CSE 610 Special Topics: System Security - Attack and Defense for Binaries

Instructor: Dr. Ziming Zhao

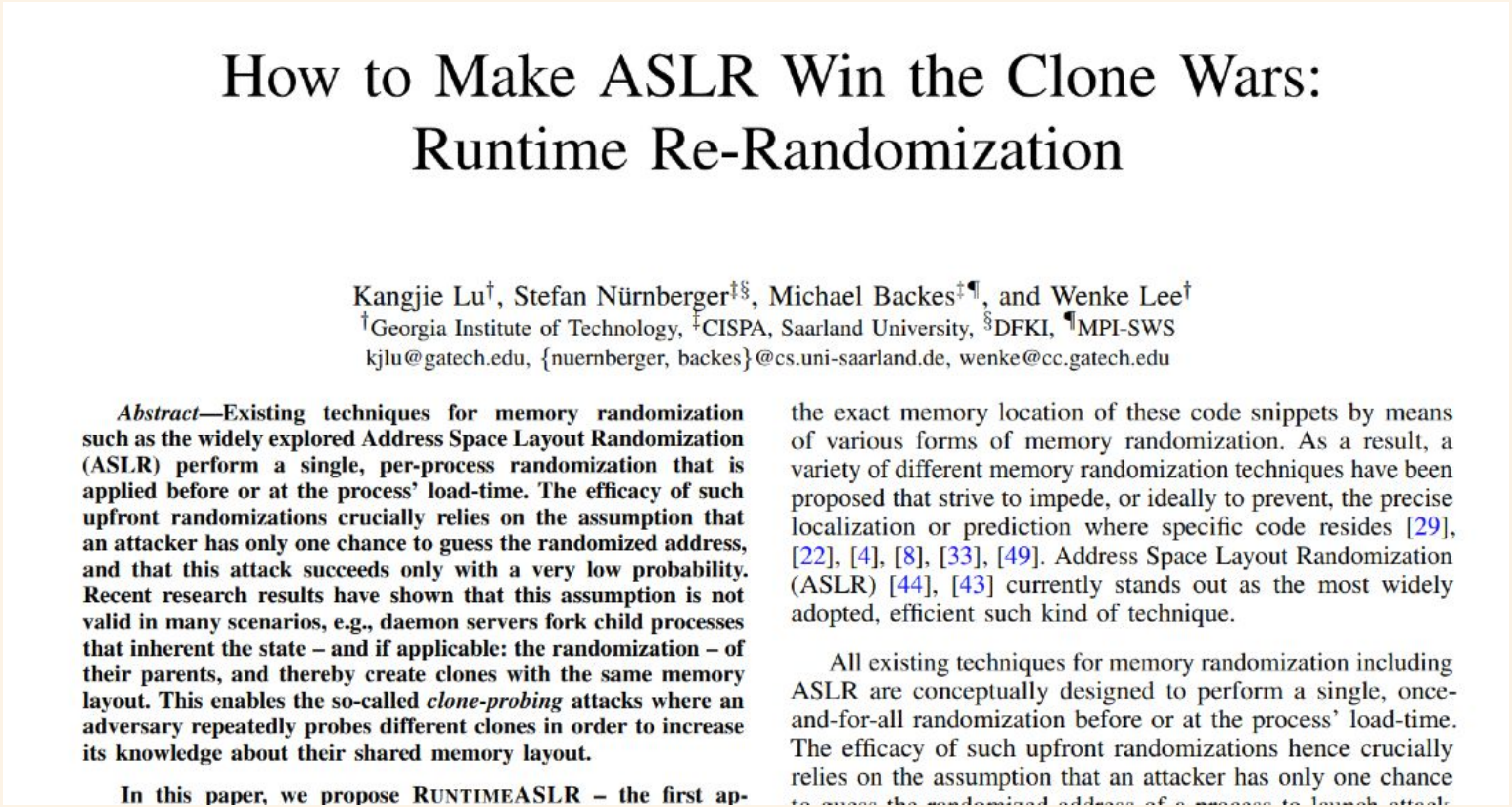
Location: Frnczk 408, North campus

Time: Monday, 5:20 PM - 8:10 PM

Last Class

1. Defenses
    a. Address Space Layout Randomization (ASLR)

Seccomp



How to Make ASLR Win the Clone Wars: Runtime Re-Randomization

Kangjie Lu[†], Stefan Nürnberger[‡§], Michael Backes[‡¶], and Wenke Lee[†]
[†]Georgia Institute of Technology, [‡]CISPA, Saarland University, [§]DFKI, [¶]MPI-SWS
kjlu@gatech.edu, {nuernberger, backes}@cs.uni-saarland.de, wenke@cc.gatech.edu

***Abstract*—Existing techniques for memory randomization such as the widely explored Address Space Layout Randomization (ASLR) perform a single, per-process randomization that is applied before or at the process' load-time. The efficacy of such upfront randomizations crucially relies on the assumption that an attacker has only one chance to guess the randomized address, and that this attack succeeds only with a very low probability. Recent research results have shown that this assumption is not valid in many scenarios, e.g., daemon servers fork child processes that inherent the state – and if applicable: the randomization – of their parents, and thereby create clones with the same memory layout. This enables the so-called *clone-probing* attacks where an adversary repeatedly probes different clones in order to increase its knowledge about their shared memory layout.