Available Online at www.ijcsmc.com

International Journal of Computer Science and Mobile Computing

A Monthly Journal of Computer Science and Information Technology

ISSN 2320–088X
IMPACT FACTOR: 7.056

IJCSMC, Vol. 10, Issue. 2, February 2021, pg.39 – 44

Obfuscation Techniques for Magecart Detection and Prevention

Alex Mathew
Department of Cybersecurity, Bethany College, USA
amathew@bethanywv.edu
DOI: 10.47760/ijcsmc.2021.v10i02.005

Abstract— The risk of an unimaginable degree of illegal and duplicating software reproduction has recently increased due to rapid growth in digital technology. The latter has also contributed to a proportional increase in the piracy rate. This progression has placed a greater danger for software designers and prompted the emergence of multiple techniques of protecting software. Many techniques for protecting software have recently been created, and one such remarkable technique is code obfuscation. This strategy is defined as the methods for hiding data structures, primary algorithms, and code logic and protects the code from intruders. Typically, code obfuscation entails hiding the adoption of code information from a competitor, such as changing the plan in a semantically systematic program, which is not easy to apprehend for an intruder. Therefore, this paper analyzes obfuscation techniques for magecart detection and prevention.
Keywords— Magecart, intruder, Code obfuscation, obfuscation techniques, software susceptibility, and JavaScript

I. INTRODUCTION

Magecart is one of the largest perpetrators of cyber-attacks during the holiday season. It is also a syndicate of criminals targeting e-commerce sites to steal clients' information using their credit cards. The risk from Magecart is immense to the extent that it has attracted the attention of the FBI, making them issue a stern warning to America's private sector concerning these cyber-attacks (Sengupta and Roy 2017). Reportedly, operatives of mega cart inject suspicious JavaScript that steals information from digital payment platforms, generally on their checkout pages, through a process described as form jacking (Chandan and Lalitha 2016). Historically, mega cart codes have been inserted on many sites and consist of multiple users' payment data. These codes gain access to sites through supply chain strikes that target third parties that offer functionality to the websites. Supply chain stacks are responsible for the biggest spikes in detections of mega carts.

Megacart strikes are problematic to detect since they are injurious script residing on the site's customer-facing side. They reside on the client-facing side, waiting to collect any data when a client is at WorldPay. The payment card data is harvested when the website is infected without the consumer or merchant is aware that the data has been affected or compromised (Hosseinzadeh et al., 2018). Business requires a constant emphasis on visibility in this increased attack surface and expanded scrutiny of third-part services utilized in their website applications. Investments in controlling Megacart attacks have recently been reported to be falling short.

I.T. industries are spending billions of dollars yearly to mitigate security strikes such as malicious and tampering reverse mechanization. Because of the vast development and use of multimedia and internet technologies, the greater necessity for the study of security and protection has been developed (Sengupta et al.,

2017). While every company has its intellectual property, most of them find this task challenging since protecting data is not easy, injection of injurious code, or software piracy (Nou et al., 2017). Data protection is a challenging task since most companies keep their information confidential, which may damage the purchaser's software.

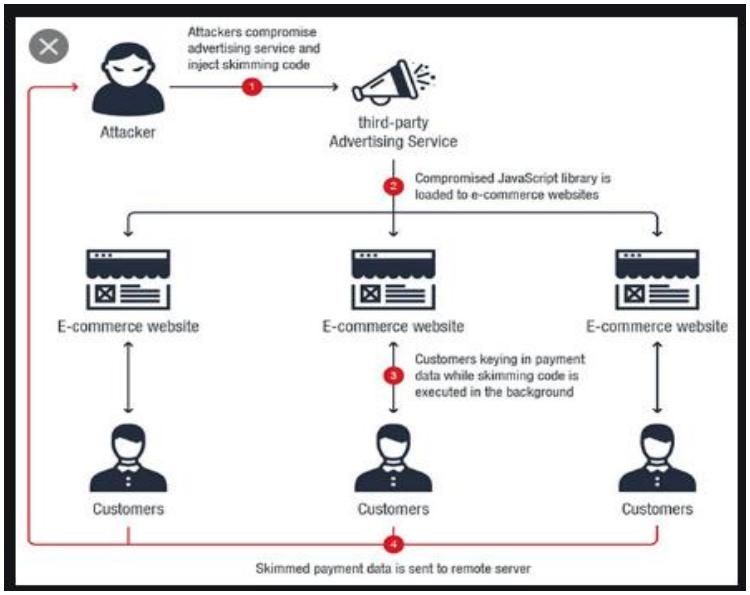
Intellectual property can be protected either technically or legally. The latter implies singing lawful contracts or copyrights against creating duplicates (Wekerle, Pfouga, Stjepandic, and Mai 2018). Technically, on the other hand, it means that software developers will offer the resolution for protecting their software. Notably, securing data implies the use of gateways and firewalls in the operating system. However, a good concept for protection from outsiders is to use these methods and mechanisms in the application software (Hashemzade and Maroosi 2018). Obfuscation is one of those methods. Therefore, it is considered a novel realm of study in software securing and gaining relevancy in the current digital realm.

