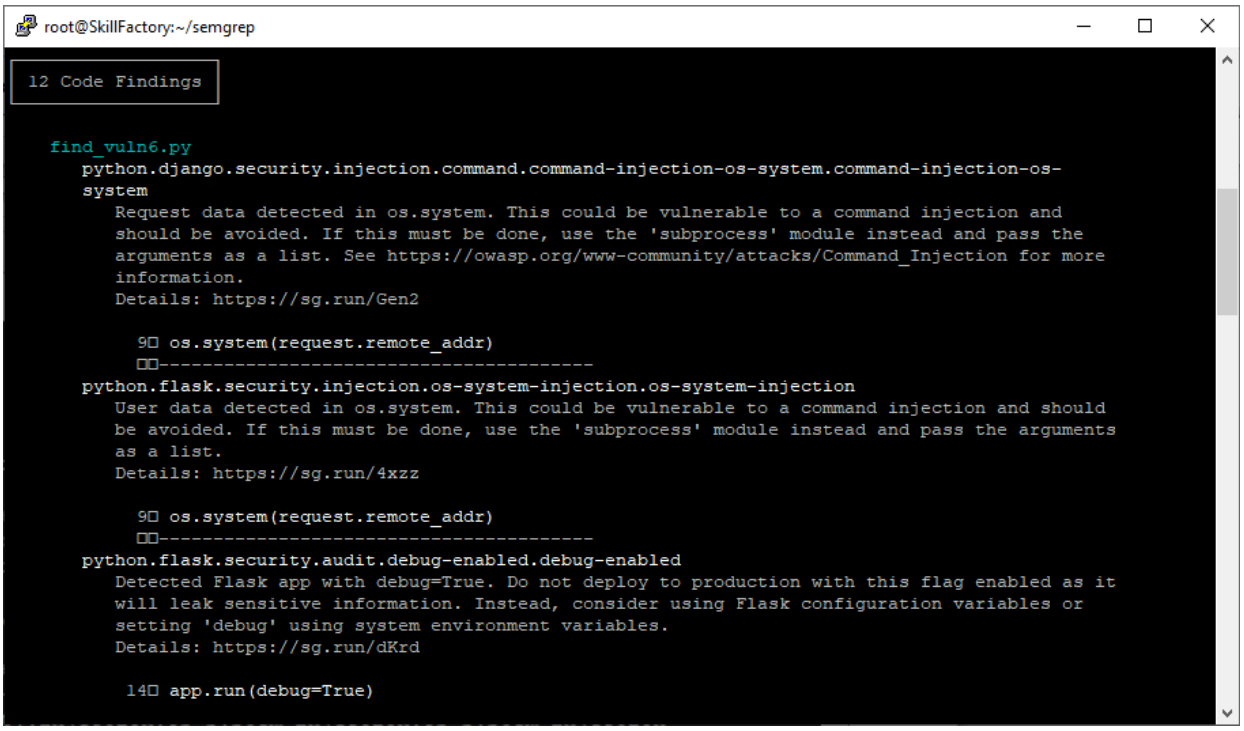
Сканирование файла **find_vuln6.py** на уязвимости