**

In this paper, we propose RUNTIMEASLR – the first ap-

the exact memory location of these code snippets by means of various forms of memory randomization. As a result, a variety of different memory randomization techniques have been proposed that strive to impede, or ideally to prevent, the precise localization or prediction where specific code resides [29], [22], [4], [8], [33], [49]. Address Space Layout Randomization (ASLR) [44], [43] currently stands out as the most widely adopted, efficient such kind of technique.

All existing techniques for memory randomization including ASLR are conceptually designed to perform a single, once-and-for-all randomization before or at the process' load-time. The efficacy of such upfront randomizations hence crucially relies on the assumption that an attacker has only one chance to guess the randomized address of a process to launch attack

NDSS 2016

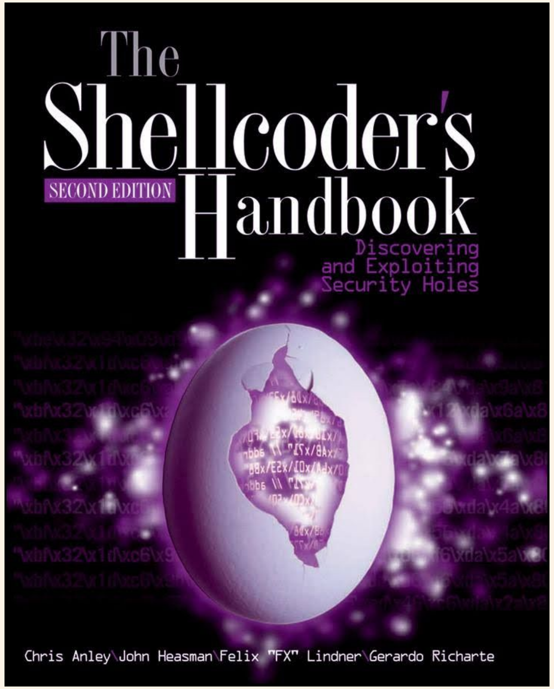
Announcement

Midterm next week. 2hrs.

1. UB Learns (Blackboard)
2. Multiple choice
3. Binary hacking

Today's Agenda

1. Developing shellcode
    a. Non-zero shellcode
    b. Non-printable, non-alphanumeric shellcode
    c. English shellcode


