the video feed while the HARD implementation restarted the interrupted process. This scenario demonstrated the option to not change software (whether due to lack of patch, or regulatory process) and instead inserted a hardware pipeline to prevent exploit. After the completion of the scenario, the aircraft either proceeded to the next scenario, or returned to land.

III. Results & Discussion

A. Demonstration Data Collection

The PUP Program used a series of exploits against the `ntpq` vulnerability, demonstrating the potential range of impacts of a cyber event. Each attack was performed either on an unprotected aircraft, a protected aircraft, or a patched aircraft. The attack and results were collected during flight operations.

- The unprotected aircraft provides a baseline of what the operator of an unprotected system would experience, proving the technical efficacy of the attack and demonstrating the resulting disruption. These attacks made up approximately $1/4$ of the scenarios executed.
- The protected aircraft hosts the HARD module that identifies and mitigates an attack. The attacks against the protected aircraft prove out the capability and the impact to the system. These attacks make up approximately $1/2$ of the scenarios executed.
- The patched aircraft hosts the publicly available patch for the `ntpq` vulnerability. This scenario demonstrates the efficacy of the software patch against the attacks. This attack made up approximately $1/4$ of the scenarios executed.

B. Demonstration Data Analysis

The PUP program collected data from the demonstration itself but also inherent data of the security implementation. The analysis of this data allows comparison between approaches and the impact of the added security capabilities.

Event Detection Ratio The HARD pipelines are part of a holistic response system where the HARD pipelines monitor for suspicious or anomalous low-level behavior. These events serve as information for a cyber protection decision system, which determines the appropriate response to protect the platform. A response system is designed to accept a range of information, but typically the ratio of reported events to actual events should be close to one. This metric is measured as the ratio of identified attacks vs exploits.

Performance The introduction of HARD pipelines to a system may impact the the processor performance when compared to an unprotected system. A difference in execution timing of the `ntpq` function will characterize this performance impact. In order to time a query to this function, a high-resolution clock was initialized before the method was called, and time elapsed calculated upon successful print of the reference clock to the log. The resulting metric is the statistical difference in mean timing between unprotected and protected hardware.

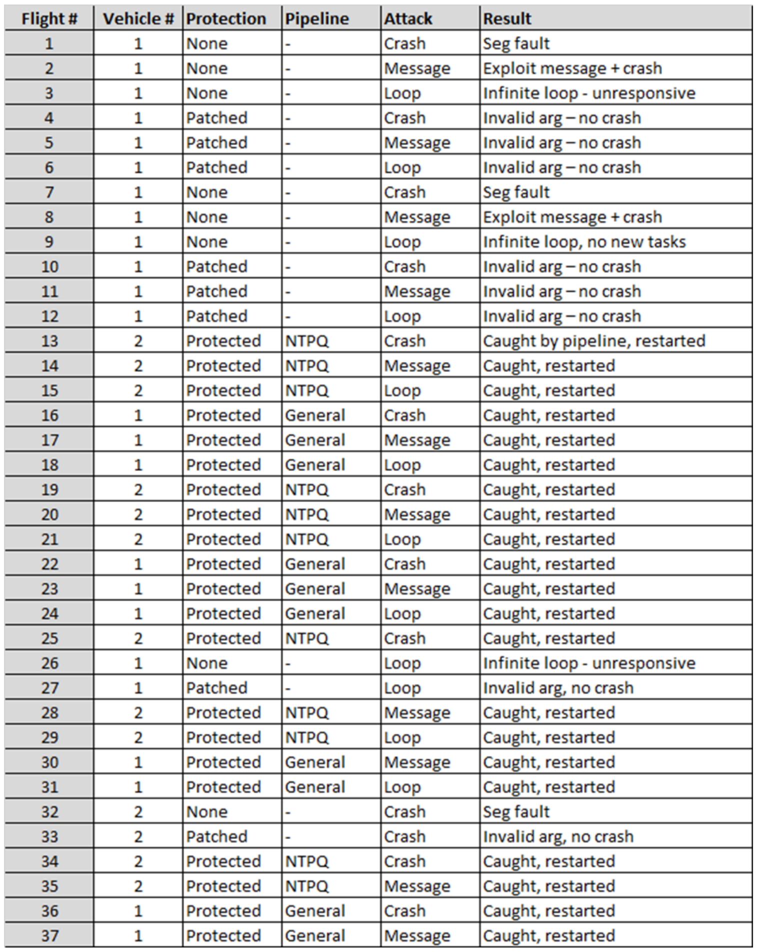
Physical Impact The HARD pipelines each require a finite amount logic to implement. The area (weighted primitive/low-level estimation) consumed by the logic is tracked as even relatively small measurements can ultimately impact the system. This metric focused on the increase in logic of the HARD pipelines as compared to the soft Arm Cortex-A55 and specifically the look up tables (LUTs) used. The LUT is typically used as a common reference in capability and capacity between various FPGAs. The LUTs provided in this report are normalized to a 4 value LUT, LUT4.

