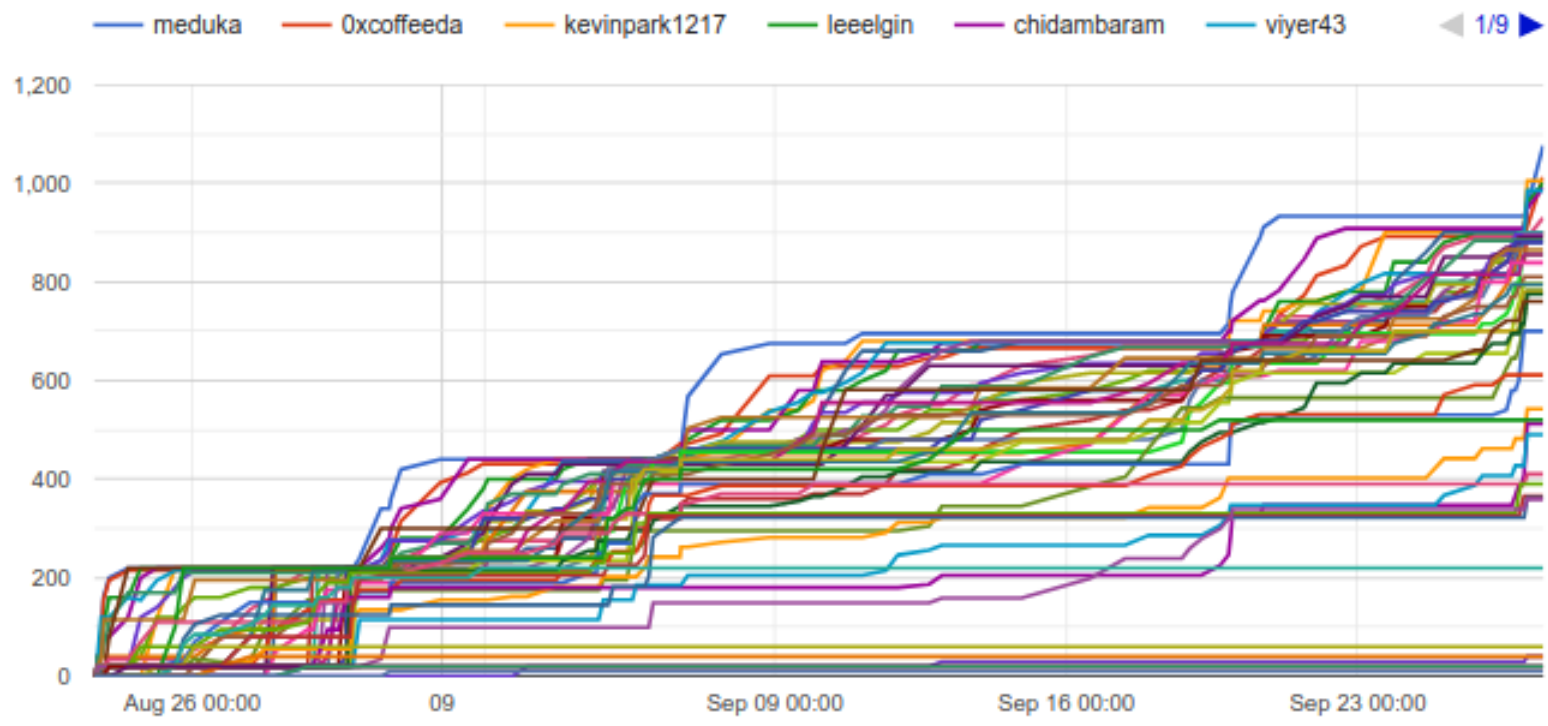
Lec06: DEP and ASLR

Taesoo Kim

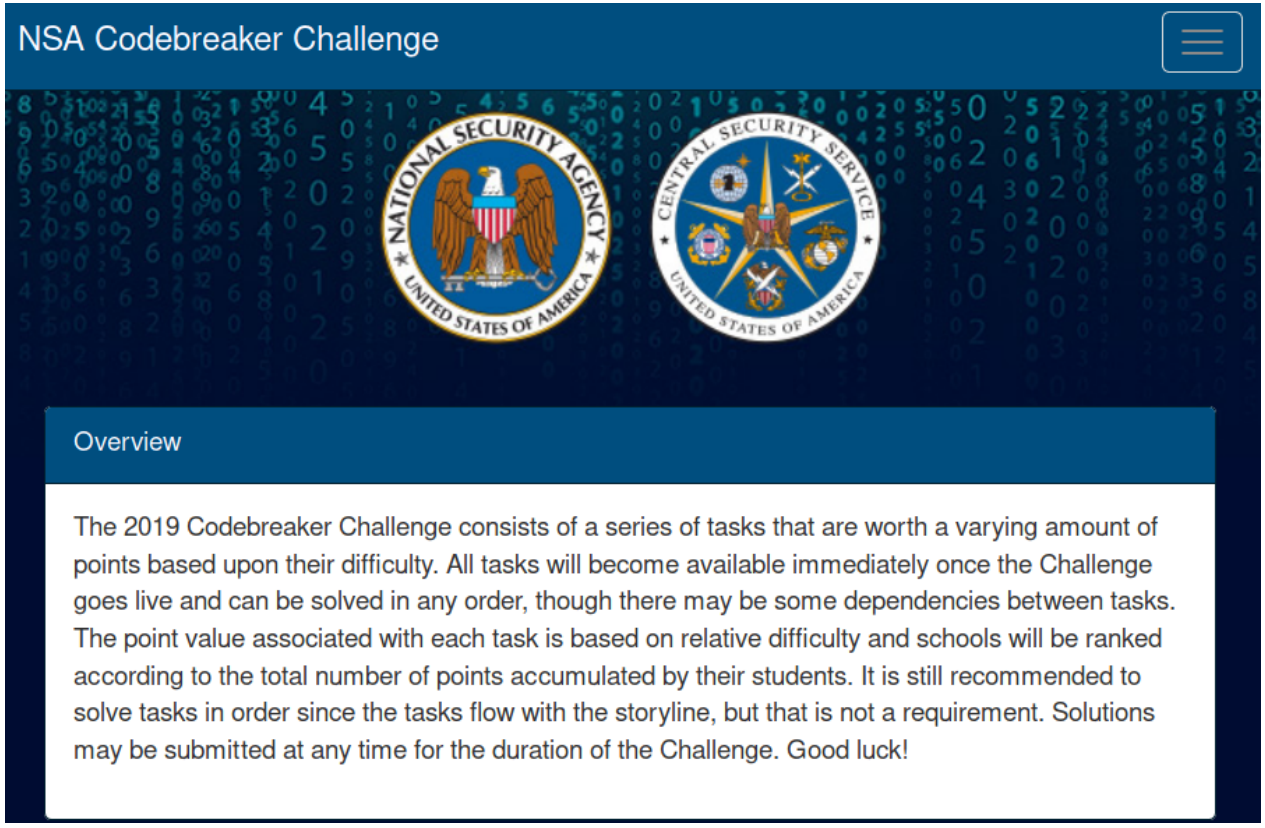
Scoreboard


meduka
0xcoffeeda
kevinpark1217
leeelgin
chidambaram
viyer43
1/9
1,200
1,000
800
600
400
200
0
Aug 26 00:00
09
Sep 09 00:00
Sep 16 00:00
Sep 23 00:00

Administrivia

- Due: Lab05 is out and its due on Oct 3 at midnight
- Lab10: NSA Codebreaker Challenge → Due: Dec 06
- In-class CTF (Nov 22/23): Please find your team mates (3-4 people)!

NSA Codebreaker Challenges



NSA Codebreaker Challenges

Background

DISCLAIMER - The following is a FICTITIOUS story meant for providing realistic context for the Codebreaker Challenge and is not tied in any way to actual events.

Tech savvy terrorists have developed a new suite of communication tools to use for attack planning purposes. Their most recent creation — TerrorTime — is a secure mobile chat application that runs on Android devices. This program is of particular interest since recent intelligence suggests the majority of their communications are happening via this app. Your mission is to reverse-engineer and develop new exploitation capabilities to help discover and thwart future attacks before they happen. There are 7 tasks of increasing difficulty that you will be working through as part of this challenge. Ultimately, you will be developing capabilities that will enable the following:

1. Spoof TerrorTime messages
2. Masquerade (i.e., authenticate) as TerrorTime users without knowledge of their credentials
3. Decrypt TerrorTime chat messages

NSA Codebreaker Challenges

- Lab10 (out of 200 pt)
  - Task1/2/3: 100
  - Task 4: 140
  - Task 5: 200 (super!)
  - Task 6: 250 (A)
  - Task 7: 300 (A+)

All subject to change depending on the quality/difficulty.

Design Choices for the Stack Canary

- Where to put? (e.g., right above ra? fp? local vars?)
- Which value should I use? (e.g., secrete? random?)
- How often do we generate the canary? (e.g., per exec? per func?)
- How to check its integrity? (e.g., xor? cmp?)
- What to do after you find corrupted? (e.g., crash? report?)

Best Write-ups for Lab04

xor	**mliu366, Aditi**
stackshield	**cosmicrao, vishiswoz**
weak-random	**mliu366, abhineet**
gs-random	**vishiswoz, abhineet**
terminator	**subuavudai, viyer43**
assassination	**viyer43, Aditi**
mini-heartbleed	**yiqincai, viyer43**
pltgot	**mliu366, viyer43**
ssp	**0xcoffeeda, mliu366**
fd	**0xcoffeeda, yiqincai**

Summary: Lab04

- Insecure materialization of canary-based protection:
  - xor: known secret
  - stackshield: incorrect checks
  - weak-random: guessable
  - terminator: unprotected fp