The Shellcoder's Handbook
SECOND EDITION
Discovering and Exploiting Security Holes
Chris Anley\John Heasman\Felix "FX" Lindner\Gerardo Richarte

code/tester.c

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/mman.h>
#include <unistd.h>

int main()
{
	void * page = 0;
	page = mmap(0, 0x1000, PROT_READ|PROT_WRITE|PROT_EXEC, MAP_PRIVATE|MAP_ANON, 0, 0);

	if (!page)
	{
		puts("Fail to mmap.\n");
		exit(0);
	}

	read(0, page, 0x1000);
	((void(*)())page)();
}
```

x86 invoke system call

https://chromium.googlesource.com/chromiumos/docs/+/master/constants/syscalls.md

- Set %eax as target system call number

- Set arguments
  - 1st arg : %ebx
  - 2nd arg: %ecx
  - 3rd arg: %edx
  - 4th arg: %esi
  - 5th arg: %edi

- Run
  - int $0x80

- Return value will be stored in %eax

x86 calling execve()

```
execve(char* filepath, char** argv, char** envp)

execve(“/bin/sh”, NULL, NULL);

%eax = $SYS_execve
%ebx = address of “/bin/sh”
%ecx = 0
%edx = 0
```

x86 how to create a string?

%ebx = address of “/bin/sh”

Use Stack

- Push $0
- push $0x67832f6e // “n/sh”
- push $0x69622f2f // “//bi”
- mov %esp, %ebx

Let us code shellcode32zero.s

```
gcc -m32 -nostdlib -static shellcode32zero.s -o shellcode32zero
objcopy --dump-section .text=shellcode32zero-raw shellcode32zero
```

amd64 invoke system call

https://chromium.googlesource.com/chromiumos/docs/+/master/constants/syscalls.md

- Set %rax as target system call number

- Set arguments
  - 1st arg : %rid
  - 2nd arg: %rsi
  - 3rd arg: %rdx
  - 4th arg: %r10
  - 5th arg: %r8

- Run
  - syscall

- Return value will be stored in %rax

amd64 how to create a string?

Rip-based addressing

```
lea binsh(%rip), %rdi
mov $0, %rsi
mov $0, %rdx
syscall
binsh:
.string "/bin/sh"
```

Let us code shellcode64zero.s

```
gcc -nostdlib -static shellcode64zero.s -o shellcode64zero
objcopy --dump-section .text=shellcode64zero-raw shellcode64zero
```

code/testernozero

```
char buf[0x1000] = {0};

int main()
{
	void * page = 0;
	page = mmap(0, 0x1000, PROT_READ|PROT_WRITE|PROT_EXEC, MAP_PRIVATE|MAP_ANON, 0, 0);

