Data Protection Method of an Unmanned Aerial Vehicle based on Obfuscation Procedure

Serhii Semenov [0000-0003-4472-9234], Denys Voloshyn [0000-0002-1077-9658], Viacheslav Davydov [0000-0002-2976-8422]

National Technical University Kharkiv Polytechnic Institute, Kharkiv, Ukraine
ultrageron@gmail.com

Abstract. This article discusses the data protection method for unnamed aerial vehicle (UAV) embedded control system. This method is based on obfuscation mechanism in Embedded Java. The principles of software development for embedded systems are highlighted. Protecting methods of the embedded systems of UAVs are analyzed. Methods: Principle of string-related obfuscated code based on the pseudorandom number generating (PRNG) sequences is given. The algorithm of obfuscating identifier names using a global unique identifier is shown. The method of synchronous and asynchronous encryption of the UAV control signal is described. Results: A comparative analysis of the main programming languages for embedded systems, their disadvantages and advantages. The need for a real-time operating system for UAVs is explained. Disclosures of obfuscated code on embedded control systems are shown. To compare the cracking difficulty, an experiment was conducted with the participation of information technology specialists. Results showed the expediency of using this method for data protection for UAV embedded control system. Conclusions: Result of the study is determination a time needed to study internal code of the UAV embedded control system with and without proposed data protection method.

Keywords: Unmanned Aerial Vehicles, UAV, Embedded Java, Obfuscation, Data Protection Method.

1 Introduction

1.1 Motivation

The analysis of most unmanned aerial vehicles (UAV) samples showed that one of the main functional elements of UAVs during flight missions is an embedded control system. Its which protection is an important component of the entire UAV operation.

The architecture of the embedded control system consists of hardware and software components. The hardware component in most cases is produced at the factory and has limitations in the modification of the structure. This introduces additional problems in the implementation of customization which is not provided in the design documentation. Therefore, in the process of improving UAVs, the software component take place a special role. In most cases that's because of:

1. It is implemented by each organization in its own way. That's why software is highly customized.
2. Any software is intellectual property.
3. Requires an additional degree of security due to the fact that it's compromise affects the communication protocols and the organization itself.

The software component of the integrated UAV control system consists of the following components:

- Data.
- Algorithms directly related to this data.
- Protocols.
- Geo-location data of the UAV.

To compare the complexity of cracking difficulty, an experiment was conducted with the participation of information technology specialists. The loss of each components leads to a decrease in the reliability of the data obtained.

1.2 Analysis of related work

Studies of the literature have shown that it is necessary to use data encryption [1,2] in order to increase the security of UAVs under external influences. In particular, symmetric encryption allows you to hide control commands from third parties, however, if UAVs are intercepted by third parties, the encryption key will be compromised. Asymmetric encryption allows you to use a private key on the transmitter side, and use a public key on a UAV. In this case, even if device is completely cracked, it will not allow you to obtain information about the private key. And since several UAVs are more often assumed to be in the air in one-time interval, the usage of asymmetric encryption is an extremely important factor for ensuring the safety of other devices [1,3].

Nevertheless, data protection of internal UAV systems remains an important task, given that in addition to encryption algorithms and transmission protocols for UAVs, there is data on its geo-location throughout the flight.

Consider an exceptional situation where an UAV is catch by attackers whose task is to study the vulnerabilities of the software components of the device. These vulnerabilities could be used in order to subsequent attacks aimed at similar UAVs. Perhaps the most reliable option to secure the internal structure of the device is the use of its own architecture in microcontroller systems. However, this method is not viable due to the expensive and time-consuming for the tasks. Therefore, the main emphasis on data protection should be focused on the software part of the device.

Studies in [3] showed that software for embedded systems today is written in C, Embedded Java, and, in rare cases, assembler. However, with the increasing complexity of the tasks programmatically implemented in embedded systems, more and more developers choose Embedded Java as main programing language for embedded control system developing. Another advantage of this language is that it has constant updates and improvements [5]. In addition, due to the vast popularity of the Java language in training, as well as the demand for embedded developers, it is advisable to use Embedded Java for this kind of devices.

