Reduction of Data Leakage Using Software Streaming

Sung-Kyung Kim[1], Eun-Tae Jang[1], Seung-Ho Lim[2], and Ki-Woong Park[1](✉)

[1] Department of Information Security, Sejong University, Seoul, South Korea
jotun9935@gmail.com, euntaejang@gmail.com , woongbak@sejong.ac.kr

[2] Hankuk University of Foreign Studies, Yongin, South Korea
lim.seungho@gmail.com

Abstract. With the increase in threats to IoT devices, interest in protecting sensitive data within such devices has intensified. For devices holding sensitive data and intellectual property software, such as military equipment, leakage of the confidential data contained within the device can cause catastrophic damage. Therefore, it is important to prevent such leakage of sensitive data. In this paper, we propose a method for reducing data leakage from military devices by minimizing the quantity of data that exist within the non-volatile memory of the device. To achieve minimization of the data loaded in non-volatile memory, we run the software in a streaming manner. However, as the execution of software over a network can result in suspension of the software depending on the state of the network, this approach can have a critical impact on system stability. Therefore, we also present a scheme to apply multi-channel communication to reduce software suspensions caused by network delays when the software is run in a streaming manner for the purpose of mitigating damage to the data leakage.

Keywords: Software streaming · Network channel scheduling · On-demand computing · Self destruction · Disposable computing · Muti-network channel

1 Introduction

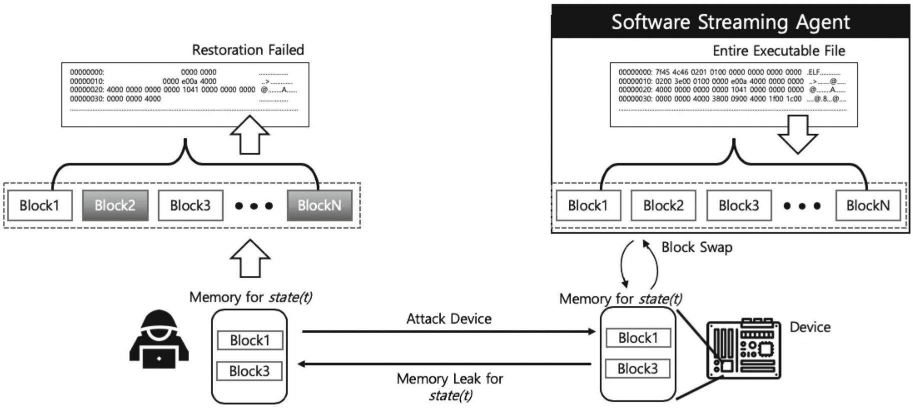
Mobile devices used for various purposes have increased with the development of IoT technology. The military industry is one of the leading areas that has been affected by the development of IoT technology. Many advanced military equipment, including reconnaissance drones and information gathering equipment,

This work was supported by the National Research Foundation of Korea (NRF) (NRF-2020R1A2C4002737) and the Institute for Information and Communications Technology Promotion (IITP) grant funded by the Korea government (MSIT) (No. 2018-0-00420).

I. You et al. (Eds.): MobiSec 2019, CCIS 1121, pp. 99–111, 2020.
https://doi.org/10.1007/978-981-15-9609-4_8

use computing systems. However, these military devices may contain confidential data, and the leakage of such data can cause significant losses. For example, in 2008, Iraqi militants bought a hacking program from a Russian hacking site and hacked the US recon drone RQ-170 modeled predator [1]. They hacked and leaked video footage that showed the predator shooting. In December 2011, a UAV stealth RQ-170, co-produced by US Lockheed Martin (USA) and Israel, was captured by Iran's GPS-managed attacks while scouting Iranian territory [2]. The Iranian government reverse-engineered the captured drones, and after two years they replicated and tested a similar drone. The above examples show that defensive techniques are required to minimize the risk of information leakage, when operating a device that can store sensitive data. Therefore, there is a need for secure disposable computing, which can guarantee complete erasure of data from state-of-the-art devices where data leaks can lead to significant losses. In this study, we propose a secure system framework that prevents data leaks through the concept of Disposable Computing to protect sensitive data in military mobile devices. In order to effectively and safely apply disposable computing to devices that require real-time computing, such as military equipment, it is necessary to minimize non-volatile and volatile memory, which contain sensitive information. We utilized software streaming technology, which is a method to use software in real-time by sending segmented software over a network. Figure 1 shows an overview of the software streaming technology to prevent exposure of sensitive information on embedded devices by minimizing the non-volatile and volatile memory.