1. Stalker

The record of cyber-attacks on the Stalker, the various configurations, and the results are shown in Table 4. The Stalker used two different HARD pipelines for protection in the various flights. The first pipeline, labeled NTPQ, is a technique customized specifically to detect the `ntpq` bug. The second pipeline, labeled General, is a more generalized technique to detect exploits similar to the `ntpq` bug. The General pipeline leveraged another technique developed in the SSITH program and was able to be used in the PUP demonstration because the underlying VCU118 hardware and soft Arm Cortex-A55 was the same. The Stalker demonstration scenario used three different types of attacks: crash, message, and loop. The crash attack simply overflowed the buffer using the `ntpq` bug and overwrote the return address with an invalid program counter, causing the vehicle wrapper software module to crash. The message attack crafted the message such that the buffer overflow overwrote the return address with a function selected by the attacker which output a specific message indicating that the attack had the ability to reach an arbitrary function and then crash the vehicle wrapper software module. The loop attack crafted the message such that the buffer overflow overwrote the return address to cause the vehicle wrapper software module to enter into an infinite loop, consuming all available resources and bogging down the hardware executing the vehicle wrapper software module. The NTPQ pipeline was able to detect 11 of the 11 events corresponding to a 1.0 event detection ratio. The General pipeline was able to detect 10 of the 10 events corresponding to a 1.0 event detection ratio.

Figure 13 shows the timing data collected across multiple trials for configurations with no pipeline, the NTPQ pipeline, and the Generalized pipeline. The difference between the implementation with no pipeline and the NTPQ

Table 4 Stalker Demonstration - Record of Cyber-attacks and Results.



Flight #	Vehicle #	Protection	Pipeline	Attack	Result
1	1	None	-	Crash	Seg fault
2	1	None	-	Message	Exploit message + crash
3	1	None	-	Loop	Infinite loop - unresponsive
4	1	Patched	-	Crash	Invalid arg – no crash
5	1	Patched	-	Message	Invalid arg – no crash
6	1	Patched	-	Loop	Invalid arg – no crash
7	1	None	-	Crash	Seg fault
8	1	None	-	Message	Exploit message + crash
9	1	None	-	Loop	Infinite loop, no new tasks
10	1	Patched	-	Crash	Invalid arg – no crash
11	1	Patched	-	Message	Invalid arg – no crash
12	1	Patched	-	Loop	Invalid arg – no crash
13	2	Protected	NTPQ	Crash	Caught by pipeline, restarted
14	2	Protected	NTPQ	Message	Caught, restarted
15	2	Protected	NTPQ	Loop	Caught, restarted
16	1	Protected	General	Crash	Caught, restarted
17	1	Protected	General	Message	Caught, restarted
18	1	Protected	General	Loop	Caught, restarted
19	2	Protected	NTPQ	Crash	Caught, restarted
20	2	Protected	NTPQ	Message	Caught, restarted
21	2	Protected	NTPQ	Loop	Caught, restarted
22	1	Protected	General	Crash	Caught, restarted
23	1	Protected	General	Message	Caught, restarted
24	1	Protected	General	Loop	Caught, restarted
25	2	Protected	NTPQ	Crash	Caught, restarted
26	1	None	-	Loop	Infinite loop - unresponsive
27	1	Patched	-	Loop	Invalid arg, no crash
28	2	Protected	NTPQ	Message	Caught, restarted
29	2	Protected	NTPQ	Loop	Caught, restarted
30	1	Protected	General	Message	Caught, restarted
31	1	Protected	General	Loop	Caught, restarted
32	2	None	-	Crash	Seg fault
33	2	Patched	-	Crash	Invalid arg, no crash
34	2	Protected	NTPQ	Crash	Caught, restarted
35	2	Protected	NTPQ	Message	Caught, restarted
36	1	Protected	General	Crash	Caught, restarted
37	1	Protected	General	Message	Caught, restarted