The main advantage of Java is the strictly regulated data types based on the object-oriented approach and the absence of pointers, which reduces the number of errors caused by the developer. Another advantage of Java is portability, because of usage of Java Virtual Machine (JVM), where code is executed [6]. Java is the language where obfuscation mechanisms are still in active development. Further improvements of obfuscation mechanisms based on Java application is important scientific task as code obfuscation mechanism complicates the process of code research to search for vulnerabilities in UAV’s embedded control system.

1.3 Goals

The main objective of the study is to develop a method for ensuring the safety of the integrated UAV control system. Finding vulnerabilities in existing methods of organizing the protection of the integrated UAV control system. In order to find specific parameters for the complexity of code cracking, time measurements are required. Time is an important criterion in ensuring the safety of communications between the command post and the UAV. Also, code obfuscation can lead to hiding some features of the embedded control system. Searching for alternative solutions for using microcontroller systems to increase the performance of the proposed method of code obfuscation.

2 Obfuscation in Embedded Java

2.1 Obfuscation of identifier names

Historically, obfuscation of identifier names has been used in the production of J2ME applications to reduce the volume of the resulting application by reducing any identifiers to 1-2 letter names.

Since modern compilers support identifier names in UTF-8 format, the development of identifier obfuscation was the reduction of identifier names to one UTF-8 character, which was visually "read" worse. However, this approach to obfuscating identifier names had a drawback - it was possible to take decompiled code, make changes to it and recompile without difficulty.

The development of this approach was the renaming of identifier names into language keywords, such as “do”, “while”, “for”, which prevent uncomplicated recompilation.

An analysis of the described implementations of obfuscation of identifier names showed their common drawback - within the scope they have the same name, which makes it possible to trace the set of actions performed with the same identifier.

Programming languages do not allow an identifier name construction during runtime - it must be determined at the compilation stage. In order to increase the confusion of the application code, the usage of a global associative array is proposed, where the key is the string - the name of the identifier. This gives us the opportunity to access to read and write an associative array element by key, which will be generated on the fly, which makes it difficult for an attacker to follow the program's progress without debugging.

```
int a = 10;
System.out.println(a);
```

Upon transition to an associative array it will turn into a construction.

```
GLOBAL_MAP.put("a", 10);
System.out.println(GLOBAL_MAP.get("a"));
```

And for this case, the usage of the string obfuscation approach based on the operation of pseudorandom number generating (PRNG) will lead to such a design that does not give a logical connection between the identifiers.

```
GLOBAL_MAP.put(ConvertString(new
long[]{2285910032648232766, 6912903053422632332}), 10);
System.out.println(GLOBAL_MAP.get(ConvertString(new
long[]{4026505391851356825, -3154998743502247311})));
```

It should be noted that the global associative array has the following features:

- All keys of this array must be unique.
- Identifiers with the same name may be present in different methods and classes.
- A class can have several instances, and therefore it is necessary to identify an instance of the class.
- Functions that use variables can be overloaded.

Due to these limitations, it was decided to use the following identifier format:

classname@hash:*global_class_id*,

where:

classname – is the full name of the class (along with the package);

hash is the hash of the current instance of the class. If the identifier is global, then the value is empty;

global_class_id – is unique identifier in the context of the current class.

This makes it possible to avoid the necessity to determine the scope and the presence of overloaded functions.

The algorithm for obfuscating identifier names in this way can be represented by the algorithm diagram presented in Fig. 1.