Obfuscation involves code transformation that makes the plan problematic to grasp by adjusting its model while keeping its primary functionalities. Firewalls and encryption are common resolutions to reduce the risk of intruders who try to split the application (Hashemzade and Maroosi, 2018). However, these methodologies do not assist in protecting the software when the striker or an intruder is a customer. From the multiple techniques accessible for safeguarding the code from various strikes, code obfuscation is common method for safeguarding from code conception and code infringement (Kavitha and Subramaniam 2017). Code obfuscation is a widely used method, and multiple obfuscation strategies have been suggested. Code obfuscation is a software protection technique against unapproved reverse engineering.

Obfuscation methods are used together with other methods such as protection updating, code tampering detection, and code replacement. Practically, watermarking, tamper-proofing, different signed native codes, hardware-based security solutions, protection by server-side, encryption, and software aging are commonly used techniques to challenge or avoid the detection engines (Zhan et al., 2019). However, the provider of these services should estimate the degree of resistance from obfuscation (times of understanding the code).

Obfuscation mechanisms can be adopted on the primary code. This prevents some adversaries from getting the code's logic or algorithm (Behera and Bhaskari, 2017). Additionally, obfuscation techniques include code transformation and re-planning to substitute effective identifier identification in the primary code with identifier renaming, the meshing of control flows, suppression of constants, creation of middle-level code, string encoding, merge local integers, random dead code, variable reassigning, transparent branch insertion, conditional jumps, unconditional jumps, and junk code insertions (Khan et al., 2020). Fundamentally, obfuscation differs from data encryption in multiple ways. This does not primarily require inverse transformation. Attackers cannot access the software without using the original code.

II. PROPOSED METHODOLOGY BLOCK DIAGRAM & FLOW CHART



III. RESULT ANALYSIS

This section shows the results of code obfuscation.

Table 1: Program 1

Program-1 (Original Code)	Program- 1 (Obfuscated Code)
.model small .code Mov DH, 65 Mov AH, 2 *Mov DL, 75* Int 21h Mov AH, 4CH Int 21h end	.model small .code Mov DH, 65 Mov AH, 2 *db 10110010 b* *Dec BX* Int 21h Mov AH, 4CH Int 21h end

Source: (Chandan and Lalitha 2016)

'Movil' represents machine code in obfuscated code. 75 dB 01001011 is the binary form of the number. To decrement the register content, the instruction is marked as '01001.reg and the registry B.X. represents '011'. Here, the binary number is substituted in the code as 'Dec BX.'

Table 2: Program 2

Program-2 (Original Code)	Program-2 (Obfuscated Code)
.model small .code Mov AH, 2 Mov DL, 65 Mov BL, 70 L1: Int 21h Add DL, 1 *Cmp DL, BL* *JLE L1* Mov AH, 76 Int 21h end	.model small .code Mov AH, 2 Mov DL, 65 L1: Int 21h Add DL, 1 Mov BL, DL *Add BL, 185* *JNC L1* Mov AH, 76 Int 21h end

Source: (Chandan and Lalitha 2016)

JNC (jump with no carry) is used as JLE. JNC, however, transfers the control when the carry flag is open to the level. It is managed to the boundary between -128 to +127. The table shows that JBC and JLE programs give similar results.

Table 3: Program 3

Program-3 (Original Code)	Program-3 (Obfuscated Code)
.model small .code Lea SI, L1 Mov AH, 2 Mov BL, 70 Mov BH, 71 Mov CL, 72 Mov CH, 73 Mov DH, 74 L1: *Mov DL, CH* Int 33 Mov AH, 76 Int 33 end	.model small .code Lea SI, L1 Mov AH,2 Mov BL,70 *Add CS:[SI+1], AH* Mov BH, 71 Mov CL, 72 Mov CH, 73 Mov DH, 74 L1: *Mov DL, BL* Int 33 Mov AH, 76 Int 33 end

Source: (Chandan and Lalitha 2016)

The information in DL, the registry of C.H., is moved from the primary program resulting in 'I' output. The directive the value of B.L., B.L.' 71, 'Mov DL, in the obfuscated code is also moved to DL. However, due to 'Add C.S; [I +SI], A.H and A.H.'s information would be included in B.L. Therefore, 72 would be kept in the B.L. However, the instruction moved to the subsequent directive. For instance, C.H. shifts to DL, and the result of 'I' moves to H, as shown in the table above.