- **Minimized Non-volatile Memory**
  Data is retained in the non-volatile memory even when the power is cut off; hence, it must be minimized to reduce the size so that sensitive data do no remain in the device. In the event that this information is captured, it will be limited.



```
00000000:                     0000 0000                ..........
00000010:                0000 e00a 4000                ..>.......
00000020: 4000 0000 0000 0000 1041 0000 0000 0000      @.......A.....
00000030: 0000 0000 4000                               ..........
```

```
00000000: 7f45 4c46 0201 0100 0000 0000 0000 0000  .ELF..........
00000010: 0200 3e00 0100 0000 e00a 4000 0000 0000  ..>.......@....
00000020: 4000 0000 0000 0000 1041 0000 0000 0000  @.......A.....
00000030: 0000 0000 4000 3800 0900 4000 1f00 1c00  ....@.8...@....
```

Fig. 1. Overview of software streaming that can prevent the leaking or disclosure of sensitive information in a device

- **Minimized Volatile Memory**
  Even if the memory state is leaked by the attacker at one point, the size of the volatile memory should be minimum so that the device can operate to minimize the sensitive data on the volatile memory. Figure 1 shows a scenario in which the block information $state(t)$ loaded into the device's volatile memory at time t is leaked. If a block is leaked, it is difficult to infer the whole program through it. In addition, since the blocks loaded in the volatile memory are continuously alternated, it is difficult to obtain the entire program through the nonvolatile memory dump even if execution continues to the end of the program.

However, the primary limitation of the existing software streaming technology is that it has execution delays due to network transmission. Therefore, when applying the technology to a real-time system that requires real-time responses, such as a military device, it is necessary to minimize the challenges posed by the execution overhead. To overcome this challenge, we used streaming data scheduled through multiple streaming channels and a pre-generated software profile data. The contributions of our research can be summarized as follows:

1. **Software streaming for intellectual property (IP) protection.** Our approach on software streaming technology was from the perspective of protecting software intellectual property. Various techniques have been studied to prevent the leakage of sensitive information and software in embedded systems. Software streaming technology can also be used as a leakage prevention method. This is because when software is transmitted through the streaming method, information in the memory can be minimized in the situations where the target system is hijacked.
2. **Identification of bottlenecks in previous software streaming.** We identified a bottleneck caused by the complexity of function call in the existing software streaming techniques. Therefore, we proposed a software streaming method that can be applied to a real-time system by minimizing the bottlenecks, as described in detail in Sect. 3.

The rest of this paper is organized as follows: Section 2, examines existing software protection work and related research on software streaming. In Sect. 3, we propose a method for efficient software streaming that applies the concept of Disposable Computing to secure a Real-time Embedded System safely. Section 4, proposes network channel scheduling based on a round-robin for efficient software streaming. We propose areas of further research for framework development in Sect. 5 and the conclusions are presented in Sect. 6.

2 Related Work

Researchers have developed various methods to prevent the leakage of sensitive data from desktop and mobile environments, include anti-analysis techniques and encryption methods. Analysis prevention techniques include obfuscation [3],

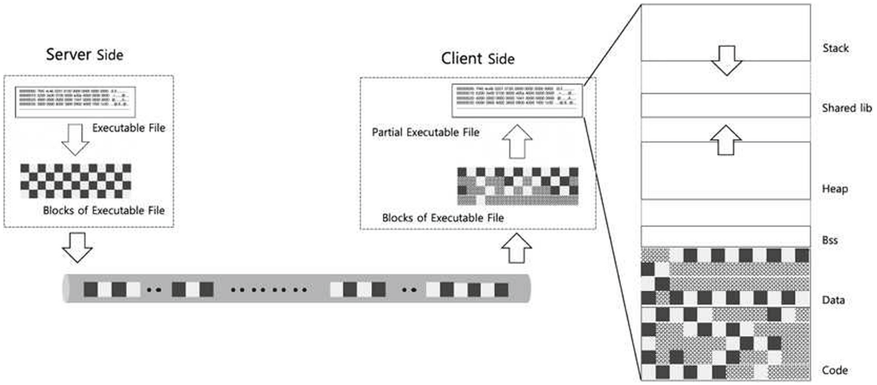


Fig. 2. Software streaming between server and client

instruction virtualization [4,5], anti-debugging [6,7], and binary code packing [8]. All these features make software difficult to analyze, but it can be analyzed by a skilled analysts [9,10]. The file system encryption method [11] is a representative example of the encryption method, a method of protecting important data by encrypting the file system itself in which data is stored. However, decryption results in a system overhead making it challenging for it to be used in real-time embedded devices. In this study, we overcame this challenge by utilizing software streaming technology which partially receives and executes software in the network.

