Security Assessment of an Open Source Software

Shankar Saikia[#1], Nife Teye[#2], Adesina Tijani[#3], Sowmya Reddy Tukakula[#4], Chas-Nwam Nomnso[#5]

University of North Texas
CSCE 4565 - Secure Software Development

***Abstract*— Understanding the vulnerabilities within an open source tool and analyzing the impacts of the vulnerability**
Open Source Software has become increasingly popular. It is free and accessible to the public. The free access to the software source code allows for innovation, but also raises potential security concerns. Without verification processes in place, vulnerabilities could be introduced that attackers might exploit to compromise systems. The goal of this paper is to perform a security assessment on open- source tools. This paper gives insight on the tool iSpy and analyzes the potential vulnerabilities present in the software using its architecture and design.

***Keywords*— Open Source software, iSpy, Security assessment, webcam, Camera, Vulnerability assessment.**

SECTION I

I. Introduction

During the mid 1900s, video surveillance systems became extremely popularized. They were used to protect homes, businesses, even streets (public spaces) by monitoring for suspicious activity and helping to deter criminals. However, since video surveillance systems are expensive, iSpy was created. It has become one of the world's most popular open source video surveillance application. It's compatible with the vast majority of consumer webcams and IP cameras. With more than 2 million users worldwide, iSpy works with a lot of cameras and devices. Started back in 2007 the software has continually evolved and improved.

The number one use of iSpy is small business security, but home monitoring, neighborhood watch, nanny-watch and mobile access are valued features. Facial recognition and detection of changes in lighting and audio offer the subtleties that set the software apart from competitors.

All that is needed for the software to work is a webcam or IP camera that is or can be connected to a computer or network. iSpy connects to the camera and shows the live view. Then the specific areas of the video that iSpy should watch for movement can be defined, and a threshold value for the amount of motion that would trigger automatic recording should be set. iSpy can also operate in always-recording or manual-recording modes and supports scheduling and remote access.
iSpy was designed to provide a low-cost alternative to expensive surveillance systems. It has become a highly scalable application that can be tailored to record and take actions on specific incidents as defined by the user either locally or remotely.

SECTION II

II. Description

iSpy has a cloud component where the footage from the cameras gets uploaded for storage. There were multiple updates made to this component (as can be seen in the Github repository) that supposedly fixed configurations and upload issues with the cloud environment and cameras, respectively.

There is also a server component which controls authentication and authorization of the subject (user) and helps ensure the CIA of the surveillance camera system.

Finally, there are the cameras themselves, which are the devices responsible for capturing footage and uploading it to the cloud environment.

Because iSpy is an open source tool, meaning the complete source code for the cloud component and the server component and any other supporting libraries is readily available online, to the public. This provides full transparency into the implementation details, which can promote community testing and review. On the other hand, it also potentially exposes internal vulnerabilities or insecure codes that skilled attackers could look to exploit.

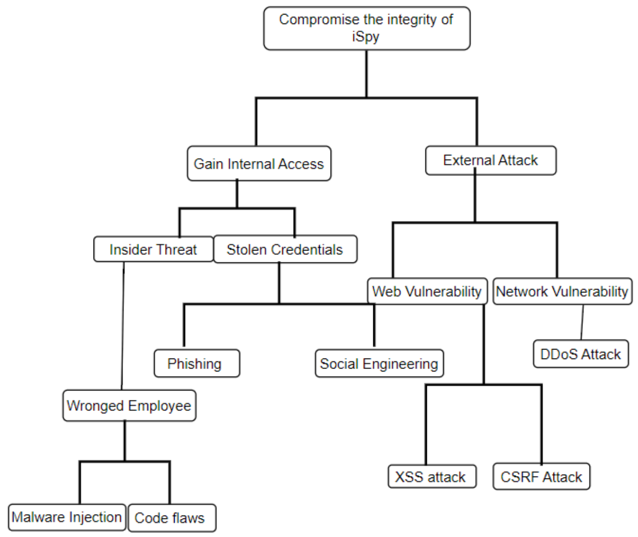
III. Threat Model

The assets within the system are as follows:

1. The cameras themselves (they can be expensive devices)
2. The footage that was captured by the cameras and stored in the cloud
3. The subject's (user's) credentials and other personal data
4. The architecture of the system (and other company trade secrets)

Attack trees

In this attack tree, the goal of the attacker is to compromise the integrity of the system.