root@SkillFactory:~/semgrep

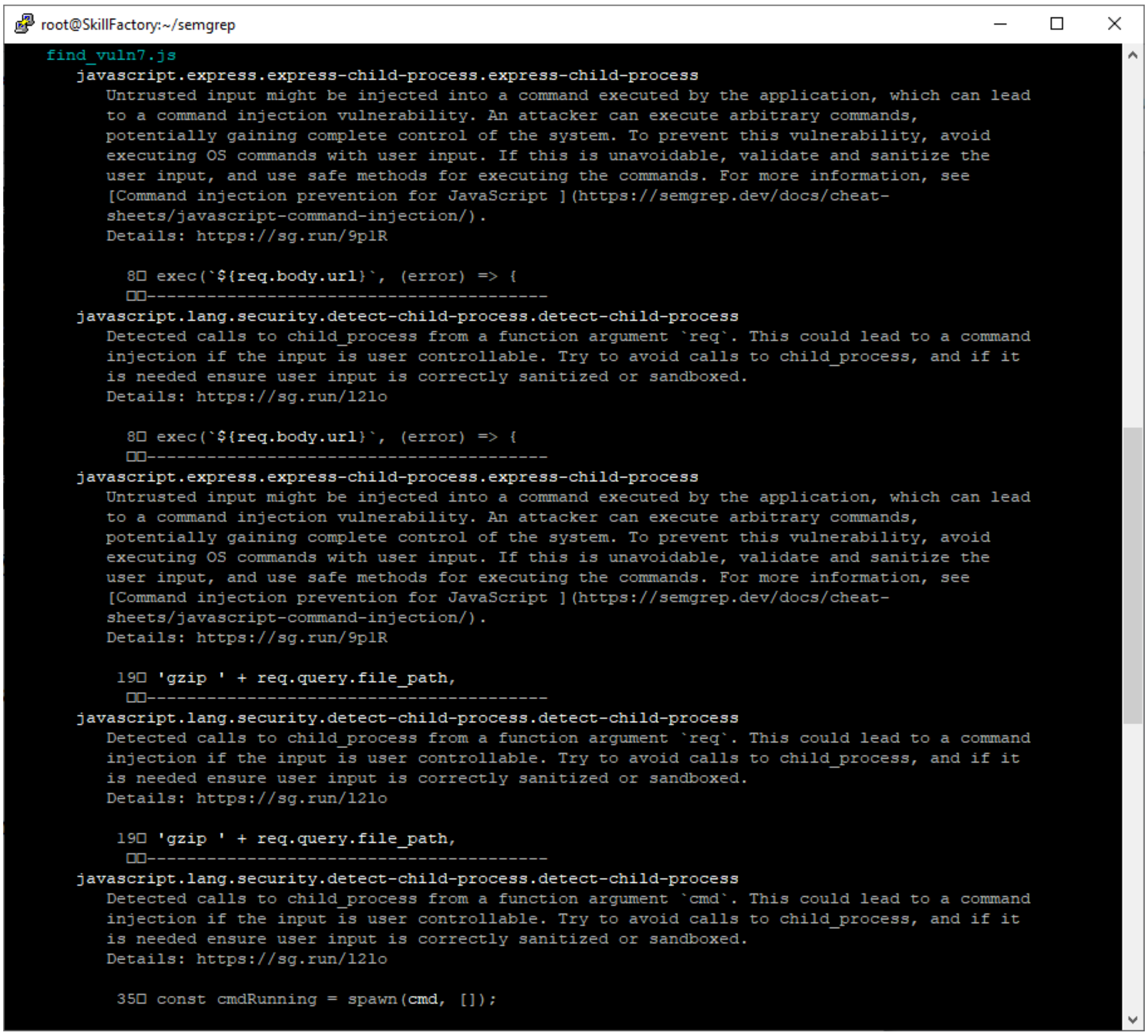
```
 12 Code Findings 


    find_vuln6.py
       python.django.security.injection.command.command-injection-os-system.command-injection-os-
       system
          Request data detected in os.system. This could be vulnerable to a command injection and
          should be avoided. If this must be done, use the 'subprocess' module instead and pass the
          arguments as a list. See https://owasp.org/www-community/attacks/Command_Injection for more
          information.
          Details: https://sg.run/Gen2

            9  os.system(request.remote_addr)
            ----------------------------------------
       python.flask.security.injection.os-system-injection.os-system-injection
          User data detected in os.system. This could be vulnerable to a command injection and should
          be avoided. If this must be done, use the 'subprocess' module instead and pass the arguments
          as a list.
          Details: https://sg.run/4xzz

            9  os.system(request.remote_addr)
            ----------------------------------------
       python.flask.security.audit.debug-enabled.debug-enabled
          Detected Flask app with debug=True. Do not deploy to production with this flag enabled as it
          will leak sensitive information. Instead, consider using Flask configuration variables or
          setting 'debug' using system environment variables.
          Details: https://sg.run/dKrd

           14  app.run(debug=True)
```

В файле найдены три уязвимости:

Command Injection (два раза)

