Drone Calibration System

1st José Costa
Department of Electronics, Telecommunications and Informatics
University of Aveiro)
Aveiro, Portugal
joselcosta@ua.pt

***Abstract*—These last few years drones have become very useful, not only in everyday tasks, but also in the method of transportation, but also in multiple industries such as agriculture, military, package delivery etc.**

Drone reliability is a big concern for these different multiple applications, these have to be stable and consistent, no only to successfully complete the missions, but to guarantee the safety of all workers that collaborate directly with the drones.

This is the case with the package delivery transportation system responsible for the automatic delivery of packages to a destination.

This report elaborates on a solution regarding the drone calibration system commonly used in today's industry, as most drones fleets periodically have to be manually calibrated by a highly skilled individual. Our solution makes this task easy, with a highly simple and intuitive software that can be widely used with most industry used drones.

***Index Terms*—drone, delivery, calibration**

I. Introduction

At this moment in time, industry used drones are becoming the main choice in multiple commercial enterprise use cases. Multiple industries such as the military and construction, use drones to reliably deliver packages to certain parts of the globe with constrained or very difficult access, with the added advantage of a very significantly low budget compared to other current solutions and the safety of workers that previously conducted these types of deliveries. The problem discussed in this report is directly related to the development of a new autonomous drone package delivery system that is currently being tested by multiple online shopping vendors.

This software is used to control and coordinate drone missions with the objective of safely and efficiently delivering a package. For this to take place, the drone must be always in a calibrated state, with multiple internal parameters such as the P gain, I gain, D gain and many others. An improper configuration of the parameters can result in unreliable drone behavior, and in the worst case, the full collapse of the drone.

In the real world, this would have a major impact not only in the security of all individuals in a public space where drones could cross, but also the damage of the to be delivered goods, that depending on the importance of them, can cause major problems to companies and users alike.

Our solution provides an easy and intuitive solution that involves an innovative software responsible for the configuration and maintenance of all drone internal parameters, guaranteeing reliable and stable flights, while the packages are being delivered. This software will use multiple calibration drills that can be performed in open spaces with no obstacles, in conjunction with a communication system responsible for immediate intervention and autonomous self calibration while missions are being conducted.

All of these calibration drills have to be performed in a controlled secure environment, with no people present in the same space as the drone while it is still uncalibrated and the use of soft materials so as to not damage the drone in case of failure.

As this software will use a communication system that is critical to the drone behavior and movement, security concerns should be evaluated, tested and successfully applied, such as the confidentiality and encryption of the messages, their integrity and the authenticity of the machines involved in the communication. The robustness of immediate intervention in case of calibration failure is also taken into account. Being capable of safely landing or correcting wrong drone trajectories are essential to a safe delivery. It will also highly compatible with most industry used drone solutions, providing a compatibility layer that integrates seamlessly with the drones proprietary internal software.

II. Secure Development Lifecycle

A. Education and awareness

In all cybersecurity, the most common weakest link lies with the users. Human error is inevitable and can greatly be reduced by training and practice on these topics. As this piece of software is going to directly impact the drones stability and safety, the education and awareness of all individuals is crucial, not only to the software engineers, but also to all that physically test the drones, so in case of emergencies everyone knows how to react and approach the situation.

B. Project Inception

During this phase, the definition of cryptographic standards is important, as this application heavily relies on a client server communication system that exchanges critical information, tampering or faking transmissions could be catastrophic.

The discussion and definition of minimum acceptable level of security, this will help not only in the problem identification, but also in their fix.

C. Analysis and requirements

This part of the secure development lifecycle focuses on finding and documenting all requirements. These should be gathered during the initial design and planning phases. Security requirements are especially important, techniques such as thread modeling can also be applied in this phase, identifying, communicating, and understanding threats and mitigations within the context of the developed software.

D. Architectural and detailed design

During this phase of the SDL, the requirements from the previous requirements analysis phase, and design an architecture that complies with them, and that are well designed with respect to security. The architecture risk analysis also allows the identification of risks to the business, directly related to the architecture. These are given a rate based on the potential impact they could have.