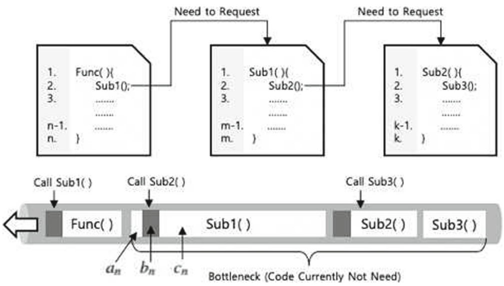
Kuacharoen et al. used Block Streaming for software Streaming [12]. Their work uses binary rewriting technology to ensure that the transmitted software could be executed continuously. The method inserts a code that requests the server for the next code to be executed as it not present in the memory of the partitioned block following partitioning by executable file in the server. If the system executes all the code within the block or enters the off-the-block position by the *jmp*/*ret* command, the code for that part is transmitted on the network. We also investigate the PoC of real-time code execution by applying software streaming to embedded devices. However, its primary limitation is that it causes high execution time overhead that results in continuous application suspension until the code is sent remotely when it is not located in memory. This approach is therefore difficult to apply in embedded system environments that require real-time response, such as real-time systems.

Kuacharoen et al. developed a previous study to devise a way to increase efficiency by sending blocks from single block unit transmission to function units [13]. They increased the transmission efficiency by repositioning functions within the blocks so that associated functions could be in a single block, and by removing unused blocks from memory, they were more efficient than reported in previous studies on memory management. However, in our study, bottlenecks occurred depending on the complexity of the function call and resulted in the suspension of applications.

Choi, Jeong-dan et al. designed a method [14] to add pre-processing steps of code clustering & pre-fetched map generation to change the application to a form that could be streamed. This is because code clustering is based on function unit blocks, and the pre-fetched map is the result of the PPM (prediction by partial matching) algorithm that generates many of the code cluster transmission histories.

3 System Architecture

Most execution delays in streaming software execution are caused by application suspension, which results in a waiting period while receiving code to be executed remotely [12]. In systems that require immediate response, such as real-time embedded systems, this waiting period/suspension can have a devastating effect on the stability of the entire system. Therefore, to minimize application suspension, research on background streaming, software profile [12], block relocation [13], and pre-fetched map [14] have been conducted. In general, in these studies code was divided into functions and classes. However, the partitioning method takes a long time when receiving and executing a complex function, and a network delay occurs that delays the reception of a function to be executed next, resulting in application suspension. Figure 3 shows the bottleneck that can occur when the function call complexity is high in an existing software streaming method. If the function $Func()$ is executed, the actual software execution is executed sequentially by the $Sub1()$, $Sub2()$, and $Sub3()$ functions inside the $Func()$ function according to the function call relationship. However, when streaming it uses a single network channel, $Func()$ does not end until $Sub3()$ is received and execution is completed; hence, a delay occurs in the execution of the next function.



```
Func( ){
    Sub1();
    .......
    .......
    .......
}
```

```
Sub1( ){
    Sub2();
    .......
    .......
    .......
}
```

```
Sub2( ){
    Sub3();
    .......
    .......
    .......
}
```

Fig. 3. Bottleneck challenges in a single-channel software streaming via block streaming

It can be expressed by the following formula. The size of the n^{th} sending block ($size_n$) can be expressed as a code before the call-routine (a_n), a call routine code (b_n), and a code to be executed after the function call returns (c_n).