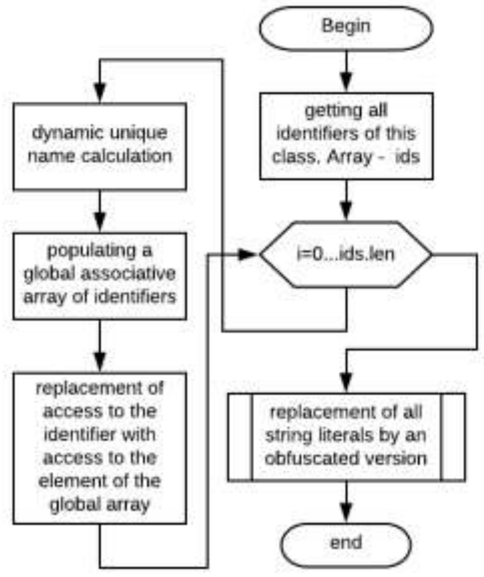

Begin
getting all identifiers of this class. Array - ids
i=0...ids.len
dynamic unique name calculation
populating a global associative array of identifiers
replacement of access to the identifier with access to the element of the global array
replacement of all string literals by an obfuscated version
end

Fig. 1. Algorithm Scheme of identifier name obfuscation using a global unique identifier.

2.2 Obfuscation of logical expressions

As part of the study of this section, it was proposed to use the features of simplifying logical expressions. So, the following simplification constructions are considered:

- usage of ternary operator:

$$[\ (a)\ ?\ 1 : 0\ ]\ \ != 0,$$

the selected fragment in square brackets can be repeated an infinite number of times, substituting itself instead of the number 1, and can be collapsed into expression (a), which in this case will lead to the fact that the condition with such a ternary operator will always be "true".

Thus, introducing an additional ternary operator into a logical expression will introduce additional difficulty in code analysis.

The C-based languages like C, C ++, Java, C# have a feature of logical expressions that allows you to speed up the program in special cases:

- in expression (a && b), condition b, which may include a complex condition or even a function call, will not be executed if condition a takes the value FALSE (since in this case the value of condition b does not play a role - the result of the expression will be FALSE)
- in the expression (a || b), condition b, which may include a complex condition or even a function call, will not be executed if condition a takes the value TRUE

(since in this case the value of condition b does not play a role - the result of the expression will be TRUE)

- the usage of Boolean Algebra Laws to simplify logical operations:

  - Double negation. The equivalence formula has the form:

$$(a) === (!!a).$$

  - Excluded third. Equivalence formulas have the form:

$$\text{false} === (a \&\& !a),$$

$$\text{true} === (a \,||\, !a).$$

  - Work with constants. Equivalence formulas have the form:

$$(a) === (a \&\& \text{true}),$$

$$\text{false} === (a \&\& \text{false}),$$

$$\text{true} === (a \,||\, \text{true}),$$

$$(a) === (a \,||\, \text{false}).$$

  - Repetition. The equivalence formula has the form:

$$(a) === (a \&\& a) === (a \,||\, a).$$

  - Absorption. Equivalence formulas have the form:

$$(a) === (a \,||\, (a \&\& b)),$$

$$(a) === (a \&\& (a \,||\, b)).$$

Entangling strings has long roots. One of the oldest methods of string obfuscation was Caesar encryption, which is one of the varieties of substitution ciphers.

The main vulnerability of the cipher lies in the fact that an attacker can figure out, by frequency analysis, exhaustive search, guessing, or in some other way, the correspondence between any two letters of the source text and ciphertext. Then the key can be found by solving a system of equations.

The development of this obfuscation block was the use of modern symmetric encryption systems, built, for example, on the symmetric Rijndael block encryption algorithm. However, such algorithms have the following disadvantages, in connection with which they did not receive mass distribution as string obfuscation methods:

- Decryption speed. Symmetric encryption algorithms are built on the principle of long-term processing to reduce the likelihood of hacking by brute force for a given period of time. Thus, the use of this approach will lead to a noticeable increase in runtime, which, nevertheless, is permissible in critical sections of code that are executed singly (for example, when starting the application - checking the license key) and the speed of the application in this section is not important.

- Static key. The keys to decrypt a block of text (in most cases) are the same for all obfuscated blocks of text.

