Poster: Korean Shellcode with ROP Based Decoding

Ji-Hyeon Yoon
Seoul Women's University
jhy@swu.ac.kr

Hae Young Lee
Seoul Women's University
haelee@swu.ac.kr

***Abstract*—Although we can hide shellcode in plain text (e.g., English shellcode), due to the signature of its decoder, it can be detected by defensive measures. In this paper, we present an approach to hide shellcode in Unicode encoded Korean text and to reconstruct it based on return-oriented programming. In our approach, shellcode is hidden within Korean text in the form of Chinese characters. By overwriting return addresses on the stack, control flow is directed through existing instructions, so that shellcode is reconstructed and then executed. Our approach is simple, yet it may be effective against payload inspection as well as last branch recording based defensive measures. With some modifications, we may hide shellcode in East Asian text and reconstruct other plain text encoded shellcode without the use of a decoder.**

I. Introduction

This paper presents another approach to hide and reconstruct shellcode in UTF-16 encoded Korean text on the x86. In our previous study [1], we showed that shellcode could be easily embedded within Korean text in the form of Chinese characters, and reconstructed by a small decoder involving bitwise XOR operations. It was indeed simple, especially compared to English shellcode [2], in terms of encoding and decoding, yet it might be effective against payload inspection. However, due to the signature of the decoder, shellcode embedded in Korean text could be detected by defensive measures.

In order to eliminate a decoder from a payload, our present approach exploits return-oriented programming (ROP) [3] to reconstruct shellcode embedded in Korean text. Computations for decoding shellcode are constructed by chaining short sequences of instructions, each of which ends with a `ret` instruction, in the target program. By overwriting the starting addresses of these sequences and of shellcode on the stack, shellcode is reconstructed, and then finally executed. ROP based decoding may be undetected by last branch recording (LBR) based defensive measures such as kBouncer [4], thanks to its simplicity. Although malicious computation itself may be conducted through 'pure' ROP (e.g., using return-oriented shellcode), it may be very difficult or even impossible to implement.

Note that our approach is not be confined to Korea since Korean text can be considered to be a part of multilingual support. Also, with some modifications of our approach, shellcode may be embedded in Chinese and Japanese text. Moreover, ROP based decoding may be employed to reconstruct plain text encoded shellcode such as English shellcode [2] without the use of a decoder.

NDSS '15, 8-11 February 2015, San Diego, CA, USA
Copyright 2015 Internet Society, ISBN 1-891562-38-X
http://dx.doi.org/10.14722/ndss.2015.23xxx

II. Payload Preparation

As shown in Fig. 1, in our approach, a payload to be overwritten on the stack is comprised of: (a) UTF-16 encoded Korean text that includes Chinese characters obtained from shellcode, (b) data used to launch ROP based decoding and to execute the reconstructed shellcode.

A. Embedding Shellcode in Korean Text

In South Korea, Chinese characters are often used with Korean alphabet, to clarify meaning of Sino-Korean words, which originate from or were influenced by Chinese words. Thus, as in [1], shellcode can be embedded within Korean text in the form of Chinese characters.

First, a Chinese character is obtained from each 2-byte code. While some 2-byte codes (e.g., Fig. 1(c)) will already appear to be Chinese characters, the others will not; but they can be easily transformed into Chinese characters through bitwise XOR operations, as shown in Fig. 1(d). Next, as shown in Fig. 1(e), these characters are grouped into pseudo-Chinese words with the consideration of reconstruction operations (i.e., XOR masks) and Sino-Korean words in Korean text. Finally, each pseudo-Chinese word is placed in the parenthesis followed by a Sino-Korean word within the text, as shown in Fig. 1(f). Alternatively, we can make Korean mixed script; Sino-Korean words are replaced with pseudo-Chinese words.