STRIDE Threat Modeling Descriptions

Spoofing: the identity of a subject (user) can be spoofed to gain access to the system

Tampering: the data of a user in the system (e.g.: credentials and personal data) can be tampered with which breaches integrity; also, the stored footage in the cloud environment can be tampered with

Repudiation: a malicious actor can claim that they did not compromise a system if they covered their trail well

Information Disclosure: the personal data and/or credentials of a user in the system can be disclosed from the system without proper protections and preventative measures

Denial of Service: the server and/or cloud environment can be denied from providing their services to the users by a malicious actor if they are able to prevent it from functioning normally (eg.: by bombarding it with many requests)

Elevation of Privilege: a malicious actor who gains access to the server could elevate their privilege to be able to perform more powerful activities (such as deleting certain footage or users, or even adding new footage or users)

Misuse cases

1. Sending many requests to authenticate/authorize to the server at once, preventing it from functioning nominally
2. Sending too many requests to upload or download footage to/from the cloud at once, which also prevents it from functioning normally
3. Removing the authentication and authorization protocol functions from the server code (since the software is open-source): without proper system logs and audits, such a drastic change could go undetected
4. Not requiring a two-factor authentication protocol for user authentication into the system

Abuse cases

1. Uploading a large amount of footage to the cloud environment so that it disrupts the service for other users
2. Elevating your privilege in the server to create new free accounts for friends to use (bypassing the official signup)

3. Gaining access to others' footage to view what it has captured (breach of confidentiality/privacy)

IV. Security Architecture and Design

Security Requirements

- Only let people who absolutely need to use the camera be able to access it. This means having a password or some kind of authorization measure to make sure the person using it is allowed to do so.
- Prevent anyone outside of administrators and the footage owners (users whose cameras filmed the footage) from accessing it.
- The camera itself should be difficult to tamper with or steal.
- Keep logs/records of when the camera is accessed and by whom.
- Use strong methods to confirm the identity of the person accessing the camera, like two-factor authentication, which is like having a second key.
- The camera itself should be hard to tamper with or steal.
- Prevent unidentified/suspicious devices from entering the network and compromising integrity of stored information(media).

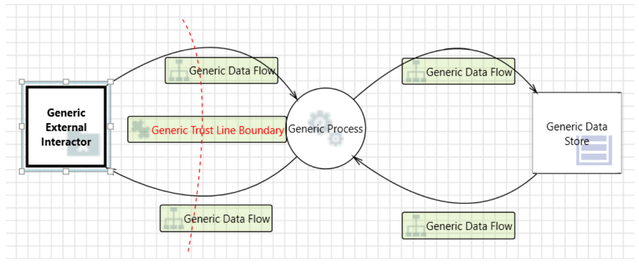
Assumptions (listed with respect to the order of the security requirements above)

- The people who need to use the camera are non-malicious and make use of its services in good faith
- The administrators are non-malicious and will only access the footage if requested in an emergency (e.g.: by law enforcement or other authorities); also, the owners' accounts don't get hacked and their identities don't get spoofed in the system
- The cameras store a small amount of recent footage on the internal memory that should only be accessed for backup emergency purposes.
- The logs/records will have an accurate list of all actions performed by subjects within the system, including if and when the cameras were accessed

Ability to Verify (listed with respect to the order of the security requirements above)

- By reviewing the system logs and using digital forensics techniques to check for intrusion, one can ensure that unauthorized subjects have not accessed the cameras
- Establish and maintain authentication and authorization checks that any subject who wants to access the footage must go through to be able to do so
- Add physical protection to the camera (e.g.: enclose the camera with a protective cover) and virtual protection (e.g.: encrypt the data stored on it)
- Conduct frequent system checks to ensure that actions are detected and successfully recorded in the logs (making sure that logs continue to function correctly)
- Give each device a unique identifier which the server can use when a user is trying to add a device to the network: the server can use it to process and verify that the device is a legitimate device

Data Flow Diagram



The Generic External Interactor represents external entities or actors interacting with the system i.e users, browsers etc. The Generic Process represents the processing components within the system such as the servers. The Generic Data Store represents data storage i.e. the cloud component. Finally, the Generic Data Flow represents the flow of data between components within the system.