Debug Enabled

Сканирование файла **find_vuln7.js** на уязвимости



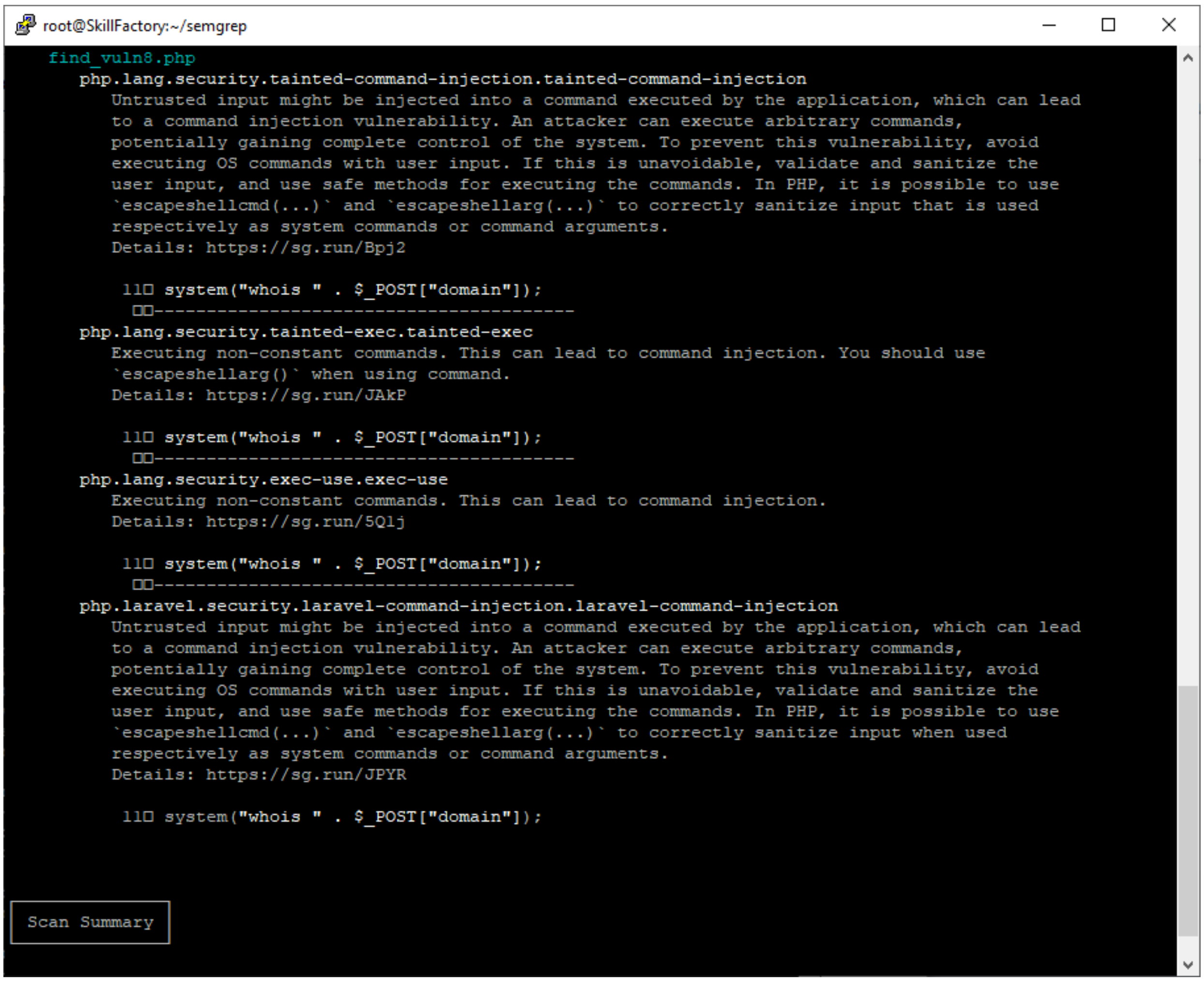
root@SkillFactory:~/semgrep

```
    find_vuln7.js
       javascript.express.express-child-process.express-child-process
          Untrusted input might be injected into a command executed by the application, which can lead
          to a command injection vulnerability. An attacker can execute arbitrary commands,
          potentially gaining complete control of the system. To prevent this vulnerability, avoid
          executing OS commands with user input. If this is unavoidable, validate and sanitize the
          user input, and use safe methods for executing the commands. For more information, see
          [Command injection prevention for JavaScript ](https://semgrep.dev/docs/cheat-
          sheets/javascript-command-injection/).
          Details: https://sg.run/9p1R

            8□ exec(`${req.body.url}`, (error) => {
            □□------------------------------------------
       javascript.lang.security.detect-child-process.detect-child-process
          Detected calls to child_process from a function argument `req`. This could lead to a command
          injection if the input is user controllable. Try to avoid calls to child_process, and if it
          is needed ensure user input is correctly sanitized or sandboxed.
          Details: https://sg.run/l2lo

            8□ exec(`${req.body.url}`, (error) => {
            □□------------------------------------------
       javascript.express.express-child-process.express-child-process
          Untrusted input might be injected into a command executed by the application, which can lead
          to a command injection vulnerability. An attacker can execute arbitrary commands,
          potentially gaining complete control of the system. To prevent this vulnerability, avoid
          executing OS commands with user input. If this is unavoidable, validate and sanitize the
          user input, and use safe methods for executing the commands. For more information, see
          [Command injection prevention for JavaScript ](https://semgrep.dev/docs/cheat-
          sheets/javascript-command-injection/).
          Details: https://sg.run/9p1R

           19□ 'gzip ' + req.query.file_path,
            □□------------------------------------------
       javascript.lang.security.detect-child-process.detect-child-process
          Detected calls to child_process from a function argument `req`. This could lead to a command
          injection if the input is user controllable. Try to avoid calls to child_process, and if it
          is needed ensure user input is correctly sanitized or sandboxed.
          Details: https://sg.run/l2lo

           19□ 'gzip ' + req.query.file_path,
            □□------------------------------------------
       javascript.lang.security.detect-child-process.detect-child-process
          Detected calls to child_process from a function argument `cmd`. This could lead to a command
          injection if the input is user controllable. Try to avoid calls to child_process, and if it
          is needed ensure user input is correctly sanitized or sandboxed.
          Details: https://sg.run/l2lo

           35□ const cmdRunning = spawn(cmd, []);
```