One of the algorithms for obfuscating strings is an algorithm based on the fact that all characters have code that can be operated on with normal numbers. Examples of characters:

- char a = ‘x’ – non-obfuscated version of the letter "x";
- char a = (char) 120 – representation of the character "x" by its code;
- char a = (char) 125 – 5 – representation of the symbol "x" by its code obtained by calculating a mathematical expression;
- char a = (char) (‘y’ - 1) – representation of the character "x" as a character offset from the character "y" by one back.

To increase the level of confusion, this array of characters is converted into a set of strings, each of which creates its own character. For example, the line

```
string data = "sec",
```

can be converted to sequence

```
string data;
data[0] = 's';
data[1] = (char)('b' + 3)
data[2] = (char)(25 * 3 - 4);
```

The proposed method consists in the particularity of the PRNG: for the same initial seed value, the sequence of numbers always turns out to be the same. As a result of the method, an array of numbers is obtained that reflects the array and, due to the symmetry of the algorithm, can be converted back to a string.

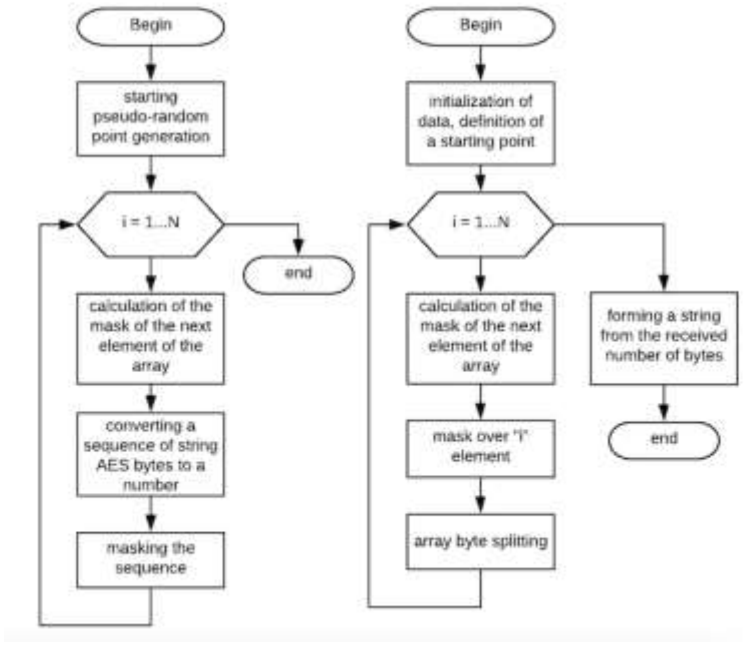
In modern operating systems, the longest and largest data type is ‘long’, which occupies 8 bytes in memory. Depending on the language features of the text, up to 8 characters can be stored in a given amount of memory (provided that they are all single-byte, for example, Latin characters, numbers). When using UTF-8 encoding, the number of characters that can fit in 8 bytes is reduced.

In order to convert a string to a number, the number of variables of type long is 8 times less than the size of the string in bytes.

Due to the fact that the developed method is based on the initial value for the function of pseudo-generation of numbers, the resulting array will contain one value more than expected.

2.3 The obfuscation algorithm

A schematic description of the string obfuscation algorithm based on the features of the PRNG is presented in Fig. 2. Consider the implementation details.



Begin
starting pseudo-random point generation
i = 1...N
end
calculation of the mask of the next element of the array
converting a sequence of string AES bytes to a number
masking the sequence
Begin
initialization of data, definition of a starting point
i = 1...N
calculation of the mask of the next element of the array
forming a string from the received number of bytes
end
mask over "i" element
array byte splitting

Fig. 2. Algorithm of strings obfuscation and deobfuscation.

Step 1. Initialization of data. At this step, memory is allocated for the resulting array according to the formula:

$$cnt = GVS + (Size \,/\, AES) + ((Size\% \; AES == 0) \; ? \; 0{:}\; 1),$$

where:

size - size of the source string in bytes,

GVS (Generated Value Size) - the number of array elements that will occupy the value of the starting point of the pseudo-random number generator (in this example, 1 element),

AES (Array Element Size) - the size of one element of the resulting array in bytes (for the long type - 8 bytes).