Security Architecture

- It is essential to incorporate key principles such as policy, mechanism, and assurance to have a secure and resilient software solution that effectively mitigates risks and protects the confidentiality, integrity, and availability of sensitive data and resources.

- Policy
    - Clear rules and guidelines regarding access control, data protection, and system integrity within the iSpy surveillance system will be established.
    - Policies that dictate who can access the cameras, what actions they can perform, and how sensitive information (such as user credentials and captured footage) should be handled and will be defined.
    - Ensure that the security policies are aligned with industry standards and regulatory requirements to mitigate potential risks effectively.

Mechanisms

- Security mechanisms within the iSpy software to enforce the defined policies and protect against common threats and vulnerabilities will be implemented.
- Authentication mechanisms to verify the identity of users and restrict access to authorized individuals only, preventing spoofing attacks will be developed.
- Encryption techniques to secure data transmission and storage, safeguarding against tampering and information disclosure will be utilized.
- Ensuring there is a secure communication layer between Ispy components. Using HTTPS for web communication and SSL/TLS for Secure Socket Communication between client and server.
- Implementing Access control to restrict access to sensitive information and data within the system. Role-Based access control is used to assign specific permission to different users role-based on their responsibilities.
- Logging and auditing mechanisms to track system activities and ensure accountability, supporting the assurance of policy enforcement will be incorporated.

Assurance

- Thorough testing and validation of the implemented security mechanisms to verify their effectiveness in enforcing the defined policies will be conducted.
- Security assessments, such as penetration testing and code reviews, to identify and address any vulnerabilities or weaknesses in the system will be performed.
- Monitoring and incident response procedures to detect and respond to security incidents promptly, ensuring continuous protection against evolving threats will be established.
- Ispy complies with security standards, regulations and conducts Regular security tests to identify the weakness and vulnerabilities of the system and remediate the security weaknesses in Ispy.
- Evaluation and update of the security architecture and mechanisms to adapt to changes in the threat landscape will be done regularly and maintaining a high level of security assurance will be a priority.

Security Architecture and Design Overview of iSpy:
iSpy utilized BountyCastle for encryption, decryption, digital signatures, and secure communication. It is used for secure data transmission, storage, or authentication.

FFmpeg.AutoGen is a multimedia framework for audio and video processing. It primarily focused on multimedia functionality. Its usage implies iSpy's capabilities to process audio and video streams. It ensures proper validation of multimedia inputs to prevent malicious content injection and secure handling of multimedia files to prevent exploits of vulnerabilities.

iSpy utilizes FlickrNet for accessing the Flickr API, enabling integration with the Flickr platform for media storage or sharing. Integration with Flickr introduces secure authentication, authorization, and data protection.

iSpy utilizes Google.Apis, Google.Apis.Auth, Google.Apis.Core, and Google.Apis.Drive packages. These provide access to Google APIs, specifically Google Drive API, enabling integration with Google services for cloud storage and file management. The integration with Google APIs ensures secure authentication mechanisms, proper authorization controls, and secure communication practices to protect sensitive data stored or accessed via Google Drive.

iSpy utilizes the LibVLC multimedia framework, allowing multimedia playback functionality within iSpy. The multimedia playback features introduce media file parsing vulnerabilities or buffer overflows. iSpy needs to implement secure coding practices and proper input validation to mitigate these risks and ensure the secure handling of multimedia content.

iSpy relies on log4net for recording system events and activities. iSpy leverages log4net to log security-related events, enabling administrators to detect and respond to security incidents effectively.

Newtonsoft.Json is a JSON serialization library utilized by iSpy for serializing /deserializing JSON data. iSpy communicates with external systems or APIs using JSON data format.

RestSharp is a REST and HTTP API client library used by iSpy to facilitate communication with RESTful APIs. However, integration with RESTful APIs requires secure communication practices, including HTTPs usage, proper authentication, and input validation to prevent injection attacks. iSpy must implement secure API communication to protect against data interception or tampering.

iSpy utilized WebSocketSharp to enable real-time communication over WebSocket protocols. This introduces secure WebSocket handshake, message validation, and protection against WebSocket-related vulnerabilities.

iSpy leverages ResX files to manage various resources such as icons, images, and strings used throughout the application. The ResX schema specifies how resources are serialized and converted between their XML representation and native .NET types.

iSpy implements role-based access control mechanisms to manage access to sensitive resources.

post-build event runs a batch script named monitor.bat after a successful build, specifically in Release mode. This script performs signing the executable, a critical step in ensuring the authenticity and integrity of the application.