The cost of executing the n^{th} block in the nested call routine is the accumulated value of the $n-1^{th}$ block size from the beginning of the block plus the size of the code from the beginning of the nth block to the call-routine.

$$cost_n = \sum_{k=1}^{n-1} size_k + a_n + b_n$$

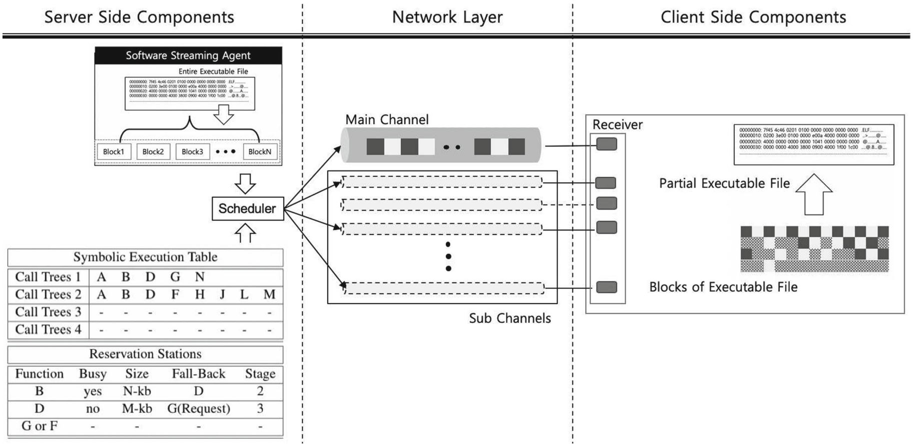
$$size_n = a_n + b_n + c_n (0 \leq a_n, b_n, c_n \leq MAX\_BLOCK\_SIZE)$$

Where, c_n is executed after a series of call routines are completed in a sequential execution structure. Therefore, the method of receiving c_n by lowering the priority of transmission a_n and b_n can improve execution efficiency in software streaming. Therefore, our aim was to design a software streaming that minimizes the overhead of c_n transfers when transferring blocks that have nested call routines.

$$ideal\_cost = \sum_{k=1}^{n} (size_k - c_k)$$

3.1 Network Multi-channel for Software Streaming

When using only one channel for software streaming, as shown in Fig. 4, when a large size function is received from the server, there is a network delay and it does not receive the next function to be executed until the current function is completed. For systems that require real-time response, such as real-time systems used in the military, execution delays can be fatal in that they can cause an overall system failure. One of our objectives was to minimize execution delay. As shown in Fig. 4, we propose a multi-channel configuration consisting of main



Symbolic Execution Table								
Call Trees 1	A	B	D	G	N			
Call Trees 2	A	B	D	F	H	J	L	M
Call Trees 3	-	-	-	-	-	-	-	-
Call Trees 4	-	-	-	-	-	-	-	-

Reservation Stations

Function	Busy	Size	Fall-Back	Stage
B	yes	N-kb	D	2
D	no	M-kb	G(Request)	3
G or F	-	-	-	-

Fig. 4. Overview of the concept of multi-channel software streaming

and sub-channels to minimize delays by scheduling to other data channels immediately when a streaming delay occurs. In addition, we developed a method for the function to occupy the channel efficiently.

Main Logic Streaming Channel. The server separates the software requested from the client into functions and sends them in blocks. A network channel between the server and the client is used when sending a block and this referred to as the Main Channel. In case of a delay, a standby on the Main Channel, it transmits it using another channel (Sub Channel).

Sub Logic Streaming Channel. This channel is a type of sub-channel established to reduce network overhead and delay. When the server sends the software, block requested by the client, it uses the Main Channel to send it to the client in real-time. However, if the block size requested by the client is large, the waiting time of the next block to be received is increased. Thus, we proposed a method for building a sub-channel in addition to the main channel. Therefore, if the size of the block received from the client is large and the waiting time increases, the task continues by moving the block to the Sub Channel, and the main channel becomes free to receive the next request block from the client.

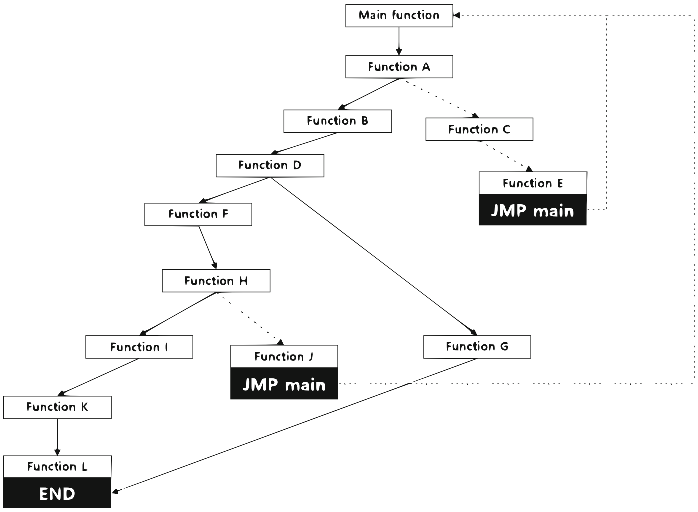


Fig. 5. The process of finding the *function N* requested by the client using symbolic execution