В файле найдены 5 уязвимости:

Command Injection (5 раз)

Сканирование файла **find_vuln8.php** на уязвимости


root@SkillFactory:~/semgrep

```
   find_vuln8.php
      php.lang.security.tainted-command-injection.tainted-command-injection
         Untrusted input might be injected into a command executed by the application, which can lead
         to a command injection vulnerability. An attacker can execute arbitrary commands,
         potentially gaining complete control of the system. To prevent this vulnerability, avoid
         executing OS commands with user input. If this is unavoidable, validate and sanitize the
         user input, and use safe methods for executing the commands. In PHP, it is possible to use
         `escapeshellcmd(...)` and `escapeshellarg(...)` to correctly sanitize input that is used
         respectively as system commands or command arguments.
         Details: https://sg.run/Bpj2

          11□ system("whois " . $_POST["domain"]);
           □□----------------------------------------
      php.lang.security.tainted-exec.tainted-exec
         Executing non-constant commands. This can lead to command injection. You should use
         `escapeshellarg()` when using command.
         Details: https://sg.run/JAkP

          11□ system("whois " . $_POST["domain"]);
           □□----------------------------------------
      php.lang.security.exec-use.exec-use
         Executing non-constant commands. This can lead to command injection.
         Details: https://sg.run/5Q1j

          11□ system("whois " . $_POST["domain"]);
           □□----------------------------------------
      php.laravel.security.laravel-command-injection.laravel-command-injection
         Untrusted input might be injected into a command executed by the application, which can lead
         to a command injection vulnerability. An attacker can execute arbitrary commands,
         potentially gaining complete control of the system. To prevent this vulnerability, avoid
         executing OS commands with user input. If this is unavoidable, validate and sanitize the
         user input, and use safe methods for executing the commands. In PHP, it is possible to use
         `escapeshellcmd(...)` and `escapeshellarg(...)` to correctly sanitize input when used
         respectively as system commands or command arguments.
         Details: https://sg.run/JPYR

          11□ system("whois " . $_POST["domain"]);


 Scan Summary
```

В файле найдены 4 уязвимости:

Command Injection (4 раз)