E. Implementation and testing

In this phase, the solution implementation will start simultaneously with the testing, this will ensure the continuous search and fix of vulnerabilities while the application is being developed. Applying static analysis security testing and dynamic analysis security testing are a few ways to analyse the system source code and find problems. In this stage, all third party components, including the drones multiple native software systems, should be considered and the risk of interacting with them should be taken into account.

F. Release, deployment and support

In this last phase of the SDL, the system should be stable, pass all the tests from the previous state and be ready to go live. The drones will be launched and the first deliveries will start to happen.

During this phase, an incident and response plan is created to address new threats, issues and physical hardware failure that can happen while the system is used.

This plan should contain information about who to contact in case of an emergency and multiple protocols for the multiple security servicing, be it hardware or software.

III. Security Requirements

Security requirements are a specific type of non-functional requirements that focus on security functionalities in the context of secure software with the objective of increasing the users trust in the system, by solving security problems and eliminating vulnerabilities.

They specify objectives and expectations to protect the services and data that are used by the system or specific software, this includes all application development, with particular consideration for outsourced development integration.

Security requirements, as most software requirements, is a goal set out for an application at its inception. As this phase is one of the first phases of development, poorly defined requirements can cause major problems to the project that can potentially lead to security breaches or compromised systems which are always very critical and can be very costly to the responsible organization.

To ensure that a software solution correctly solves a particular problem, we must initially fully understand the problem that needs to be solved, discover why the problem needs to be solved and determine who should be involved in its resolution.

To obtain the correct security requirements its essential to understand and apply the relevant phases of the Software Security Life-cycle, starting with identifying potential vulnerabilities and weaknesses, assessing the risk by assigning a severity degree for example, fixing or mitigating the vulnerability if it can't be fixed and monitoring the, now mitigated, vulnerability.

This report will focus on three main security goals:

- Confidentiality.
- Integrity.
- Availability.

Confidentiality is essential to secure software, it allows the application data to be stored or communicated between different sources, with the guarantee that only trusted entities can understand it. This is essential to protect against Man In the Middle attacks for example. Integrity is also important to secure software, it allows for the application data to not forged by an attacker, be it in communications or storage. Lastly, availability is the need for the protection against denial of service attacks for example, by keeping the system with the most up time possible.

Identifier	Description	type
SYS-R-010	The calibration communication system should have a 99.9% up time availability.	Availability
SYS-R-020	The software must be replicated in two offsite locations to provide execution redundancy, with proper server synchronization.	Availability
SYS-R-030	All data used in the communication system must be encrypted. Only the application should be able to use the communication channels successfully. No outside information should be leaked or understood.	Confidentiality
SYS-R-40	The system should log all communications and calibrations drills details, including timestamps and location information.	Non Repudiation

SYS-R-050	All stored or communicated data should only be able to be generated by the software. The data used by software should only be generated by the software, no other party is able to generate valid data.	Integrity
SYS-R-060	When performing calibration drills, a user that supervises the drill must be authenticated in the software.	Authentication
SYS-R-070	All information transmitted and received from the software must be digitally signed using a unique serial number that identifies the software installed on a specific machine.	Authentication
SYS-R-080	Only system administrators with specific elevated permissions should be able to manually configure critical internal drone calibration parameters.	Authentication
SYS-R-090	User input used in the application should be validated, verified and examined with particular attention to special characters that can escalate to command injections.	Integrity
SYS-R-100	All communications have to use a secure communication protocol with at least a 256 bit key.	Confidentiality

IV. CWE Analysis

The Common Weakness Enumeration (CWE) is a classification system that groups software weaknesses and vulnerabilities in categories. It serves as a common language, a way of measuring security tools and as a baseline for weakness identification, mitigation, and prevention.

CWE addresses both hardware and software vulnerabilities / weaknesses. By educating engineers in both of these areas, the main goal is to stop all vulnerabilities in all phases of the software development life cycle, greatly reducing the most common mistakes before products are delivered.

Ultimately,use of CWE helps prevent the kinds of security vulnerabilities that have plagued the software and hardware industries and put enterprises at risk.