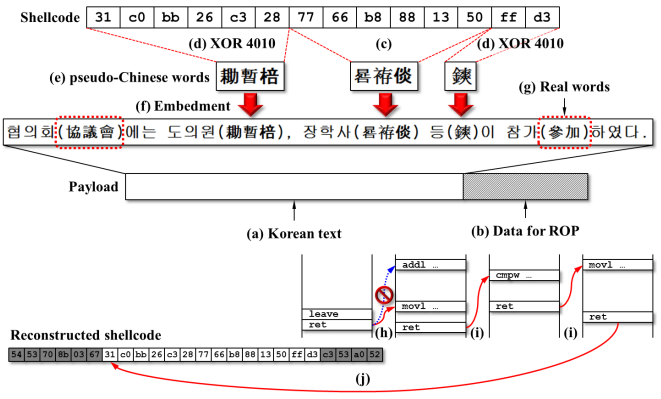

Fig. 1. Overview of Our Approach

These pseudo-Chinese words in Korean text may be undetected by automatic and even manual payload inspection; since many Korean young people are not much familiar with Chinese characters, they may miss *mismatches* between Sino-Korean words and pseudo-Chinese ones. Additionally, as shown in Fig. 1(g), we can place some 'real' Chinese words within text, which may make it difficult to be distinguished from 'real' Korean text.

B. Data for ROP Based Decoding

In our approach, computations to decode shellcode are constructed by chaining 'gadgets,' each of which is a sequence of instructions in the target program, and ends with a `ret` instruction (`c3`). A payload includes the starting addresses of these gadgets and of shellcode to be reconstructed (e.g., addresses pointed by arrows (h), (i), and (j) in Fig. 1) so that they can be overwritten on the stack. Even if the program does not have some 'useful' instructions, we may find them by 'byte-shifting' sequences of instructions due to the x86's variable-length instruction set [3]. Any base pointers and values used in decoding (e.g., XOR masks) can be also included in the payload if needed.

Decoding computations are also simple; each Chinese word in Korean text is retransformed through an XOR operation with a mask hinted by the *very last* Korean character in front of the word (or by the *very next* Korean character in case of Korean mixed script). Every Korean character is comprised of at least two or often three Korean *letters*, each of which can be used as a 'hint.' Any real Chinese words can be ignored in a similar way (i.e., based on hints). These computations may not be much affected by the complexity of shellcode. Therefore, as in [3], to find gadgets for shellcode reconstruction may be fully automated.

III. Shellcode Reconstruction

By using a buffer overflow vulnerability of the target program, a payload that includes Korean text and data for ROP based decoding can be overwritten on the stack. Then, as shown in Fig. 1(h), the first encounter with a `ret` instruction diverts the control flow of the program to the first gadget due to the pop operation of its starting address that has been overwritten through the buffer overflow. As a result of the execution of the gadget chain (Fig. 1(i)), shellcode is reconstructed from the text. Upon encountering a `ret` instruction within the last gadget, the control flow is diverted to the reconstructed shellcode, as shown in Fig. 1(j), so that the shellcode is finally executed.

In our proof-of-concept attacks, shellcode, regardless of its complexity, could be reconstructed with about 30 instructions. Thus, through further optimization, we may make ROP based decoding undetected by LBR based defensive measures.

IV. Conclusions and Future Work

This paper presented *Korean shellcode* with ROP based decoding, in which shellcode is embedded in UTF-16 encoded Korean text and reconstructed with computations constructed by gadget chaining. Our approach can be easy to implement, yet it may be effective against payload inspection and LBR based defensive measures. It may be particular useful when malicious computation to be conducted is highly complex. Our future work includes: (a) automation of our approach, (b) detection of Korean shellcodes, and (c) applications to other languages.

Acknowledgment

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT and Future Planning (NRF-2013R1A1A1006542).

References

[1] J.H. Yoon and H.Y. Lee, "Hiding Shellcodes in Korean Texts," *Proc. of USENIX Security '14*.

[2] J. Mason and S. Small, "English Shellcode," *Proc. of ACM CCS '09*.

[3] H. Shacham, "The Geometry of Innocent Flesh on the Bone: Return-into-libc without Function Calls (on the x86)," *Proc. of ACM CCS '07*.

[4] V. Pappas, M. Polychronakis, and A.D. Keromyt, "Transparent ROP Exploit Mitigation Using Indirect Branch Tracing," *Proc. of USENIX Security '13*.