A pseudo-random number is also generated, which will be the “starting” point for the operation of the pseudo-random number generator. This number is generated depending on the current time, which means that the two output sequences of the same line will be completely different.

The first GVS elements of the array are filled with a pseudo-random number - the starting point of the PRNG.

Step 2. Filling in the remaining elements of the output array.

Step 2.1 For each subsequent element of the array, the next pseudo-random number is calculated, which will act as a “bit mask” for hiding data.

Step 2.2 Converts the next AES bytes of a string to a number by representing them as composite bytes of a number. For example, the string ABCD is converted to a number: 0x44434241

Step 2.3 Overlay the mask obtained in step 2.1 with the number obtained in step 2.2.

2.4 Deobfuscation Algorithm

Step 1. Initialization of data. At this step, memory is allocated for the resulting string (in bytes) according to the formula:

$$Size = (cnt - GVS) * GVS \quad (9),$$

where:

GVS (Generated Value Size) - the number of array elements that will occupy the value of the starting point of the pseudo-random number generator (in this example, 1 element),

AES (Array Element Size) - the size of one element of the resulting array in bytes (for the long type - 8 bytes),

cnt is the number of elements in the source array.

The PRNG is initialized according to the starting point, which is calculated from the data of the first GVS elements of the array. So, in the presented example, the 0th element of the array is the starting point.

Step 2. Formation of a byte array of the resulting string.

Step 2.1 For each subsequent sequence of AES bytes, a “mask” is formed according to the PRNG algorithm.

Step 2.2 The mask is superimposed on a sequence of raw AES bytes. The result is an array similar to that obtained from Step 2.2 of the obfuscation process.

Step 2.3 Splitting a number of AES bytes into AES of individual bytes and adding them to the resulting byte array.

Step 3. Formation of a string from an array of bytes according to the selected encoding. In the UTF-8 encoding used, there is no clear correspondence 1 byte = 1 character, therefore, the size of the resulting string may be less.

The development of this algorithm is the modification of the output sequence not as an array of numbers (2, 4, 8 - bytes), but as a byte array or BigInteger number. Using this approach, the need for alignment will be eliminated, which makes it possible to make the value of the PRNG starting point of practically unlimited length (including for using its own PRNG), as well as using a mask whose size is not a multiple of the selected type of array element (AES) - so, when the element type of the resulting array is long, each element will occupy 8 bytes, and, for alignment matching, it is necessary that the mask length be 1, 2, 4, 8 bytes.

The subsequent development of the proposed algorithm is the use of converting numerical values to a string and repeating the algorithm a specified number of times.

3 Results

An experiment that shows the dependency of cracking obfuscated and non-obfuscated code time was conducted as part of the study. A module has been developed for the UAV control embedded system. Is has been processed via the developed obfuscation methods. The purpose of the experiment is to determine how much time it takes for IT specialists to unravel the algorithm and automate the process of obtaining the necessary data. A sample of 100 people was made. The result showed that for unraveling the code obfuscated by the given algorithm, on average, 5.32 hours are spent with a standard deviation of 1.16 hour. Fig.3 shows a graph in which the y-axis indicates the time for hacking, and the x-axis indicates the number of specialists. This fact makes it possible to conclude that the time required to unravel the algorithm is sufficient to 1) conduct an anti-terrorist operation and seize the UAV until the final unraveling of the algorithm; 2) take measures to update the encryption keys of other UAVs in order to exclude the stolen UAV from the list of communicating ones.