	if (!page)
	{
		puts("Fail to mmap.\n");
		exit(0);
	}

	read(0, buf, 0x1000);
	strcpy(page, buf);
	((void(*)())page)();
}
```

Non-shell shellcode

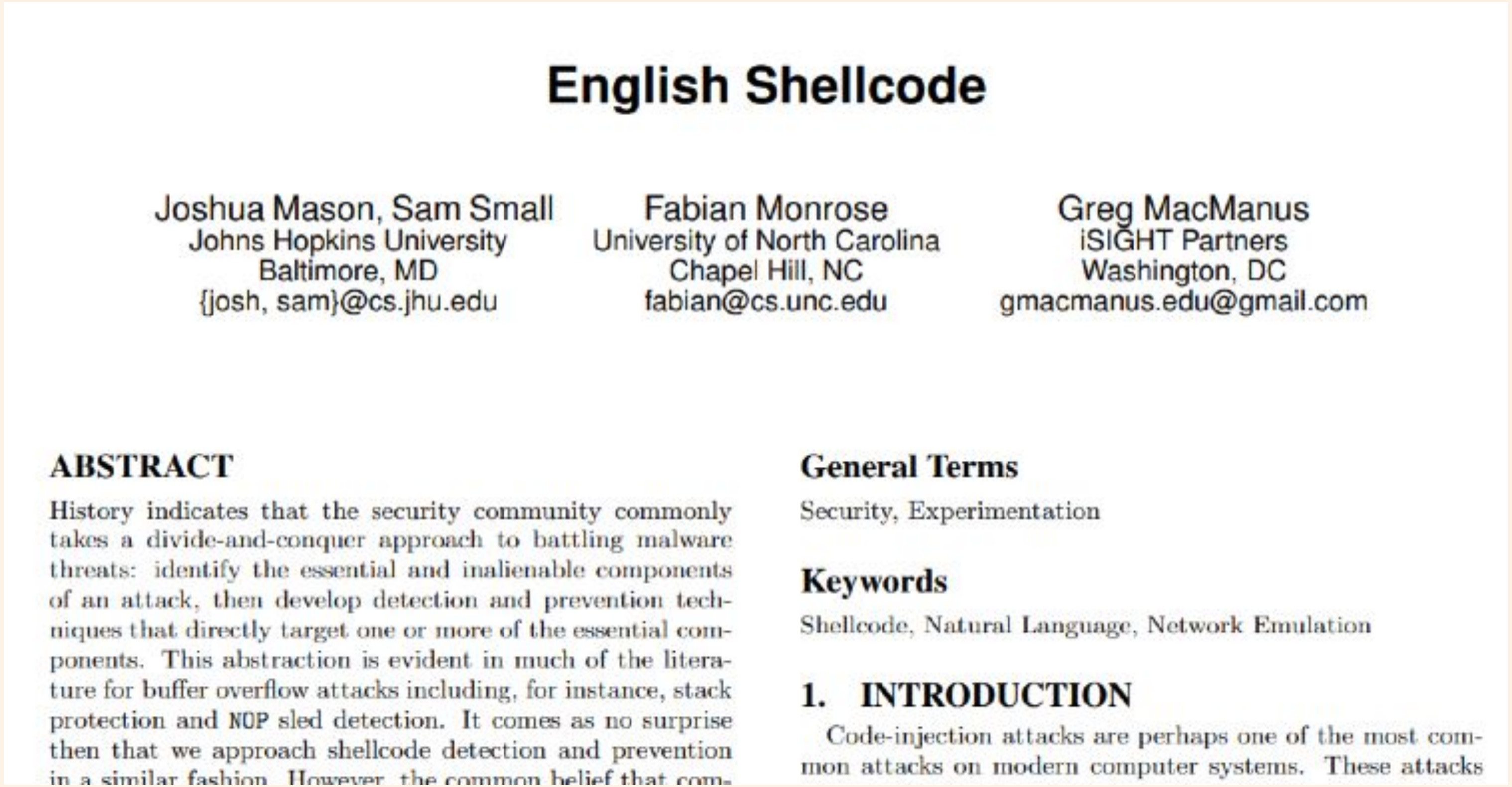
Finish another task but do not return a shell.

Print out the secret file in the folder

code/testerascii

```
char *asciicpy(char *dest, const char *src)
{
        unsigned i;
        for (i = 0; src[i] > 0 && src[i] < 127; ++i)
                dest[i] = src[i];

        return dest;}

int main()
{
        void * page = 0;
        page = mmap(0, 0x1000, PROT_READ|PROT_WRITE|PROT_EXEC, MAP_PRIVATE|MAP_ANON, 0, 0);

        if (!page)
        {
                puts("Fail to mmap.\n");
                exit(0);
        }