pipeline implementation is 0.1 ± 2.6 ms. The difference between the implementation with no pipeline and the General pipeline implementation is 0.5 ± 2.7 ms. These results indicate no difference in performance within statistical deviation between an implementation with no security capabilities and implementations with security capabilities.

Table 5 shows the physical impact of the HARD pipelines in terms of an increase of logic used in implementation. The baseline Arm Cortex-A55 had the equivalent of 615035 LUT4 primitives. The NTPQ pipeline implementation increased 4.4% from baseline to 642257 LUT4 primitives. The General pipeline implementation increased 6.5% from baseline to 654747 LUT4 primitives.

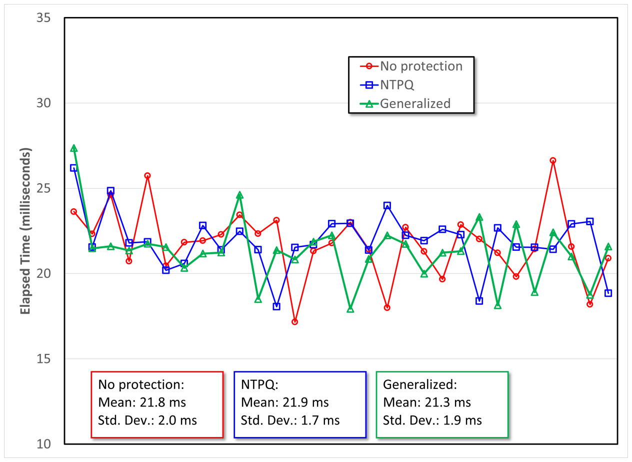
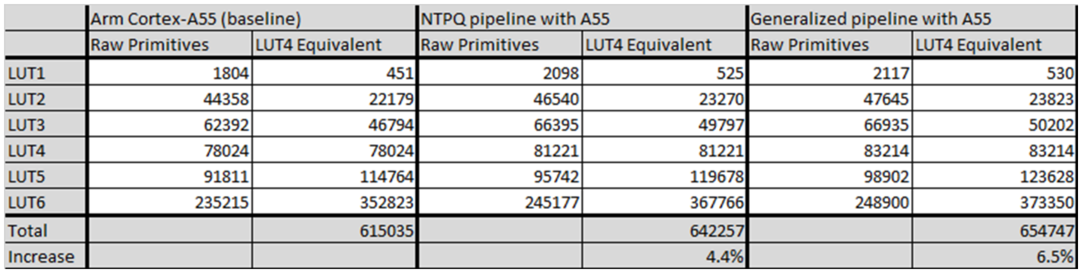


