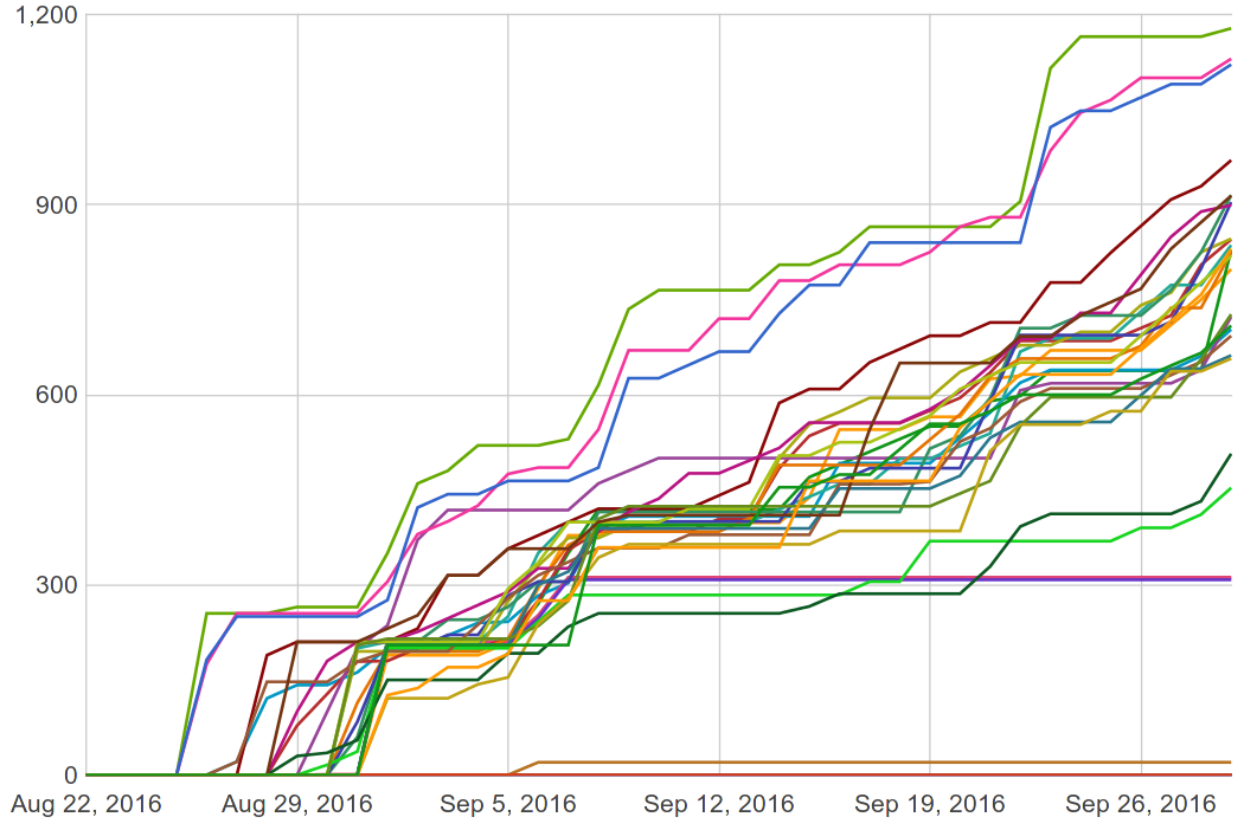
Lec06: DEP and ASLR

Taesoo Kim

Scoreboard


1,200
900
600
300
0
Aug 22, 2016
Aug 29, 2016
Sep 5, 2016
Sep 12, 2016
Sep 19, 2016
Sep 26, 2016