3.2 Symbolic Execution

The client must request the server for streaming the next block to be executed. Symbolic Execution for efficient software streaming was used and the results were saved in a table to enable efficient communication between server and client. The process of identifying all the paths from the main function to the destination function N requested by the client using symbolic execution when the client requests Function N is shown in Fig. 5. Symbolic Execution can be used to identify a successful path condition (e.g. A, B, D, G, L) for the destination and write the results in the Symbolic Execution Table. The key to Symbolic Execution is to run it in multiple ways. Thus, when the program runs, it will follow every branch statement and generate inputs until the user meets the path condition. The software was divided into functions in the server, the table for the Call Tree was created using symbolic execution and the data required for the function corresponding to the client's request were sequentially provided.

The Tomasulo algorithm places Reservation Stations to create information about the commands that should be executed per CPU cycle [15]. The Tomasulo algorithm eliminates WAR/WAW hazards through Register Renaming and allows for sequential execution of instructions. Register Renaming is a method used by Tomasulo's Algorithm to perform out-of-order execution.

Function scheduling was based on the Register Renaming used in the Call-Tree-Table of the software obtained using Symbolic Execution and Reservation Stations of the Tomasulo algorithm. Reservation Stations consult the Symbolic Execution Table for information on the next functions to be sent. The Reservation Stations element contains the function to be processed, Busy (State), Function Size, Fall-Back, and the current progress. A fall-back field was placed in the reservation station to minimize the hazards that could occur in software streaming. A table for software streaming based on the Tomasulo algorithm is given in Table 2. For example, when a client requests function N, such as shown in Fig. 5, two Call Trees are created, as shown in Table 1. Therefore, we proposed that function B, function D, must be transmitted one after the other, and that the failure cost for function G and function F must be prioritized to the lesser value.

Table 1. Symbolic execution table for *Function N*

Symbolic execution table								
Call Trees 1	A	B	D	G	L			
Call Trees 2	A	B	D	F	H	I	K	L
Call Trees 3	-	-	-	-	-	-	-	-
Call Trees 4	-	-	-	-	-	-	-	-

Table 2. Reservation station for software streaming

Reservation stations				
Function	Busy	Size	Fall-Back	Stage
B	Yes	N-kb	D	2
D	No	M-kb	G (Request)	3
G or F	-	-	-	-

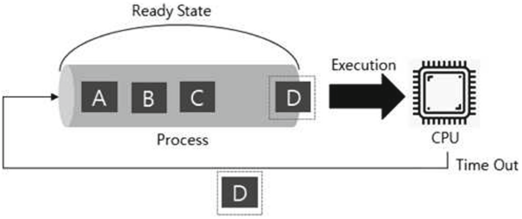


Fig. 6. Round Robin process scheduling

3.3 Network Scheduling for Software Streaming

When divided software is transmitted to the client through the network channel according to the client's request, multi-channel can be used instead of the single channel to reduce network overhead and delay. In this case, efficient scheduling is applied to the network channel to facilitate interactions between the main channel and the sub-channel, thereby enabling real-time software streaming between the server and client. Our network scheduling scheme for software streaming was based on the round-robin process scheduling scheme. The round-robin scheduling method executes all processes running at regular time intervals regardless of the termination of the process. However, its disadvantage is that context exchange occur frequently and overhead also occur frequently if the time interval is short. This because the Round Robin method uses a single queue. We proposed a multi-channel implementation for real-time software streaming between a server and

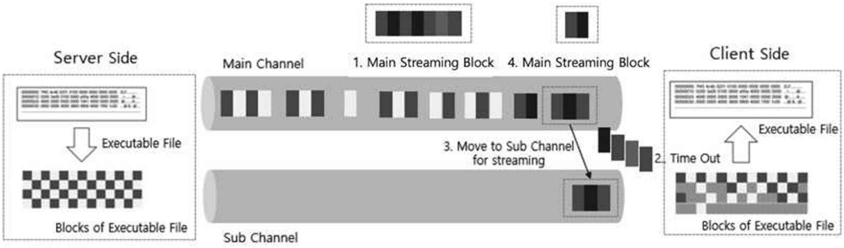


Fig. 7. Network scheduling scheme in multi-channel based on Round Robin process scheduling

a client. In addition, we proposed a scheduling scheme in multi-channel to overcome the disadvantages of the round-robin method. The round-robin scheduling method is presented in Fig. 6.

