Poster: Obfuscating Program Control Flow with Intel SGX

Yongzhi Wang, Yulong Shen, Ke Cheng, Yibo Yang, Cuicui Su, Anter Faree
School of Computer Science and Technology, Xidian University
Xian, Shaanxi
yzwang@xidian.edu.cn,ylshen@mail.xidian.edu.cn,kechengstu@gmail.com,bobyangpopo@gmail.com
ccsu.hannah@gmail.com,dafir.net@gmail.com

ABSTRACT

Control flow obfuscation is a direct approach in protecting the confidentiality of program logic. However, existing works in this direction either failed to offer high confidentiality guarantees or incurred high performance overheads. In this paper, we propose CFHider, a high security and high performance control flow obfuscation technique. By leveraging program transformation and Intel Software Guard Extension (SGX) technology, CFHider hides control flow information to an opaque yet trusted execution environment, i.e., the SGX enclave. Our evaluation showed that, CFHider extensively raises the bar for reverse-engineering attacks targeting on the control flow confidentiality, and incurs a moderate performance overhead.

CCS CONCEPTS

• **Security and privacy → Software security engineering**;

KEYWORDS

Control flow confidentiality, SGX, Cloud computing

ACM Reference Format:
Yongzhi Wang, Yulong Shen, Ke Cheng, Yibo Yang, Cuicui Su, Anter Faree. 2018. Poster: Obfuscating Program Control Flow with Intel SGX. In *ICSE '18 Companion: 40th International Conference on Software Engineering Companion, May 27-June 3, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3183440.3194990

1 INTRODUCTION

As remote computation paradigms, such as public cloud computing, is gaining popularity, protecting the confidentiality of remote program's logic has drawn increasing attentions. Control flow obfuscation, a method transforming control flows into unintelligible forms, can directly protect program logic confidentiality. However, existing works in achieving this goal still have limitations in the aspect of security [4] [3] and expressiveness [2].

In this paper, we combine program transformation and Intel *Software Guard Extension* (*SGX*) [1] technology, and propose *CFHider*, a hardware-assisted control flow obfuscation solution. CFHider protects the condition of branch statements by moving them into the *CF Enclave*, an opaque and trusted SGX enclave. In addition,

ICSE '18 Companion, May 27-June 3, 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5663-3/18/05.
https://doi.org/10.1145/3183440.3194990

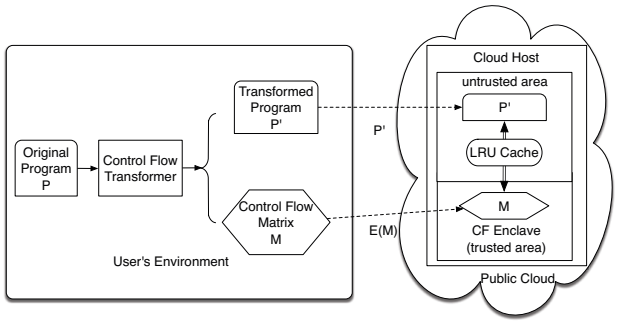

Figure 1: The architecture of CFHider

CFHider inserts *fake branch statements* into the program to further obfuscate the control flow. As a result, CFHider extensively raises the bar of the reverse-engineering attacks. To the best of our knowledge, CFHider is the first solution that leverages SGX technology to protect control flow confidentiality, which achieves a high confidentiality guarantee and a low performance overhead. Moreover, since CFHider only targets on branch statements, which is the essential component for most programming languages, it makes CFHider a general solution to most application-level programs. We implemented CFHider as a program transformation tool, which automatically transforms Java programs into the CFHider-compatible format. Our experimental results showed that, when all branch statement conditions are protected, the average performance overhead incurred by CFHider is 14.26%.

2 SYSTEM DESIGN

2.1 Overview

CFHider can be used in a remote computing setting, which consists of a local environment and a remote environment. In this architecture, we assume that the local environment is secure and trusted. The attacker cannot compromise it. We assume the remote environment is untrusted. However, we assume the processors (i.e., the CPU) on the remote environment support SGX technology and are trusted. Yet, the software stacks on the remote environment are untrusted.

Fig. 1 shows an instance of such an architecture. The architecture consists of the user's environment (i.e., the local environment) and the public cloud (i.e., the remote environment). One or multiple cloud hosts are deployed on the public cloud. The processor of each host supports Intel SGX technology.

CFHider is performed in the following sequence. Suppose a program P (called the *original program*) needs to be executed on the public cloud, CFHider will first transform it into a *transformed program* P' and a *control flow matrix* M in the user's environment. P' differs from P in that more branch statements are inserted to

obfuscate the control flow graph and the condition logic of each branch statement is moved to M. After the transformation, P' will be uploaded to the cloud host to execute. Meanwhile, M will be sent to a SGX enclave, called *CF Enclave*, on the cloud host. During the execution, P' will be executed in the untrusted area. The conditions in each branch statement of P' will be evaluated in the CF Enclave based on items in M. In other words, the untrusted area and the CF Enclave will work together in order to complete the computation.

Notice that the control flow matrix M is transmitted and stored in the public cloud in an encrypted format, marked as $E(M)$. $E(M)$ can only be decrypted in the CF Enclave, when CF Enclave testifies its integrity through SGX remote attestation protocol [1]. Therefore, CFHider will not directly leak the condition information to the attacker. In the following sub-section, we will specify the details of the program transformation.

