Korean Shellcode with ROP Based Decoding

SEOUL WOMEN'S UNIVERSITY

Ji-Hyeon Yoon* and **Hae Young Lee**
Department of Information Security
Seoul Women's University
Republic of Korea
{jhy,haelee}@swu.ac.kr

22nd Annual Network and Distributed System SECURITY SYMPOSIUM
Catamaran Resort Hotel & Spa
San Diego, California
February 8-11, 2015

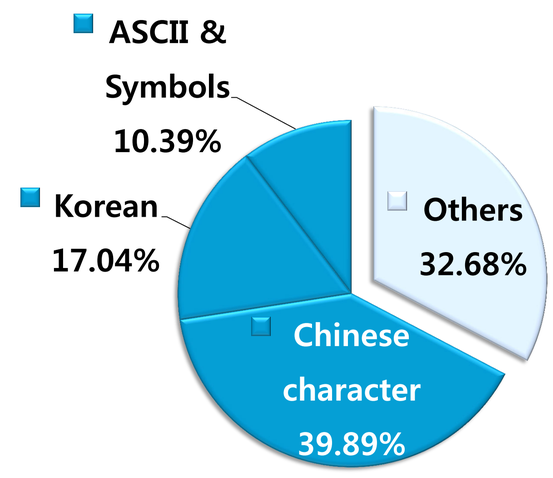
Our Previous Work (Korean Shellcode)

➢ ***Background & Motivation***
- Sino-Korean: About 6~70% words in the Korean vocabulary originated from Chinese words.
- Chinese characters are often used to clarify meaning of Sino-Korean.
- Korean text may include Korean, Chinese, alphanumeric characters, and symbols, which make up a large portion (approximately 70%) of the UTF-16 character set.

➢ ***Basic Idea***
- Each 2-byte code of shellcode is transformed into a Chinese character and then placed within Korean text.
- Many 2-byte codes will already appear to be Chinese characters.
- The others can be transformed into Chinese ones by XOR operations.

➢ ***Our Approach: Hiding shellcode by placing pseudo-Chinese words***
- A simple decoder retransforms these words through XORs hinted by Korean characters.

➢ ***Merits***
- Shellcode can be easily embedded within Korean text and reconstructed by a simple decoder.
- Shellcode hidden in text may not be detected by automatic and even manual payload inspection.
- It could be extended to East Asian languages that use Chinese characters (e.g., Chinese and Japanese).



Portion of Unicode characters

Background & Motivation

➢ ***Demerits in Korean shellcode***
- Shellcode embedded in Korean text could be detected due to 'the signature of its decoder.'

➢ ***Return-Oriented Programming (ROP)***
- A computer security exploit technique that allows an attacker to divert control flow and execute arbitrary code using existing codes – without injecting any code.
- Gadget: several small instruction sequences of existing code used in ROP. Gadgets end with an indirect ret instruction and are chained together through that instruction.
- Demerits: Conducting malicious operations through 'pure' ROP may be very difficult or even impossible to implement if there is no appropriate instructions in the target program.

➢ ***Motivation – How about decoding Korean shellcode based on ROP?***
- The signature of Korean shellcode may be virtually eliminated if we can reconstruct it using ROP.
- It would be easier to implement than 'pure' ROP; we just need to find appropriate instructions for the reconstruction.