Summary: Lab04

- Abusing the canary implementation itself:
  - pltgot: hijacking ssp's plt
  - ssp: overwriting a pointer to the program name
- Fundamental limitations:
  - assassination: local variable → arbitrary write
  - fd: local variable → vtable
  - mini-heartbleed: leaked canary

Introducing DEP/ASLR

```
$ checksec target
[*] '/home/lab05/libbase/target'
Arch:     i386-32-little
RELRO:    Partial RELRO
Stack:    No canary found   <- lab04
NX:       NX enabled        <- lab05
PIE:      PIE enabled       <- lab05
```

- Data Execution Prevention (DEP, aka X^W or NX)
- Address Space Layout Randomization (ASLR, PIE)

ASLR

```
$ cat /proc/sys/kernel/randomize_va_space
2

$ ./check
stack   : 0xff930aa0
system(): 0xf7521c50
printf(): 0xf7536670

$ ./check
stack   : 0xff930250
system(): 0xf755dc50
printf(): 0xf7572670
```

Today's Tutorial

- Learning a power class of bug, format string bug
  - A format string bug → an arbitrary read
  - A format string bug → an arbitrary write
  - A format string bug → an arbitrary execution

Format String: e.g., printf()

- How does printf() know of #arguments passed?
- How do we access the arguments in the function?

```
1) printf("hello: %d", 10);
2) printf("hello: %d/%d", 10, 20);
3) printf("hello: %d/%d", 10, 20, 30);
```

Format String: e.g., printf()

- What does it happen if we miss one argument?

```
// buggy
3) printf("hello: %d/%d/%d", 10, 20);
```

Format String: e.g., printf()

- What does printf() print out? guess?

```
printf("%d/%d/%d", 10, 20)

     +----(n)----+
     |           v
[ra][fmt][10][20][??][..]
         (1) (2) (3) ....
```

About a “Variadic” Function

```
int sum_up(int count,...) {
  va_list ap;
  int i, sum = 0;

  va_start (ap, count);
  for (i = 0; i < count; i++)
    sum += va_arg (ap, int);

  va_end (ap);
  return sum;
}
```

About a “Variadic” Function

```
va_start (ap, count);
  lea   eax,[ebp+0xc]             // Q1. 0xc?
  mov   DWORD PTR [ebp-0x18],eax

for (i = 0; i < count; i++)
  sum += va_arg (ap, int);

  mov   eax,DWORD PTR [ebp-0x18]
  lea   edx,[eax+0x4]             // Q2. +4?
  mov   DWORD PTR [ebp-0x18],edx
  mov   eax,DWORD PTR [eax]
  add   DWORD PTR [ebp-0x10],eax
  ...
```

In-class Tutorial

- Enhanced crackme0x00
  - Step1: A format string bug → an arbitrary read
  - Step2: A format string bug → an arbitrary write
  - Step3: A format string bug → an arbitrary execution

Format String Specifiers

```
printf(fmt);

%p: pointer
%s: string
%d: int
%x: hex

Tip 1.
   %[nth]$p
   (e.g., %1$p = first argument)
```

Arbitrary Read

- If fmtbuf locats on the stack (perhaps, one of caller's),
- Then, we can essentially control its argument!

```
printf(fmtbuf)
printf("\xaa\xbb\xcc\xdd%3$s")

     +---(3rd)---+
     |           v
[ra][fmt][a1][a2][\xaa\xbb\xcc\xdd%3$s]
         (1) (2) (3) ....

                     (1)(2)(3)
=> printf("...%3$s", _, _, 0xddccbbaa)
```

More Format Specifiers

```
printf("1234%n", &len) => len=4

%n: write #bytes
%hn (short), %hhn (byte)

Tip 2.
    %10d: print an int on 10-space word
    (e.g., "        10")
```

Write (sth) to an Arbitrary Location

- Similar to the arbitrary read, we can control the arguments!

```
printf("\xaa\xbb\xcc\xdd%3$n")

     +---(3rd)---+
     |           v
[ra][fmt][a1][a2][\xaa\xbb\xcc\xdd%3$n]
         (1) (2) (3) ....

                    (1)(2)(3)
=> printf("...%3$n", _, _, 0xddccbbaa)
   *0xddccbbaa = 4 (#chars printed so far)
```

Arbitrary Write

- In fact, we can control what to write (see more in the tutorial)!

```
printf("\xaa\xbb\xcc\xdd%6c%3$n")

     +---(3rd)---+
     |           v
[ra][fmt][a1][a2][\xaa\xbb\xcc\xdd%6c%3$n]
         (1) (2) (3) ....

=> *0xddccbbaa = strlen("\xaa\xbb\xcc\xdd......") = 10
```

In-class Tutorial

- Step1: A format string bug → an arbitrary read
- Step2: A format string bug → an arbitrary write
- Step3: A format string bug → an arbitrary execution

```
$ ssh lab05@3.95.14.86
Password: <password>

$ cd tut05-fmtstr
$ cat README
```

References

- Bypassing ASLR
- Advanced return-into-lib(c) exploits
- Format string vulnerability