Regarding code signing and authenticity, the SignAssembly property is set to false, indicating that iSpy does not sign the assembly with a strong named key file. While strong naming can enhance security by preventing tampering with assemblies, code-signing certificates are indispensable for verifying the authenticity and integrity of executable files. iSpy should consider leveraging code-signing certificates to bolster trust.

The inclusion of Monitor.resx and Resources.resx implies iSpy's use of embedded resources for managing localized strings and other resources. iSpy should implement input validation and sanitization to further fortify against injection vulnerabilities.

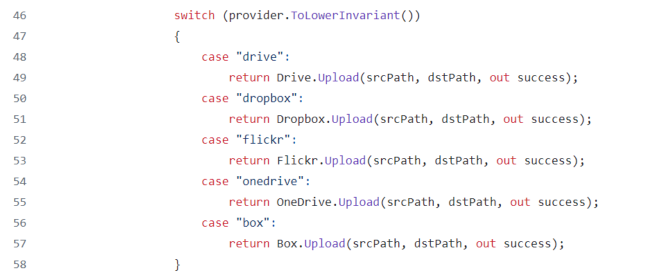
The code snippet named JoystickDevice in the iSpy employs exception handling to capture and log any errors that may occur during joystick initialization, polling, or other operations. Logging exceptions is crucial for security auditing and troubleshooting, as it helps identify and address potential vulnerabilities. It ensures that resources are properly acquired and released, mitigating the risk of resource leaks or denial-of-service attacks caused by resource exhaustion. The class performs input validation when acquiring joystick devices. It checks if the requested joystick device exists before attempting to acquire it, preventing potential errors or security vulnerabilities resulting from invalid input or unauthorized device access. The class adheres to secure coding practices by using exception handling, input validation, and data sanitization techniques.

V. Security Assessment

Code Analysis

In order to perform the code analysis for the iSpy software, we decided to use a combination of peer review of the source code as well as a code analysis tool called SonarQube.

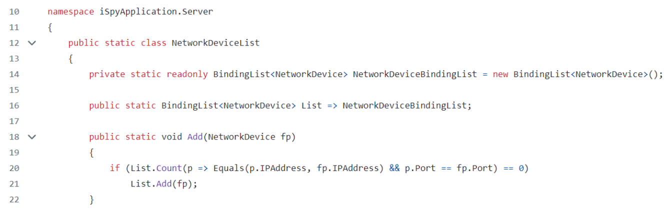
Beginning with the peer review, we manually inspected core components of the source code that make up the foundation of the iSpy camera system. Since the entire code repository is vast and quite extensive, we chose not to go through every single file, instead choosing to focus on the major foundational elements of the system (such as the server, cloud, etc.). Beginning with the cloud component, we visually found a few potential vulnerabilities. The cloud environment is what the iSpy software uses to upload and store captured photos and videos and is essential to ensuring that legitimate users can access their captured footage remotely from anywhere. Therefore, it is equally important to ensure that proper security measures are in place for the cloud environment to make sure that the stored footage stays safe from malicious actors. The first potential vulnerability is shown below:



```
switch (provider.ToLowerInvariant())
{
    case "drive":
        return Drive.Upload(srcPath, dstPath, out success);
    case "dropbox":
        return Dropbox.Upload(srcPath, dstPath, out success);
    case "flickr":
        return Flickr.Upload(srcPath, dstPath, out success);
    case "onedrive":
        return OneDrive.Upload(srcPath, dstPath, out success);
    case "box":
        return Box.Upload(srcPath, dstPath, out success);
}
```

The above code snippet is from the portion of the cloud program responsible for uploading the captured footage to the cloud environment. In this case, it does not appear that proper input sanitization of the captured media is being implemented prior to calling the upload function. An attacker could embed malicious code in certain photos and videos using a hiding technique like steganography - this code could then execute within the environment to perform a multitude of different attacks from deleting all other media to polluting the environment with false media. Therefore, proper sanitization of input is very necessary.