If the server sends the software block requested by the client in real-time, the client should receive and execute the block. The requested software blocks are sent at regular time intervals, like the process of round-robin process scheduling. However, unlike the existing conventional round-robin method, if a block is not processed for a certain time, the transmission of the block is not interrupted, and it is not sent as the last operation. We proposed a method of moving the a block in progress with a time delay to a Sub Channel to allow for it continued execution and for the next requested block to be executed using the Main Channel. If the scheduling method is used, streaming can be performed without moving the blocks to the sub-channel even if the receiving time of the function requested from the client increases. In addition, the main channel can stream the next block to minimize the overhead and delay of work. The process of applying network scheduling to multi-channel software streaming is presented in Fig. 7.

4 Implementation

In this paper, we propose a multi-channel scheme to minimize application suspension that can occur in software streaming. As mentioned above, existing works that transmit software in a streaming manner generally perform software streaming in units of functions and classes. In this case, application suspension increases when a software block containing a function with a high call depth is transmitted. In order to minimize this application suspension, we present a proof of concept of network scheduling for effective software streaming in the manner specified in Algorithm 1.

Algorithm 1: pseudo code for scheduling of software streaming

```
while client_request do
    main_channel ← request_block;
    if block_receive_time > set_time then
        sub_channel ← request_block;
        main_channel ← next_request_block;
    end
end
```

When a client requests software blocks, the server transmits the requested block through a main channel that uses a relatively high network bandwidth at a set time interval. When the transmission time specified in the main channel expires, the server transmits the software block through a sub channel that uses a relatively low network bandwidth. If there is a new block request from the client, the block is assigned to the main channel. This is because the scheme we propose gives the highest possible transmission priority for the latest request from the client. If there is no new request from the client, the server transmits the block allocated to the sub channel for transmission through the main channel.

5 Future Works

We proposed Disposable Computing using software streaming technology to protect important data in the Real-time system. However, research should be conducted on how to safely destroy internal data in cases of system intrusion in order to minimize internal data leaks cases where all the contents of the real-time system's memory can be recorded. Therefore, we would like to study the Rapid Self-Destruction method for secure data protection of future software streaming systems. A summary of some related research studies and techniques are presented in Table 3 [20].

The Rapid Self-Destruction method presented in this paper is an electronic destruction technology. In cases where multiple devices operating in the central system are compromised the power supplied by the device itself through power supply interruption (Switch off) serves a deletion order and it is issued after the central system recognizes it. It is designed to allow for Rapid Self-Destruction by not storing anything in the volatile memory.

Table 3. Self-destruction Methods and Classification

Type	Self-destruction
Electronic destruction	Power supply interruption (Switch Off)
	Method and apparatus for fast self-destruction of a CMOS integrated circuit [16]
Device fragmentation	Directed fragmentation for unmanned airborne vehicles [17]
Integrating chemical substances	From chips to dust: The MEMS shatter secure chip [18]
	Simulation research on a novel micro-fluidic self-destruct device for microchips [19]

6 Conclusion

The military is one of the areas that has been impacted by technological development of IoT's. Military-purpose reconnaissance drones or intelligence-gathering devices mostly operate using built-in computing systems to conduct their missions. However, there have been various cases in which the sensitive data built into these devices has been leaked and used. Therefore, we proposed a framework that can be applied even for the real-time systems, that which requires real-time execution among IoT devices, in order to prevent the leaking or loss of sensitive data through the use of on-demand computing technology. However, due to the high network overhead of the existing software, real-time execution techniques were appropriate for systems that were less impacted by the delay in running the software and there are challenges in applying them to systems that need to be ensured in real-time. We proposed efficient software streaming in multi-channel using three methods, i.e., symbolic execution, Tomasulo algorithm, and Round Robin. We created a call tree table for a function that was streamed by symbolic execution. Reservation stations and register renaming in Tomasulo algorithm, were used to refer to the Call Tree, and the function blocks were efficiently sent to the client through the Main Channel. The method used a scheduling technique that transferred a function block to a network channel and moved it from the main channel to the sub-channel over time by Round Robin. In our future work, we intend to introduce the concept of Disposable Computing for self-destruction capability.