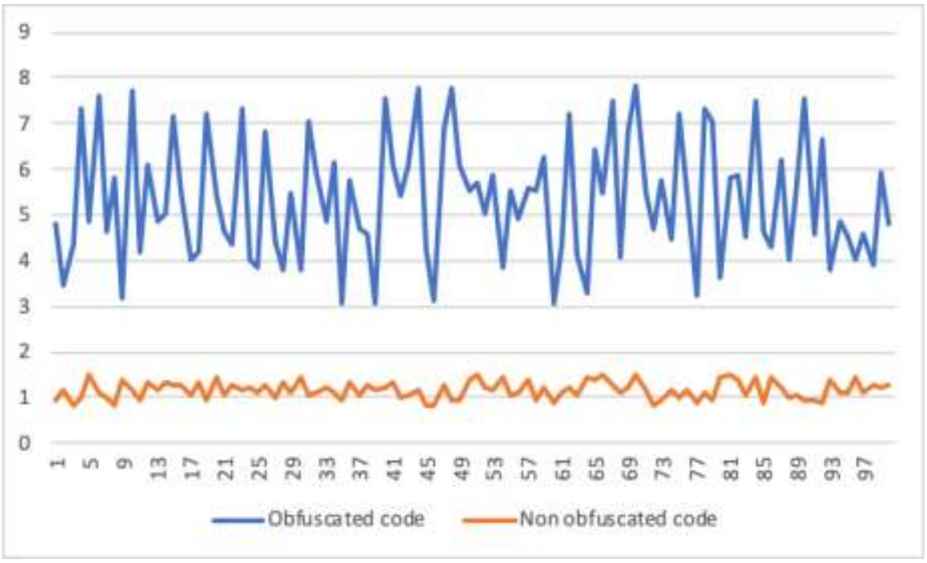


Fig. 3. Diagram of time comparison

4 Conclusions

UAV control systems require a real-time operating system for timely response to external events. However, obfuscation algorithms, including those introduced, degrade system performance (while leaving it to work within an acceptable framework). The solution to this problem is to transfer the software implementation of obfuscation to the hardware. This will allow us to: 1) speed up the data processing; 2) use an epoxy polymer in order to conceal the structure of the circuit and minimize the possibility of recovery. For UAV embedded control system its necessary to have software methods to protect internal structure from malefactor activities. That's provide a new opportunity for development and modification, in case increased source code security. Average, it takes 5 times more time to crack an obfuscated code than non-obfuscated code. That provide using proposed obfuscation method for code protection for embedded system developers.

References

1. Zeng, Y., Zhang, R., Lim, T.J.: Wireless communications with unmanned aerial vehicles. Opportunities and challenges, IEEE Commun. Magazine 54(5), 15–35, (2016).
2. Semenov, S., Sira, O., Kuchuk, N.: Development of graphicanalytical models for the software security testing algorithm. Eastern-European Journal of Enterprise Technologies 4 (92), 39-46, (2018). DOI: https://doi.org/10.15587/1729-4061.2018.127210
3. Hossein Motlagh, N., Bagaa, M., Taleb, T.: UAV-Based IoT Platform: A Crowd Surveillance Use Case.IEEE Communications Magazine 55(2), 1-4 (2017).
4. Kolding Foged-Ladefoged, K.: Real-Time Embedded Systems in Java.Master Thesis, 3-4 (2016).
5. Oleksandr S. Moliavko, Taras A. Drozdovskyi, Vitalii M. Petrychenko, Oleg E. Kopysov: Computational Reflection for Embedded Java Systems. The Journal of Open Source Software 4(36), 2-5 (2019).
6. Di Stefano, A., Fargetta, M., Tramontana, E.: Computational Reflection for Embedded Java Systems. Lecture Notes in Computer Science, 3-8, (2003).
7. Melnikov V., Kleimenov S., Petrakov A.: Information Security and Information Protection. 3rd ed. Training Allowance for stud. higher studies. institutions, Moscow - Academia (2008)
8. Programming Concepts: Compiled and Interpreted Languages. https://thecodeboss.dev/2015/07/programming-concepts-compiled-and-interpreted-languages, last accessed 2019/10/28.
9. Collberg, C., Thomborson, C., Low, D.: Taxonomy of Obfuscating Transformations. Department of Computer Science, The University of Auckland (1997)
10. Replacing the conditional operator with the polymorphism https://refactoring.guru/ru/replace-conditional-with-polymorphism, last accessed 2019/10/28.