In addition to the cloud upload component, another core component is the server which performs all of the essential operations for the camera system to function. For example, the server provides authentication and authorization services to ensure that only legitimate users and administrators gain access to the system. Based on our observation, this component, for the most part, utilizes proper security measures for the front-end like RSA encryption of sensitive data when a user logs into the system and SSL certificates to establish secure connections with the online user interface. In other words, the server in its current state does a good job with authentication and authorization of a subject. However, it lacks a verification measure to ensure whether a new device that a user is trying to add to the network is legitimate or not (referencing the last security requirement in the "Security Requirements" section above). The code snippet below shows the issue:



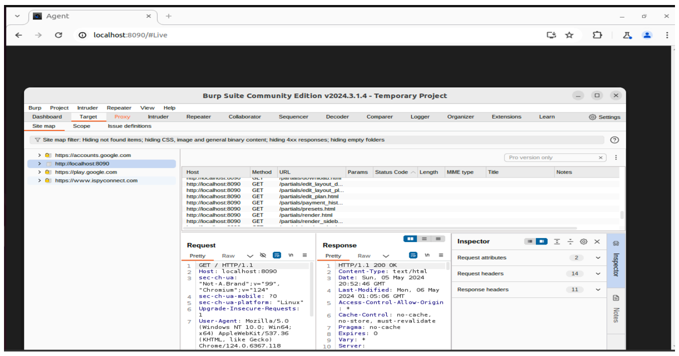
```
namespace iSpyApplication.Server
{
    public static class NetworkDeviceList
    {
        private static readonly BindingList<NetworkDevice> NetworkDeviceBindingList = new BindingList<NetworkDevice>();

        public static BindingList<NetworkDevice> List => NetworkDeviceBindingList;

        public static void Add(NetworkDevice fp)
        {
            if (List.Count(p => Equals(p.IPAddress, fp.IPAddress) && p.Port == fp.Port) == 0)
                List.Add(fp);
        }
```

This code (which is in the "NetworkDevicesList.cs" file in the server component adds a new device to the iSpy network for the user. However, there is no check to see if the device is legitimate or not. For example, each device could have a unique identifier, and this part of the code could contain an if-else condition to see if the identifier is part of a database of legitimate identifiers: if it is, the addition of the device can proceed. If it is not, the addition is rejected. This is important to implement since an attacker could connect a malicious device to the network without such verification and compromise the integrity of the media stored in the cloud environment.

The code used to create the software is written in C#, and SonarQube provides good support for this language. The code analysis was performed in Kali Linux. The GitHub repository where the original code was created was cloned providing a localized environment, which in turn allows us to fully work on the code analysis without disrupting the process of the main GitHub repository.

<u>PenTesting</u>

This section provides an overview of the penetration testing performed on iSpy using Burp Suite community edition which focused on techniques like intercepting traffic and mapping the site. The pentesting was simplified to include 3 major parts. Site mapping, intercepting traffic and testing for vulnerabilities. Using Burb's site map feature, we explored the iSpy web interface to list all the accessible endpoints. This step is crucial in identifying if there's an entry point for security threats.


Agent
localhost:8090/#Live
Burp Suite Community Edition v2024.3.1.4 - Temporary Project
Burp Project Intruder Repeater View Help
Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Organizer Extensions Learn Settings
Site map Scope Issue definitions
Pro version only
Request
Response
Pretty Raw
Inspector
Notes

Site map filter: Hiding not found items; hiding CSS, image and general binary content; hiding 4xx responses; hiding empty folders

https://accounts.google.com
http://localhost:8090
https://play.google.com
https://www.ispyconnect.com