Table 4: Program 4

Program-4 (Original Code)	Program-4 (Obfuscated Code)
.model small .code Mov AH, 2 *Mov AL, 180* Mov DL, AL Int 33 *Mov BH, 186* Mov DL, BH Mov AH, 2 Int 33 Mov AH,76 Int 33 end	.model small .code *Mov AX, 47796* Mov CH, AH Mov AH, 2 Mov DL, AL Int 33 Mov DL, CH Mov AH, 2 Int 33 Mov AH, 76 Int 33 end

Source: (Chandan and Lalitha 2016)

The table above shows the contents of the original code of B.H. and A.L. This table indicates those values of 186 and 180 codes, respectively. Consequently, in this obfuscated program, the content B.H. is not utilized. Instead, A.H. content is used, and keeping the information of C.H. B.H. and A.L. contents in the primary plan is kept in the registry of A.X. of obfuscated methods. Values of 47796 are computed by 186 X 256 + 180 or information of (AH) X 28 + information (AL). The content B.H. in the primary code is used instead of A.H.

Table 5: Program 5

Program-5 (Original Code)	Program-5 (Obfuscated Code)
.model small .code Mov DH, 85 Mov AH, 2 Mov CL, 80 *Mov DX, 35537* Int 21h Mov DL, DH Int 21h Mov AH, 76 Int 21h end	.model small .code Mov DH, 85 Mov AH, 2 Mov CL, 80 db 10111010b Mov DL, CL Int 21h Mov DL, DH Int 21h Mov AH, 76 Int 21h end

Source: (Chandan and Lalitha 2016)

The results of the table above show that system codes and numbers are interchanged in assembly codes. The system code '10111010' shown in the obfuscated code is presented as 1011-1-010'. 'Move DX' is presented as '1011wreg number'. Similarly, this format can be changed in the assembly code. Accordingly, 10001010010001 in binary form are shown as 35537 and are read as 10010-10-11 in 'Mov-dw-11-DL-CL'such as 'Mov DL, and CL.'

Table 6: Program 6

Program-6 (Original Code)	Program-6 (Obfuscated Code)
.model small .code Mov BL, 80 Mov BH, 85 Mov AH, 2 *Mov BL, 178* *Mov BH, 98* Mov DL, BH Int 21h Mov DL, BL Int 21h Mov AH, 76 Int 21h end	.model small .code Mov BL, 80 Mov BH, 85 Mov AH, 2 *db 10110011b* *Mov DL, 183* *db 98d* Mov DL, BH Int 21h Mov DL, BL Int 21h Mov AH, 76 Int 21h end

Source: (Chandan and Lalitha 2016)

In the primary program, 'Mov BL, 178' instruction is shown as binary code 10110011 and shown in the obfuscated code. 1010010 binary model of 178 is explained as is 'Mov DL' and '1011- 0-010'. This shows binary and assembly code instruction with a decimal number of the obfuscated program common to the original program.

Table 7: Program 7

Program-7 (Original Code)	Program-7 (Obfuscated Code)
.model small .code Mov AH, 2 *Mov DL,* 70 Int 33 Mov AH, 76 Int 33 end	.model small .code Mov AH, 2 *db 178d* *db 70d* Int 33 Mov AH, 76 Int 33 end

Program-8 (Original Code)	Program-8 (Obfuscated Code)
.model small .code Mov BL, 70 Mov CL, 80 Mov DH, 90 Mov AH, 2 *Mov DL, CL* Int 21h Mov AH, 76 Int 21h end	.model small .code Mov BL, 70 Mov CL, 80 Mov DH, 90 Mov AH, 2 *db 10001010b* *db 11010001b* Int 21h Mov AH, 76 Int 21h end

Source: (Chandan and Lalitha 2016)

The table above shows that the primary code of the instruction CL, Mov DL, the information of the registry shifts to another one. The system code '100010dw 11reg1, reg2' accounts for 100010, w=0 (for byte) and d=1(for correct). Therefore, the system code 1001010 is produced in the obfuscated code. Another section of the binary is '1.1reg1, & reg 2'. The CL and Dl, for instance, are used. The system codes for CL and DL are 001 and 010.