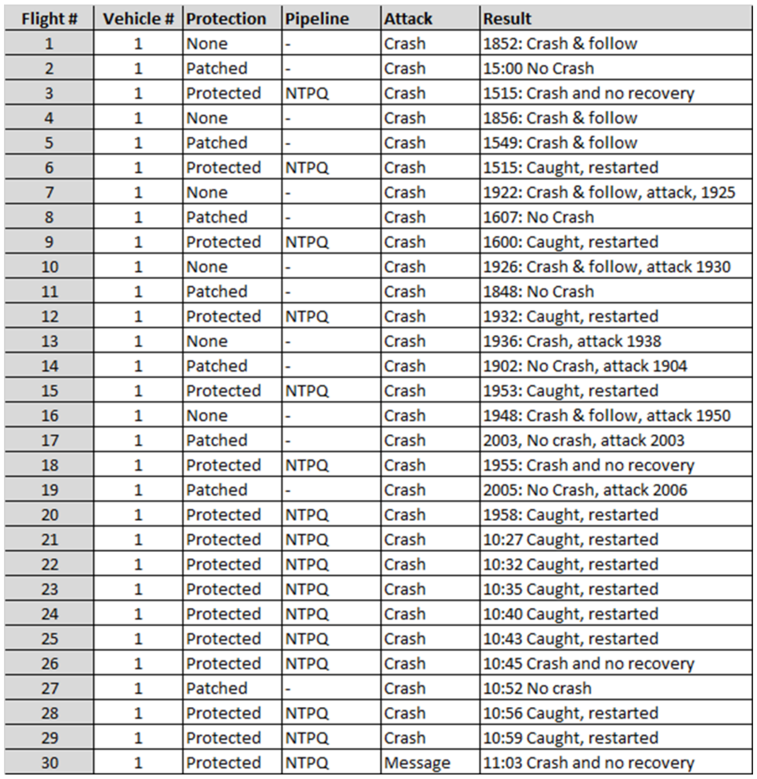
Fig. 13 Timing values for the `ntpq` function call and statistics across protection cases for the Stalker platform.

Table 5 Logic consumed by the implementations across protections cases for the Stalker platform.



	Arm Cortex-A55 (baseline)		NTPQ pipeline with A55		Generalized pipeline with A55	
	Raw Primitives	LUT4 Equivalent	Raw Primitives	LUT4 Equivalent	Raw Primitives	LUT4 Equivalent
LUT1	1804	451	2098	525	2117	530
LUT2	44358	22179	46540	23270	47645	23823
LUT3	62392	46794	66395	49797	66935	50202
LUT4	78024	78024	81221	81221	83214	83214
LUT5	91811	114764	95742	119678	98902	123628
LUT6	235215	352823	245177	367766	248900	373350
Total		615035		642257		654747
Increase				4.4%		6.5%

Table 6 FreeFly Alta-X Demonstration - Record of Cyber-attacks and Results.



Flight #	Vehicle #	Protection	Pipeline	Attack	Result
1	1	None	-	Crash	1852: Crash & follow
2	1	Patched	-	Crash	15:00 No Crash
3	1	Protected	NTPQ	Crash	1515: Crash and no recovery
4	1	None	-	Crash	1856: Crash & follow
5	1	Patched	-	Crash	1549: Crash & follow
6	1	Protected	NTPQ	Crash	1515: Caught, restarted
7	1	None	-	Crash	1922: Crash & follow, attack, 1925
8	1	Patched	-	Crash	1607: No Crash
9	1	Protected	NTPQ	Crash	1600: Caught, restarted
10	1	None	-	Crash	1926: Crash & follow, attack 1930
11	1	Patched	-	Crash	1848: No Crash
12	1	Protected	NTPQ	Crash	1932: Caught, restarted
13	1	None	-	Crash	1936: Crash, attack 1938
14	1	Patched	-	Crash	1902: No Crash, attack 1904
15	1	Protected	NTPQ	Crash	1953: Caught, restarted
16	1	None	-	Crash	1948: Crash & follow, attack 1950
17	1	Patched	-	Crash	2003, No crash, attack 2003
18	1	Protected	NTPQ	Crash	1955: Crash and no recovery
19	1	Patched	-	Crash	2005: No Crash, attack 2006
20	1	Protected	NTPQ	Crash	1958: Caught, restarted
21	1	Protected	NTPQ	Crash	10:27 Caught, restarted
22	1	Protected	NTPQ	Crash	10:32 Caught, restarted
23	1	Protected	NTPQ	Crash	10:35 Caught, restarted
24	1	Protected	NTPQ	Crash	10:40 Caught, restarted
25	1	Protected	NTPQ	Crash	10:43 Caught, restarted
26	1	Protected	NTPQ	Crash	10:45 Crash and no recovery
27	1	Patched	-	Crash	10:52 No crash
28	1	Protected	NTPQ	Crash	10:56 Caught, restarted
29	1	Protected	NTPQ	Crash	10:59 Caught, restarted
30	1	Protected	NTPQ	Message	11:03 Crash and no recovery