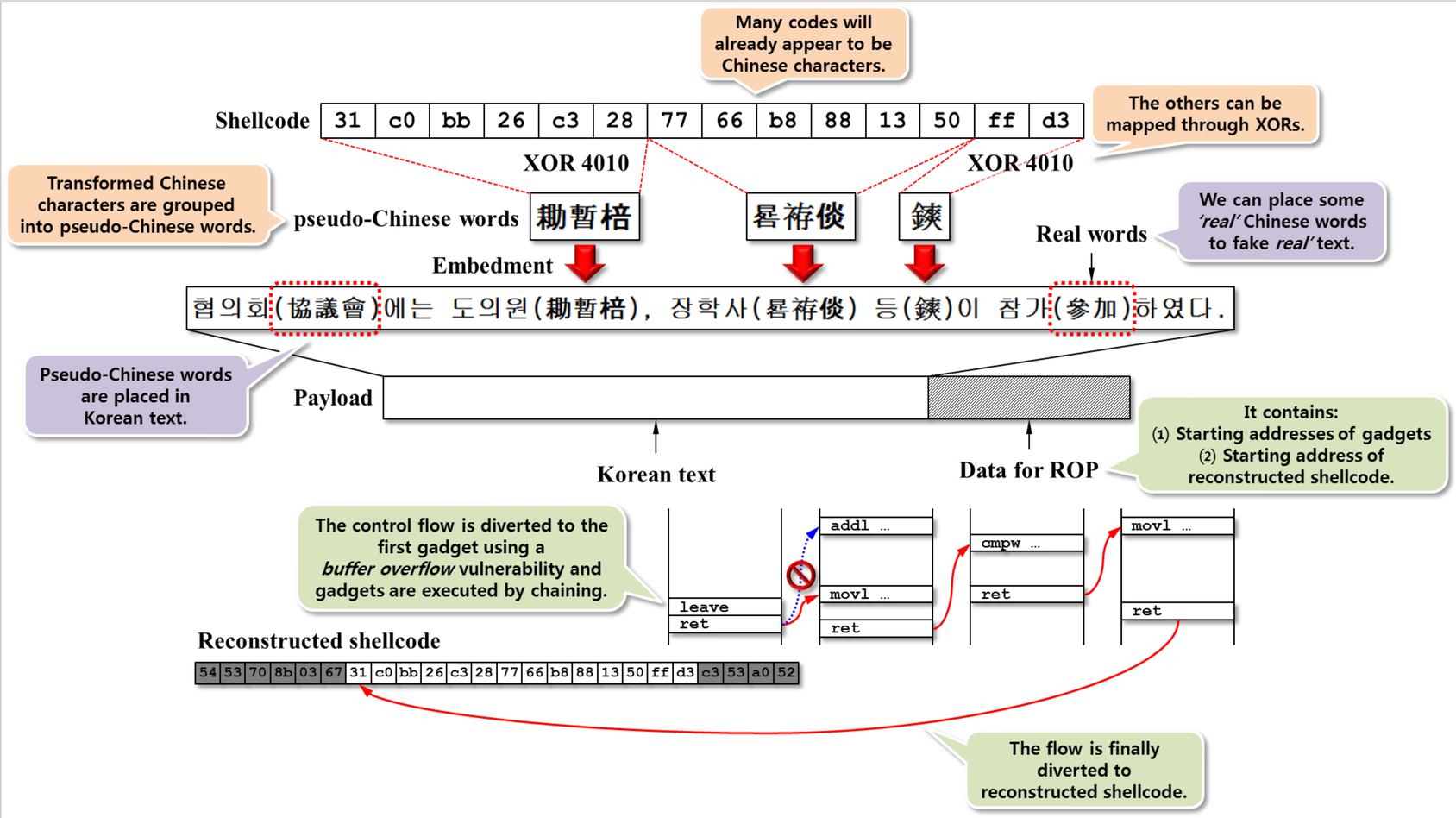
Conclusions & Future Work

➢ ***Korean shellcode with ROP based decoding:***
- Shellcode can be hidden in Korean text and reconstructed by ROP based gadgets.
- May evade many detection techniques thanks to the elimination of the signature.
- Easy to be implemented, yet effective against payload inspection and LBR based defensive measures.
- Can be applied to other East Asian languages such as Chinese and Japanese.

➢ ***The future work includes:***
- Automation of our approach
- Detection of Korean shellcode
- Applications to other languages

NRF DR. LEE'S LAB

Our Present Approach



1) Hiding Shellcode in Korean Text
- Some 2-byte codes already will appear to be Chinese characters and the others can be easily transformed into Chinese characters through XORs.
- These Chinese characters are grouped into pseudo-Chinese words based on reconstruction operations (XOR masks in the figure).
- Each pseudo-Chinese word is placed within text.
- Some 'real' Chinese words can be placed to make text difficult to be distinguished from 'real' text.

2) Data for ROP Based Decoding
- Korean shellcode is reconstructed through chaining 'gadgets.'
- Gadgets are consist of instructions existing in the target program and end with a ret instruction.
- A payload contains: ① Korean shellcode, ② starting addresses of gadgets, and ③ starting address of reconstructed shellcode.
- Each Chinese word is retransformed through an XOR with a 'hint' in text.
- Any real Chinese words can be ignored based on hints.

3) Shellcode Reconstruction
① **Injecting Korean shellcode**
Through the buffer overflow vulnerability of the target program, the stack is overwritten by a payload that includes Korean shellcode and data for ROP based decoding.

② **Diverting Control Flow**
The first encounter with a ret instruction diverts the control flow of the program to the first gadget.

③ **Reconstructing Shellcode by Gadget Chaining**
The other gadgets are executed by gadget chaining, so that shellcode is reconstructed.

④ **Executing Reconstructed Shellcode**
The encounter with a ret instruction within the last gadget diverts the flow to the reconstructed shellcode, so that it is finally executed.