NSA Codebreaker Challenges

University	Task 1	Task 2	Task 3	Task 4	Task 5	Task 6
Georgia Institute of Technology	48	38	34	27	10	3
Carnegie Mellon University	24	22	13	9	5	2
Dakota State University	53	37	23	17	8	0
Naval Postgraduate School	5	5	5	5	4	0
University of Colorado at Colorado Springs	12	10	8	8	3	0
Rensselaer Polytechnic Institute	5	4	4	3	2	0
University of Tulsa	5	5	3	2	2	0
University of Maryland, Baltimore County	26	21	11	9	1	0
Purdue University	11	9	6	5	1	0
United States Military Academy	7	6	5	4	1	0

Showing 1 to 10 of 326 entries

Previous 1 2 3 4 5 … 33 Next

Administrivia

- Congrats!! We've completed the half of labs!
- Due: Lab06 is out and its due on Oct 6 at midnight
- NSA Codebreaker Challenge → Due: Dec 1
- We'll release new lab every Thursday at 10pm
- Oct 14 : About Web Penetration Testing
- If you haven't read yet, please read Zixuan & Insu's posts on Piazza!

Lab05: Stack Protection

Name	Points	Release	Deadline	Solved
xor	20	09-23-2016 00:00:00	09-30-2016 00:00:00	25
stackshield	20	09-23-2016 00:00:00	09-30-2016 00:00:00	25
weak-random	20	09-23-2016 00:00:00	09-30-2016 00:00:00	25
gs-random	20	09-23-2016 00:00:00	09-30-2016 00:00:00	18
terminator	20	09-23-2016 00:00:00	09-30-2016 00:00:00	17
assassination	20	09-23-2016 00:00:00	09-30-2016 00:00:00	22
mini-heartbleed	20	09-23-2016 00:00:00	09-30-2016 00:00:00	8
pltgot	20	09-23-2016 00:00:00	09-30-2016 00:00:00	16
ssp	20	09-23-2016 00:00:00	09-30-2016 00:00:00	23
fd	20	09-23-2016 00:00:00	09-30-2016 00:00:00	8

Discussion: Lab05

- What's the most "annoying" bug or challenge?
- What's the most "interesting" bug or challenge?
- So, should we use canary or not?
- So, which one would you like to use?

Take-outs from Stack Canary?

- Stack Canary indirectly protects the "integrity" of RA, funcptr, etc
  - (e.g., exploitation mitigation → NX, canary)
- We better prevent buffer overflows at the first place
  - (e.g., code analysis, better APIs)

Subtle Design Choices

- Where to put the canary? (e.g., right above RA? fp? local vars?)
- Which value should I use as a canary? (e.g., secrete? random? per exec? per func?)
- How to compare the canary value? (e.g., xor? cmp?)
- What to do after you find the canary value is corrupted? (e.g., crash? report?)

Discussion: How to make exploitation difficult?

- What if the stack address (or code/heap) is random?
  - Could you exploit any challenge in the last week?
- What if the stack/heap memory is not executable?
  - Then, where to put your shellcode?

Today's Tutorial

- In-class tutorial:
  - About: format String vulnerability
  - Format string to arbitrary read
  - Format string to arbitrary write
  - (optional) Format string to arbitrary execution

Format string: *printf

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

Format string: *printf

```
printf("%d/%d/%d", a1, a2 ...)

     +----(n)----+
     |           v
[ra][fmt][a1][a2][a3][..]
         (1) (2) (3) ....
```

Format string specifiers

```
printf(fmt);

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

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

Arbitrary Read

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

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

-> "\xaa\xbb\xcc\xdd[value]"
```

More Format Specifiers

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

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

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

Write (sth) to an Arbitrary Location

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

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

-> "\xaa\xbb\xcc\xdd" = 4
```

Arbitrary Write

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

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

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

In-class Tutorial

- Step1: Format string to arbitrary read
- Step2: Format string to arbitrary write
- Step3: (optional) Format string to arbitrary execution

```
$ git git@clone tc.gtisc.gatech.edu:seclab-pub cs6265
or
$ git pull
$ cd cs6265/lab06
$ ./init.sh

$ cd tut
$ cat README
```

References

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

Lec06: DEP and ASLR

Taesoo Kim