Table 6: Program 8

Program-9 (Original Code)	Program-9 (Obfuscated Code)
.model small .code Mov BL, 70 Mov CL, 80 Mov DH, 90 Mov AH, 2 *Mov DL, 178* Int 21h Mov AH, 76 Int 21h end	.model small .code Mov BL, 70 Mov CL, 80 Mov DH, 90 Mov AH, 2 *db 10110010b* *db 10110010b* Int 21h Mov AH, 76 Int 21h end

Source: (Chandan and Lalitha 2016)

The table above shows that the original binary directive where the first four digits represent 'Mov' and the other 4 bits shows the DL registry. As shown in the table, another binary directive indicated the numerical value 11 x16 + 2 (178). While the based are software-based, they increase the difficulty of reverse-engineering for Magecart detection and prevention via matching pattern, particularly when these methods would be integrated with the present code of obfuscation approaches.

IV. CONCLUSIONS

The latest rate of Magecart strikes is placing a strong focus on software susceptibility, incorrect configuration, and defense loopholes. This gives attackers room for maliciously accessing the company's software. Although current best-practices consistently remain to be significant, there are some security methods that a company can implement to address the unique strike vectors by Magecart. This incidence increases the effort needed to prevent attacks before they start. They also detect unapproved modification of the codes, relate application events in wider security images, and cooperate with other third-party organizations committed to effective security practices.

The increase in software piracy has led to an attempt in discussing and implementing obfuscation techniques. It has been reported that the difficulty of code increases both structurally and reasonably due to the insertion, removal, and arrangement of the code. The obfuscation methods have been reported to be practical in magecart detection and prevention. This involves taking corrective actions to obfuscate the code with less complexity. Finally, there is a need for future research in developing the automation system of obfuscation techniques.

REFERENCES

[1] Chandan K and Lalitha B (2016). *Different obfuscation techniques for code protection*. Procedia Computer Science Volume 70, Pages 757-763.

[2] Hosseinzadeh, S., Rauti, S., Laurén, S., Makela, J.M., Holvitie, J., Hyrynsalmi, S. and Leppänen, V., 2018. Diversification and obfuscation techniques for software security: A systematic literature review. *Information and Software Technology*, *104*, pp.72-93.

[3] Now, L., Rahimian, A., Michel, D., Debbabi, M. and Hanna, A., 2017, May. Binding: fingerprinting binary functions to support automated analysis of code executables. In *IFIP International Conference on ICT Systems Security and Privacy Protection* (pp. 341-355). Springer, Cham.

[4] Hashemzade, B. and Maroosi, A., 2018. Hybrid obfuscation using signals and encryption. *Journal of Computer Networks and Communications*, *2018*.

[5] Kavitha, D. and Subramaniam, C., 2017. Security threat management by software obfuscation for privacy on the internet of medical thing (IoMT) application. *Journal of Computational and Theoretical Nanoscience*, *14*(7), pp.3100-3114.

[6] Zhan, X., Zhang, T. and Tang, Y., 2019, February. A comparative study of android repackaged apps detection techniques. In *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)* (pp. 321-331). IEEE.

[7] Behera, C.K. and Bhaskari, D.L., 2017. Self-Modifying Code: A Provable Technique for Enhancing Program Obfuscation. *International Journal of Secure Software Engineering (IJSSE)*, *8*(3), pp.24-41.

[8] Khan, F., Ncube, C., Ramasamy, L.K., Kadry, S., and Nam, Y., 2020. A digital DNA sequencing engine for ransomware detection using machine learning. *IEEE Access*, *8*, pp.119710-119719.

[9] Wekerle, T., Pfouga, A., Stjepandic, J., and Mai, P., 2018. Intellectual property protection in smart systems engineering on an exchange of simulation models. *Advances in transdisciplinary engineering*, *7*, pp.198-207.

[10] Hashemzade, B. and Maroosi, A., 2018. Hybrid obfuscation using signals and encryption. *Journal of Computer Networks and Communications*, *2018*.

[11] Sengupta, A, Roy, D and Corcoran, P (2017). "DSP Design Protection in C.E. through Algorithmic Transformation Based Structural Obfuscation," IEEE Transactions on Consumer Electronics, Volume 63, Issue 4, November, pp: 467 – 476.

[12] Sengupta, A and Roy, D (2017). "Protecting an Intellectual Property Core during Architectural Synthesis using High-Level Transformation Based Obfuscation" IET Electronics Letters, Volume: 53, Issue: 13, pp. 849 – 851.