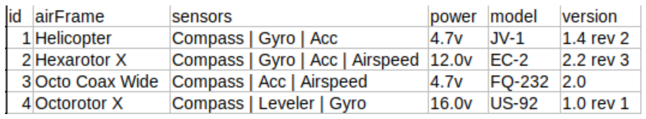
On the following subsections, a software vulnerability will be discussed, detailing the all of the following steps:

- Problem Description.
- Problem Detection.
- Problem Solution.

A. *Problem Description*

The drone calibration application lets the user import and export information that is used to identify and characterize drones. Information such as the drone id, model, airFrame etc. can be uploaded to the application front-end as a simple csv file.

The following image illustrates the default format for a valid csv file:



id	airFrame	sensors	power	model	version
1	Helicopter	Compass | Gyro | Acc	4.7v	JV-1	1.4 rev 2
2	Hexarotor X	Compass | Gyro | Acc | Airspeed	12.0v	EC-2	2.2 rev 3
3	Octo Coax Wide	Compass | Acc | Airspeed	4.7v	FQ-232	2.0
4	Octorotor X	Compass | Leveler | Gyro	16.0v	US-92	1.0 rev 1

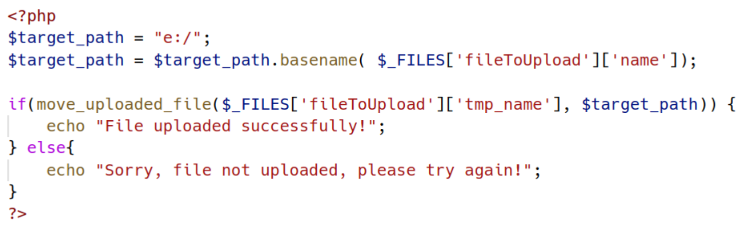
As the application does not validate the file information or extension, any file can be accepted as input and saved on the server where the application is executing.

This vulnerability generally fits the following CWE:

- CWE-434: Unrestricted Upload of File with Dangerous Type

The attacker can use this file upload to execute malicious code on the server machine, leading to RCE or malware injection. This is a common attack that is used mainly with web servers that run PHP, which is the case of the application in question.

A piece of the vulnerable file upload in the php programming language can be seen in the following picture:



```
<?php
$target_path = "e:/";
$target_path = $target_path.basename( $_FILES['fileToUpload']['name']);

if(move_uploaded_file($_FILES['fileToUpload']['tmp_name'], $target_path)) {
    echo "File uploaded successfully!";
} else{
    echo "Sorry, file not uploaded, please try again!";
}
?>
```

B. Problem Detection

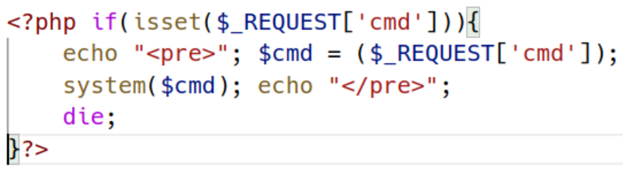
To detect problems in the system, two methods of verification and testing were applied.

The first verification method applied was static code analysis. This type of analysis addresses weaknesses in source code that might lead to vulnerabilities.This may also be achieved through manual code reviews, but using automated tools is much more effective. To perform this first verification, the SonarQube tool was used to scan the website yielding multiple results of code optimization and the file upload vulnerability described previously.

Another method of verification was also used, the penetration test method consists in an authorized simulated cyber attack on a computer system, conducted by an individual or a team with knowledge on this field. A penetration test target may be a white box (information to the system and full documentation is provided) or a black box (no information, or very limited public information). A gray box penetration test is a combination of the two.

The code snippet from the previous subsection clearly does not take into account any information of the file that is being uploaded to the server, there is no file mime verification and no extension verification to check if it is a csv file.