2. FreeFly Alta-X

The record of cyber-attacks on the FreeFly Alta-X, the various configurations, and the results are shown in Table 6. The FreeFly Alta-X used only one HARD pipeline for protection in the various flights. The first pipeline, labeled NTPQ(PYNQ), is a technique customized to detect the `ntpq` bug from the level 1 information available on the implemented ports in the SoC of the PYNQ Z2. The FreeFly Alta-X demonstration scenario used only one type of attacks, a crash. The crash attack simply overflowed the buffer using the `ntpq` bug and overwrote the return address with an invalid program counter, causing the vehicle wrapper software module to crash. The NTPQ(PYNQ) pipeline was able to detect 13 of the 16 events corresponding to a 0.81 event detection ratio.

Figure 14 shows the timing data collected across multiple trials for configurations with no pipeline and the NTPQ(PYNQ) pipeline. The difference between the implementation with no pipeline and the NTPQ(PYNQ) pipeline implementation is 0.02 ± 0.11 ms. These results indicate no difference in performance within statistical deviation between an implementation with no security capabilities and implementation with security capabilities.

Table 7 shows the physical impact of the HARD pipelines in terms of an increase of logic used in implementation. The baseline Arm Cortex-A55 had the equivalent of 615035 LUT4 primitives. The NTPQ(PYNQ) pipeline implementation increased 0.7% from baseline to 642257 LUT4 primitives.

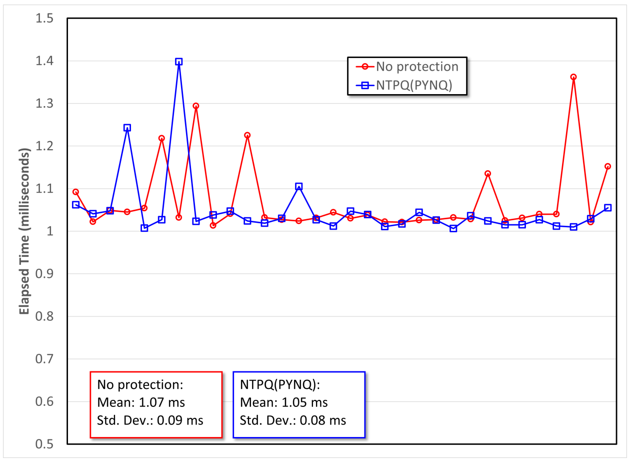


Fig. 14 Timing values for the `ntpq` function call and statistics across the no protection case and the NTPQ(PYNQ) protection case for the FreeFly Alta-X platform.

Table 7 Logic consumed by the implementations across protections cases for the FreeFly Alta-X platform.

	Arm Cortex-A55 (baseline)		NTPQ(PYNQ) pipeline with A55	
	Raw Primitives	LUT4 Equivalent	Raw Primitives	LUT4 Equivalent
LUT1	1804	451	1864	466
LUT2	44358	22179	44955	22478
LUT3	62392	46794	63155	47367
LUT4	78024	78024	78661	78661
LUT5	91811	114764	92470	115588
LUT6	235215	352823	236559	354839
Total		615035		619399
Increase				0.7%

C. Summary

Table 8 summarizes the analysis of the data collected during the Stalker and FreeFly Alta-X demonstrations. The NTPQ and Generalized security pipelines effectively detected every event that occurred with no impact to the performance of the monitored processor. These pipelines had a minimal impact to the physical area when compared to the Arm Cortex-A55. The NTPQ(PYNQ) security pipeline was able to be implemented on an off-the-shelf SoC, aligning more closely with the commercial market. The physical area implementation of the NTPQ(PYNQ) security pipeline was miniscule compared to the baseline Arm Cortex-A55. However, the light weight implementation did not have as high detection ratio as a trade off, but there was no impact to the performance of the monitored processor.

Table 8 The capabilities of the various security pipelines was independent of the impact to the performance and scaled with physical area of the implementation.