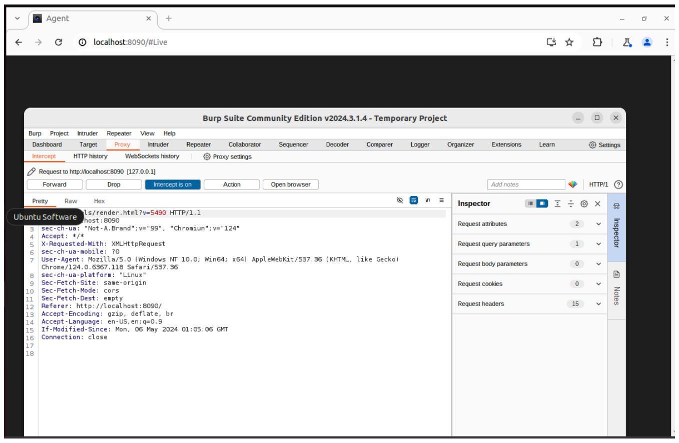
Host	Method	URL	Params	Status Code	Length	MIME type	Title	Notes
http://localhost:8090	GET	/partials/edit_layout_d...						
http://localhost:8090	GET	/partials/edit_layout_pl...						
http://localhost:8090	GET	/partials/edit_plan.html						
http://localhost:8090	GET	/partials/payment_hist...						
http://localhost:8090	GET	/partials/presets.html						
http://localhost:8090	GET	/partials/render.html						
http://localhost:8090	GET	/partials/render_sideb...						

```
GET / HTTP/1.1
Host: localhost:8090
sec-ch-ua:
"Not-A.Brand";v="99",
"Chromium";v="124"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Linux"
Upgrade-Insecure-Requests:
1
User-Agent: Mozilla/5.0
(Windows NT 10.0; Win64;
x64) AppleWebKit/537.36
(KHTML, like Gecko)
Chrome/124.0.6367.118
```

```
HTTP/1.1 200 OK
Content-Type: text/html
Date: Sun, 05 May 2024
20:52:46 GMT
Last-Modified: Mon, 06 May
2024 01:05:06 GMT
Access-Control-Allow-Origin
: *
Cache-Control: no-cache,
no-store, must-revalidate
Pragma: no-cache
Expires: 0
Vary: *
Server:
```

Request attributes	2
Request headers	14
Response headers	11

The screenshot above displays sample requests intercepted from the iSpy web interface. This shows the structure of the application and the types of data exchanged between the client and the server. Intercepting traffic enabled us to exampine the HTTP and HTTPS traffic between the client and the iSpy servers. Intercepting traffic also helped us to observe the data flows and where sensitive information might be exposed.


Agent
localhost:8090/#Live
Burp Suite Community Edition v2024.3.1.4 - Temporary Project
Burp Project Intruder Repeater View Help
Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Organizer Extensions Learn Settings
Intercept HTTP history WebSockets history Proxy settings
Forward Drop Intercept is on Action Open browser
Add notes
HTTP/1
Pretty Raw Hex
Ubuntu Software
Inspector
Notes

Request to http://localhost:8090 [127.0.0.1]

```
ls/render.html?v=5490 HTTP/1.1
host:8090
sec-ch-ua: "Not-A.Brand";v="99", "Chromium";v="124"
Accept: */*
X-Requested-With: XMLHttpRequest
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/124.0.6367.118 Safari/537.36
sec-ch-ua-platform: "Linux"
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://localhost:8090/
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9
If-Modified-Since: Mon, 06 May 2024 01:05:06 GMT
Connection: close
```

Request attributes	2
Request query parameters	1
Request body parameters	0
Request cookies	0
Request headers	15

The screenshot above is a visual aid that demonstrates how traffic is intercepted and analyzed, highlighting the details of HTTP requests and responses. Through traffic intercepting, we can see that some data transmissions are unencrypted. This must be due to the use of HTTP. Upgrading to HTTPS for all communications would enhance security. Overall the penetration testing conducted with Burp suite provided good insights to the security aspect of iSpy. Regular reviews and updates are recommended to ensure the system remains secure and protected against potential cyber threats.

VI. Summary

This project involved analyzing the security of the source code for the iSpy camera system software. By using different methods and techniques, we determined a number of requirements that the software should have and identified areas of the software that lacked these requirements. We also proposed potential solutions to these security gaps.

VII. Conclusion

Software is a core component of modern day life. Many of the essential functions of society are run by software: the iSpy camera system is no exception. It can do anything, from protecting a household or business to observing wildlife. However, if the software that runs the system is not secure, it could allow malicious actors to enter and breach confidentiality, integrity and/or availability (the three fundamental pillars of cybersecurity). Therefore, it is important to conduct code analysis and security assessment procedures to identify potential vulnerabilities and issues in the software that could let attackers in. By maintaining its security, everyone else's security can be maintained.