To exploit example of this vulnerability can be the following php script:



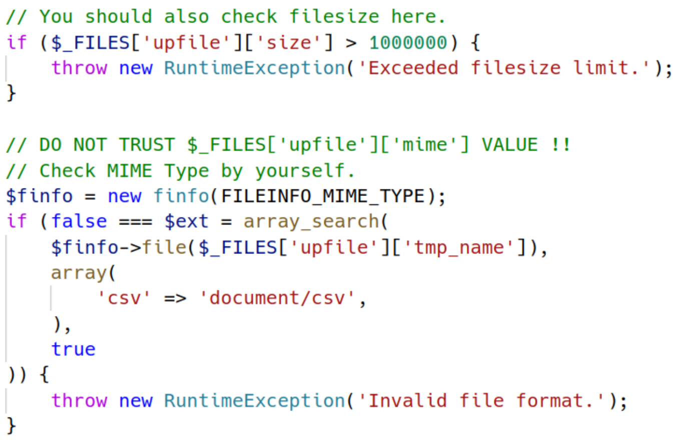
```php
<?php if(isset($_REQUEST['cmd'])){
    echo "<pre>"; $cmd = ($_REQUEST['cmd']);
    system($cmd); echo "</pre>";
    die;
}?>
```

This code snippet allows the user to load the uploaded php file on the website, and by sending an get request with the cmd parameter equal to some os command, this will be executed in the host machine, and show the results in the web page.

C. Problem Solution

The solution to the unrestricted file upload presented in this section, requires the validation of all the files that can be uploaded to the server. An example solution that solves this vulnerability is presented in the following picture:

[h]



```php
// You should also check filesize here.
if ($_FILES['upfile']['size'] > 1000000) {
    throw new RuntimeException('Exceeded filesize limit.');
}

// DO NOT TRUST $_FILES['upfile']['mime'] VALUE !!
// Check MIME Type by yourself.
$finfo = new finfo(FILEINFO_MIME_TYPE);
if (false === $ext = array_search(
    $finfo->file($_FILES['upfile']['tmp_name']),
    array(
        'csv' => 'document/csv',
    ),
    true
)) {
    throw new RuntimeException('Invalid file format.');
}
```

This solution differs from the original vulnerable code snippet by not only checking the file size before upload, but also manually checking the mime-type of the file, and associating it with the corresponding extension. In this case the csv type is compared to the document/csv mime-type to verify the uploaded file.

V. Testing

This section has the objective of specifying test cases for each one of the previously defined security requirements, planning a penetration test to the performed on the system that provides the service and lastly elaborating a plan to perform fuzzy testing.

A. Testing Setup

Requirement ID	Test ID	Description
SYS-R-010	SYS-TR-010	By checking at a constant interval if the communication system can successfully send and receive messages correctly, the up time can be calculated.
SYS-R-020	SYS-TR-020	By checking all drone information on differently located redundant servers in an instant, the data should be the same.

SYS-R-030	SYS-TR-030	By capturing traffic from the application (MITM), the data must encrypted and not easily crackable. If any information is inserted by a outside source, the information must be discarded by the software.
SYS-R-040	SYS-TR-040	After performing a calibration drill or a communication, verify the log to check if all the information was stored properly.
SYS-R-050	SYS-TR-050	Check if a file with information generated by the application has the digital signature of that specific software.
SYS-R-060	SYS-TR-060	Check if during the calibration drill, the user that supervises the drill is authenticated in the software. An alternative is to check the logs for this information.
SYS-R-070	SYS-TR-070	Try to send information to an application communication channel with a random encryption using a random serial number and check if the software blocks this fake information.
SYS-R-080	SYS-TR-080	Login in the application with a normal user, and try to manually change the critical drone parameters and verify that the application blocks this behaviour.
SYS-R-090	SYS-TR-090	Apply fuzzing techniques to all user input fields in the application, to check for potential command execution or information disclosure.
SYS-R-100	SYS-TR-100	Verify if the encryption algorithm used in the communication system has a 256 bit key.

B. Penetration Testing Plan

A penetration test, also known as a pen test, is a simulated cyber attack against a system to check for exploitable vulnerabilities.

Pen tests serve as a way to examine whether an organization's security policies on systems and software are genuinely effective.

To perform a penetration test on the application, information and control may be achieved with exploitation of the frontend of the software, that can lead to a compromise of the system, with a reverse shell for example, or just leak information and change the software functionalities with no system access (only on the frontend).

Using the NMAP tool enumeration can be performed in the ip where the web server (frontend) is hosted. After checking for running services, check if any of them have open access with no authentication.

Using the DirBuster tool, the directories and web pages of the application can be enumerated, and hidden pages can be detected, that may be exploited, or contain relevant information.