        read(0, buf, 0x1000);
        asciicpy(page, buf);
        ((void(*)())page)();}
```

English Shellcode



English Shellcode

Joshua Mason, Sam Small
Johns Hopkins University
Baltimore, MD
{josh, sam}@cs.jhu.edu

Fabian Monrose
University of North Carolina
Chapel Hill, NC
fabian@cs.unc.edu

Greg MacManus
iSIGHT Partners
Washington, DC
gmacmanus.edu@gmail.com

ABSTRACT

History indicates that the security community commonly takes a divide-and-conquer approach to battling malware threats: identify the essential and inalienable components of an attack, then develop detection and prevention techniques that directly target one or more of the essential components. This abstraction is evident in much of the literature for buffer overflow attacks including, for instance, stack protection and NOP sled detection. It comes as no surprise then that we approach shellcode detection and prevention in a similar fashion. However, the common belief that com-

General Terms

Security, Experimentation

Keywords

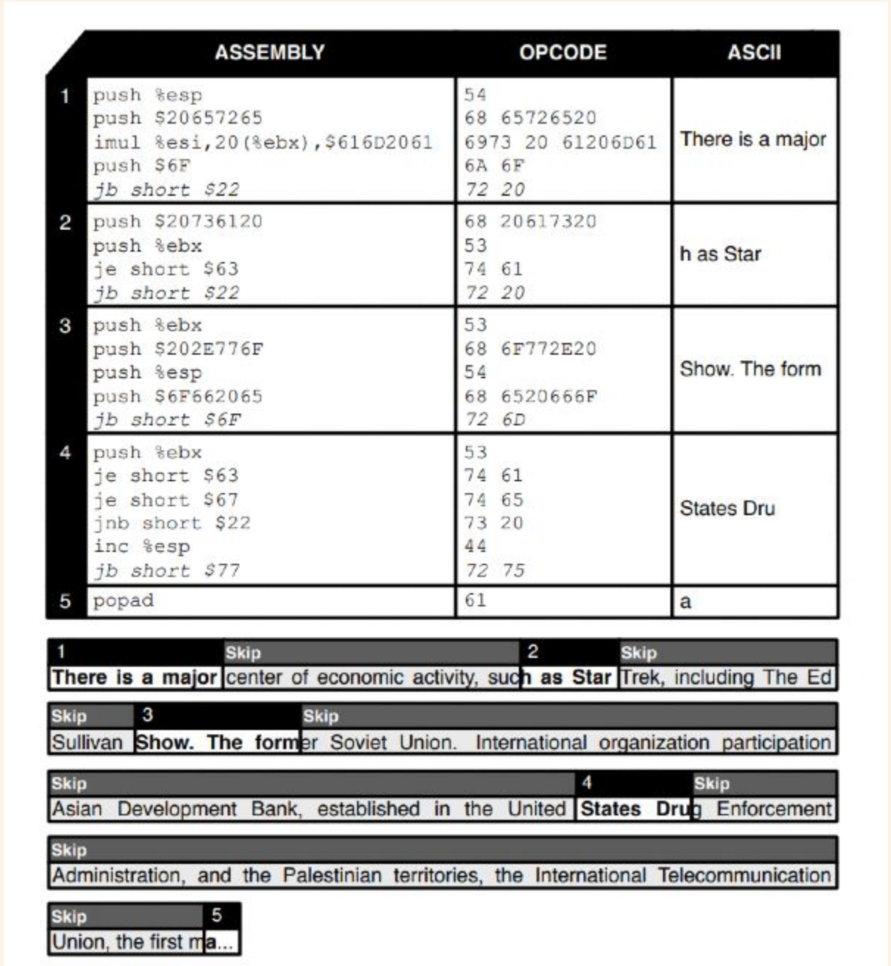
Shellcode, Natural Language, Network Emulation

1. INTRODUCTION

Code-injection attacks are perhaps one of the most common attacks on modern computer systems. These attacks

CCS 2009

English Shellcode



	ASSEMBLY	OPCODE	ASCII
1	push %esp push $20657265 imul %esi,20(%ebx),$616D2061 push $6F *jb short $22*	54 68 65726520 6973 20 61206D61 6A 6F *72 20*	There is a major
2	push $20736120 push %ebx je short $63 *jb short $22*	68 20617320 53 74 61 *72 20*	h as Star
3	push %ebx push $202E776F push %esp push $6F662065 *jb short $6F*	53 68 6F772E20 54 68 6520666F *72 6D*	Show. The form
4	push %ebx je short $63 je short $67 jnb short $22 inc %esp *jb short $77*	53 74 61 74 65 73 20 44 *72 75*	States Dru
5	popad	61	a

1	Skip	2	Skip
There is a major	center of economic activity, suc	**h as Star**	Trek, including The Ed

Skip	3	Skip
Sullivan	**Show. The form**	er Soviet Union. International organization participation

Skip	4	Skip
Asian Development Bank, established in the United	**States Dru**	g Enforcement

Skip
Administration, and the Palestinian territories, the International Telecommunication

Skip	5
Union, the first m	**a...**

How breakpoints work?

int $3

Set breakpoint by yourself.