References

1. Iraq-RQ-170 Homepage. https://www.wired.com/2011/12/iran-drone-hack-gps/. Accessed 22 Jan 2020
2. Iran-RQ-170 Homepage. https://www.csmonitor.com/World/Middle-East/2011/1215/Exclusive-Iran-hijacked-US-drone-says-Iranian-engineer. Accessed 22 Jan 2020
3. Banescu, S., Collberg, C., Ganesh, V., Newsham, Z., Pretschner, A.: Code obfuscation against symbolic execution attacks. In: Proceedings of the 32nd Annual Conference on Computer Security Applications, pp. 189–200 (2016)
4. Fang, H., Wu, Y., Wang, S., Huang, Y.: Multi-stage binary code obfuscation using improved virtual machine. In: Lai, X., Zhou, J., Li, H. (eds.) ISC 2011. LNCS, vol. 7001, pp. 168–181. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-24861-0_12
5. Xue, C., et al.: Exploiting code diversity to enhance code virtualization protection. In: 2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS), pp. 620–627. IEEE (2018)
6. VMProtect Homepage. https://vmpsoft.com/. Accessed 22 Jan 2020
7. Themida Homepage. https://www.oreans.com/themida.php. Accessed 22 Jan 2020
8. Kim, M.-J., et al.: Design and performance evaluation of binary code packing for protecting embedded software against reverse engineering. In: 2010 13th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing, pp. 80–86. IEEE (2010)
9. Suk, J.H., Lee, J.Y., Jin, H., Kim, I.S., Lee, D.H.: UnThemida: commercial obfuscation technique analysis with a fully obfuscated program. Softw. Pract. Exp. **48**(12), 2331–2349 (2018)
10. Yadegari, B., Johannesmeyer, B., Whitely, B., Debray, S.: A generic approach to automatic deobfuscation of executable code. In: 2015 IEEE Symposium on Security and Privacy, pp. 674–691. IEEE (2015)

11. Hasan, S., Awais, M., Shah, M.A.: Full disk encryption: a comparison on data management attributes. In: Proceedings of the 2nd International Conference on Information System and Data Mining, pp. 39–43 (2018)
12. Kuacharoen, P., Mooney, V.J., Madisetti, V.K.: Software streaming via block streaming. In: Jerraya, A.A., Yoo, S., Verkest, D., Wehn, N. (eds.) Embedded Software for SoC, pp. 435–448. Springer, Boston (2003). https://doi.org/10.1007/0-306-48709-8_32
13. Kuacharoen, P., Mooney III, V.J., Madisetti, V.K.: Efficient execution of large applications on portable and wireless clients. In: Proceedings of the Mobility Conference & Exhibition (2004)
14. Choi, J., Kim, J., Jang, B.: A software wireless streaming architecture supporting telematics device. In: 2007 Digest of Technical Papers International Conference on Consumer Electronics, pp. 1–2. IEEE (2007)
15. Tomasulo, R.M.: An efficient algorithm for exploiting multiple arithmetic units. IBM J. Res. Develop. **11**(1), 25–33 (1967)
16. Shield, D.J., Davis, D.L.: Method and apparatus for fast self-destruction of a CMOS integrated circuit. U.S. Patent 5,736,777, issued 7 April 1998
17. Mishra, P.K., Goyal, D.: Directed fragmentation for unmanned airborne vehicles. U.S. Patent 9,828,097, issued 28 November 2017
18. Banerjee, N., Xie, Y., Rahman, M.M., Kim, H., Mastrangelo, C.H.: From chips to dust: the MEMS shatter secure chip. In: 2014 IEEE 27th International Conference on Micro Electro Mechanical Systems (MEMS), pp. 1123–1126. IEEE (2014)
19. Gu, X., Lou, W., Song, R., Zhao, Y., Zhang, L.: Simulation research on a novel micro-fluidic self-destruct device for microchips. In: 2010 IEEE 5th International Conference on Nano/Micro Engineered and Molecular Systems, pp. 375–378. IEEE (2010)
20. Kim, S., Youn, T.-Y., Choi, D., Park, K.-W.: UAV-undertaker: securely verifiable remote erasure scheme with a countdown-concept for UAV via randomized data synchronization. Wirel. Commun. Mob. Comput. **2019**, 1–11 (2019)