2.2 Program Transformation

The program transformation is performed on the *3-address code*, an intermediate representation of the program. In the 3-address code, a branch statement will be in the following format, where x *op* y is the *condition* of the branch statement. X is the *target statement* that the control flow will jump to.

$$\text{if } (x \ op \ y) \text{ then } \{\text{goto X}\} \tag{1}$$

In the condition, x and y are two variables in the program, *op* is the comparison operator that can be one of the following six values: $\{>, <, ==, !=, >=, <=\}$.

During the transformation, CFHider first assigns a unique identifier to each branch statement s, marked as I_s. For the condition of each branch statement, x *op* y, we generate a *variable list* L_s and a tuple T_s. The list L_s consists of x, y, and other variables appeared before the execution of s. The order of elements in L_s is randomly shuffled. The positions of x and y in L_s are marked as i_x and i_y and will be stored in tuple T_s. In addition to i_x and i_y, the statement identifier I_s and the operator *op* are also stored in tuple T. Based on L_s and T_s, we replace the condition of branch statement s with an invocation of *CFQ function*, `cfQuery`, using L_s and I_s as parameters.

Fig. 2 shows an example of the program transformation. As the figure shows, CFHider replaces the condition of `x>y` with a function call `cfQuery((a,y,b,x), B1)` and stores the tuple $< B1, 3, 1, \text{“} > \text{”} >$ to the control flow matrix. During the execution, the parameter $B1$ helps the CF Enclave to identify the operator “>" and the index of condition variables, i.e., 3 and 1, in the parameter list. As a result, the condition $x > y$ is reassembled and evaluated in the CF Enclave.

To further obfuscate the control flow, CFHider can insert *fake branch statements* before any original statements. The condition of a fake branch statement s is still be an invocation of `cfQuery` function, making it indistinguishable from the transformed original condition. The variable list L_s can be any variables appeared before s. For the inserted s, a tuple T_s is added to the control flow matrix M. The content of T_s is I_s and a fixed boolean value that ensures the correctness of the control flow during the execution. As Fig. 2 shows, the second branch statement in the transformed program is a fake branch statement. During the execution, its corresponding tuple always returns `false`. Executing this statement always leads to *L2*. In consideration of performance, for each original statement, we insert a fake branch statement before it with a certain probability, called *confusion degree*, marked as d.

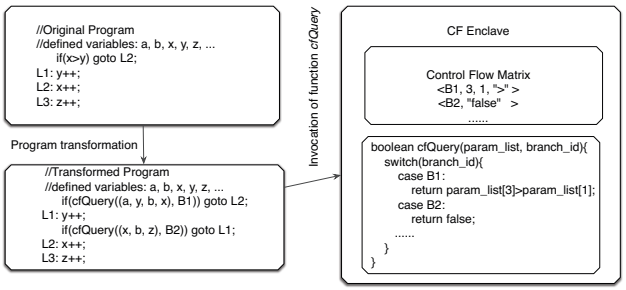

Figure 2: An example of CFHider

3 EVALUATION

We implemented CFHider as a transformation tool with *Soot*, Intel SGX SDK and Java Native Interface (*JNI*). To reduce performance overhead by frequent inquiries on CF Enclave, we introduce a *LRU cache* in the untrusted area (see Fig. 1). The actual inquiry on CF Enclave will happen only when the cache miss appears.

We selected several Hadoop applications (include Word Count, Pi, Tera Sort and Page Rank) to test the applicability and the performance of CFHider, and set the length of the variable list L_s of the CFQ function as 10, the LRU cache size (W) as 10, 000, and the confusion degree (d) as 0%, 30% and 50%, respectively for each application. Experimental results indicate that CFHider can effectively protect the control flow confidentiality of programs and incurs an acceptable performance overhead. For instance, when d was set to 0%, 30% and 50%, the average performance overheads are 14.26%, 32.91% and 46.38%, respectively.

ACKNOWLEDGMENTS

This paper is supported in part by the National Natural Science Foundation of China (grant No. 61602364), Natural Science Foundation of Shaanxi Province (grant No. 2017JM6083), National Key R&D Program of China (grant No. 2017YFB1400700), Natural Science Foundation of China (grant No. 61373173 and U1536202) and Fundamental Research Funds for the Central Universities (grant No. JB180303).

REFERENCES

[1] Victor Costan and Srinivas Devadas. 2016. Intel SGX Explained. *IACR Cryptology ePrint Archive* 2016 (2016), 86.
[2] Monirul I Sharif, Andrea Lanzi, Jonathon T Giffin, and Wenke Lee. 2008. Impeding Malware Analysis Using Conditional Code Obfuscation. In *Network and Distributed System Security Symposium (NDSS)*.
[3] Yongzhi Wang and Jinpeng Wei. 2015. Toward protecting control flow confidentiality in cloud-based computation. *Computers & Security* 52 (2015), 106–127.
[4] Zhi Wang, Jiang Ming, Chunfu Jia, and Debin Gao. 2011. Linear obfuscation to combat symbolic execution. In *European Symposium on Research in Computer Security*. Springer, 210–226.