Platform	Pipeline	Event Detection Ratio	Performance	Physical Impact (Area)
Stalker	NTPQ	1.00	No Statistical Difference	4.4%
Stalker	Generalized	1.00	No Statistical Difference	6.5%
FreeFly Alta-X	NTPQ(PYNQ)	0.81	No Statistical Difference	0.7%

IV. Conclusion & Future Work

The PUP Team demonstrated HARD cyber protection capabilities, first utilizing the Lockheed Martin Stalker UAS, and then a commercially available FreeFly Alta-X quadcopter. The team illustrated a real-world cyber threat to networked, edge-compute systems, and displayed the ability to contain the exploit in an independent system while still showing applicability to UAS.

A multi-layered security approach typically relies on physical security, secure networks, and software patches. This leaves UASs susceptible to cyber-attacks through unpatched vulnerabilities, new attack capabilities, and malicious users. Lockheed Martin's HARD capability acts as a foundational security layer to this multi-layered security approach, protecting against exploits that bypass traditional security layers. Specifically, HARD protects systems at the hardware level, turning a traditional software discovery and patch cycle into a one-time pattern mitigation implementation. This capability will protect against a broader range of exploits, including those that would have been missed in patches.

Lockheed Martin's HARD capability also plays a role in a more proactive cybersecurity system. This cybersecurity system is a modular and open system architecture designed to monitor, detect, report, and optionally respond to a wide variety of cyber events occurring on a platform that can be tailored to address the specific needs of platform, including sUAS. An example of such as system is shown in Figure 15. In the awareness component of this architecture, various levels of the platform are monitored, generating cyber data. The levels may include behavior of the networks, operating systems, applications, and processors in the platform. For example, a module may be plugged into a spare port of an Ethernet switch and monitor traffic between devices. The module may digest and format the monitored information and pass on this generated cyber data to a detection function. This detection function filters through the cyber data and extracts any cyber anomalies. Depending on the number of monitors, the cyber anomalies may be aggregated together into a stream of noteworthy cyber events. These cyber events may be passed to a logging system where the reports of the cyber anomalies may be analyzed offline at a later time which would support the more passive approach of developing software patches. A response system may consume the cyber events and analyze the holistic view of cyber events producing more informed decisions. For example, an increase in network traffic by itself does not warrant a response by itself, but if an application in a connected subsystem is showing anomalous behavior as well, then the analysis may show another subsystem is attacking the struggling subsystem. This cybersecurity system then may mitigate the cause (disabling the subsystem originating the attack) rather than attempting to remedy the symptom (rebooting the struggling subsystem) which may then immediately come under attack again.

The SSITH HARD techniques uniquely fit into the hardware/processor level of the example proactive cybersecurity system shown in Figure 15. The PUP demonstration is limited to a single hardware/processor level monitor and detector with a simplified response where the vehicle wrapper module of the Mosaic Warfare system is restarted when a cyber anomaly is discovered.

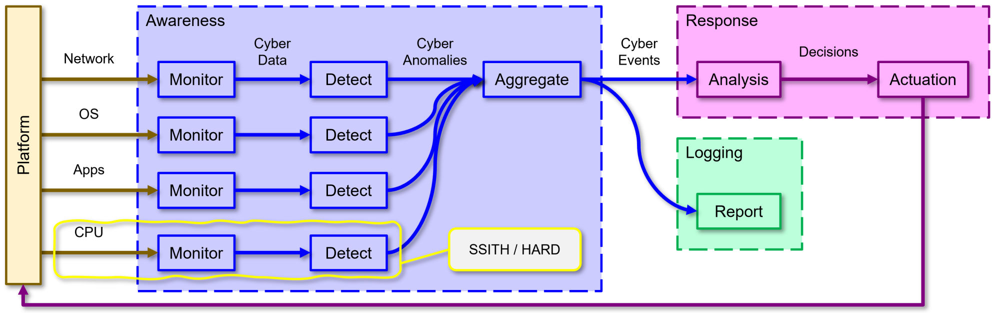


Fig. 15 A cybersecurity system of systems takes a holistic view of the platform and crafts its responses appropriately.