Check the website for file uploads and user inputs, use fuzzing techniques to check for information leakage with SQL injections or successful malicious file uploads.

If any information is leaked and RCE is achieved in the system hosting the application, try the check for the files permissions to check for any strategic files that can be altered and are used by the software. Also checking for admin authentication written in files in the system is advised.

Another attack vector can be to sniff the application information, to check if any domain name resolution is used by the application, that can be fakes with a fake DNS server, which will redirect the communications.

After performing the penetration testing, it is important to perform a cleanup on the system, for all files and information used in the pen test.

It's also important to summarize all discovered vulnerabilities during the penetration test in a structured report containing:

- Vulnerability brief description
- Severity
- How to exploit
- Mitigation recommendations

C. *Fuzzy Testing Plan*

In this subsection a plan for the execution of fuzzy testing on the application is presented.

Fuzz testing or Fuzzing is a software testing technique of putting invalid or random data called FUZZ into software system to discover coding errors and security loopholes.

The purpose of fuzz testing is inserting data using automated or semi-automated techniques and testing the system for various exceptions like system crashing or failure of built-in code, exposing data etc.

This fuzz testing process is different for every software or system that is being assessed, but it generally complies with the following basic steps:

- Identify the target system
  (according to the scope of the test)
- Identify inputs
- Generate or download Fuzzed data
- Execute the test using fuzzy data
- Monitor system behavior
- Log results

With these process in mind, this test is going to be applied to the software to validate the compliance to the previously established security requirements defined for this project.

The SYS-R-090 requirement states that the user input used in the application should be validated,verified and examined. This is an example of a requirement in which fuzz testing can be applied to verify if no user input can have a strange response from the software.

In this application, as a web frontend is used to interact with the software, most likely, HTTP requests are used to perform the communication between the frontend and the software service API. A simple proxy can be set up on the machine to redirect all traffic to a packet sniffer, that can modify the contents of the payload that is sent to the services API. By identifying the field in which the user input is present, this input is replaced by attack vectors (previously generated normally) and this tampered request is forwarded to the services API. After the request has been sent, the response is also going to be captured by the proxy, in which error messages or response sizes can be used to check if normal behaviour occurred or not.

One of the most common tool to perform this type of testing is Burp suite, which has the capability of acting as a proxy, intercept traffic, alter that traffic and forward the altered traffic, as required for the fuzz test.

The attack vector in this case, should be a list of command injections or SQL injections for example, depending on the input that is being tested.

In the case that the input field is the name of a backup file, command injections should be the attack vector.

If a search bar, or a login for are the inputs that are being tested, SQL injections should be selected as the attack vector.

To end this section some attack vectors that can used in the fuzzy testing are presented as well as the expected system behavior and the possible suspicious behaviour on a search bar used to search available drones, where special characters are now allowed.

Vector	Expected behaviour	Possible Suspicious Behaviour
”OR x=y #”	The input should not show any drone available.	An error is displayed.
”drone-01”	The input should show an drone, if it exists with that name.	——
”delivery_drone”	The input should show an drone, if it exists with that name.	——
”drone-01 /*comment*/ ”	The input should not show any drone available.	The response shows drone-01 as available, if it exists
”ORDER BY 1–”	The input should show an drone, if it exists with that name.	The drones response list is ordered by a parameter
”drone1 s2”	The input should not show any drone available as no white spaces can be used.	——
””	The input should not show any drone available.	The response shows all drones available
temporary_delivery-drone1	The input should show an drone, if it exists with that name.	——-
drone-01 – comment	The input should show an drone, if it exists with that name.	The response should not show drone-01.
1’ || pg_sleep(10)	The input should not show any drone available.	The response takes roughly 10 seconds.

VI. Hazards

In this section 5 security hazards were analysed with the objective of determining the hazard consequences, origin, state etc:

- Denial of service

- Malware
- Data corruption
- Cryptojacking
- Password theft

An analysis of the following 2 functionalities was also conducted in the form of a FMEA:

- Drone communication
- Application login

Both of these analysis are provided in the external documents system_fmea.xlsx and hazard_XX.xlsx.