Secure Code Inspection

This lab will cover the following topics:

- Secure coding best practices
- Vulnerable code patterns for every programming language
- Automating secure code scanning tools

Automatic secure code inspection script in Linux

For this approach, we recommend an all-in-one shell script, the **Code Review Audit Script Scanner** (**CRASS**). This one script includes everything needed for secure code scanning, and it defines the secure code scanning patterns for Java, JSP, Flex Flash, .NET, PHP, HTML, Android, iOS, Python, Ruby, and C. It can easily be extended by editing the [grep-it.sh] file. We may use the same vulnerable Python project from before as our example for the following steps.

Step 1 -- downloading the CRASS

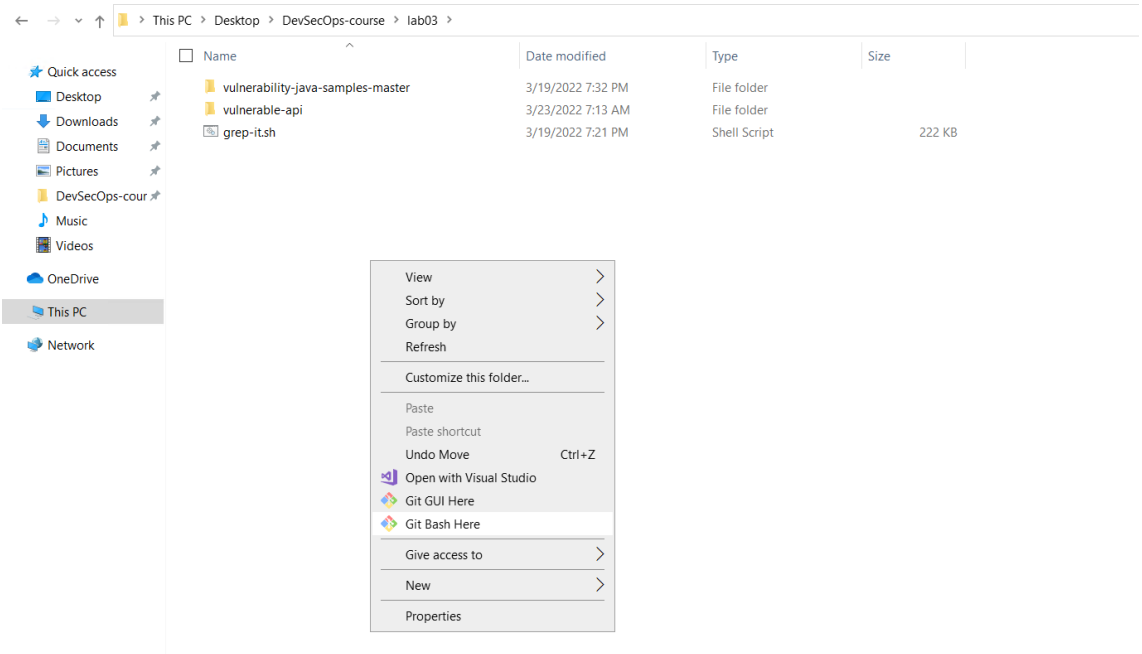
`grep-it.sh` script has been downloaded already from here: **https://github.com/floyd-fuh/crass/blob/master/grep-it.sh** in `lab03` folder.

Step 2 -- executing the code review audit scan

Following repository contains an example Python API that is vulnerable to several different web API attacks. It has been cloned already in `lab03` folder. https://github.com/mattvaldes/vulnerable-api

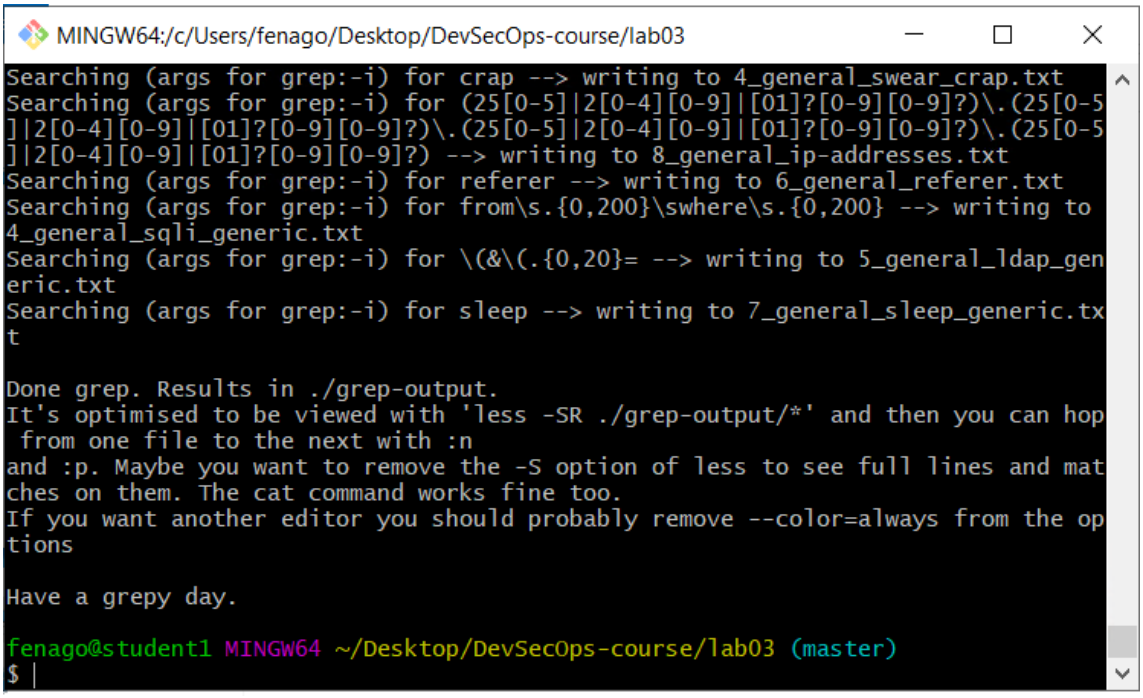
Execute the command with a parameter to specify the target project folder. The following command will scan the vulnerable source code under the [/vulnerable-api] folder:

Important: Make sure to run this exercise from git bash:



```
bash  grep-it.sh  ./vulnerable-api
```

Wait for scan to complete for few minutes.



Step 3 -- reviewing the results

Once the scanning is done, the scanning results will be output under the [\grep-output] folder of the target scanning project.

The scanning results will be generated into files separated by security topic, as shown in the following diagram:

```
fenago@student1 MINGW64 ~/Desktop/DevSecOps-course/lab03/grep-output (master)
$ ls
2_general_sql_injection.txt                5_general_exec_wide.txt
2_general_uris_auth_info_wide.txt          5_general_exploit.txt
2_general_xss_lowercase.txt                5_general_http_urls.txt
3_apikeys_TOKEN.txt                        5_general_https_urls.txt
3_dotnet_unsafe_declaration.txt            5_general_popen_wide.txt
4_cryptocred_ciphers_sha1_lowercase.txt    5_general_relative_paths.txt
4_cryptocred_ciphers_sha1_uppercase.txt    5_general_sql_cursor.txt
4_cryptocred_password.txt                  5_general_sql_sqlite.txt
4_general_popen_narrow.txt                 5_general_system_wide.txt
4_general_sql_insert.txt                   5_html_text_plain.txt
4_general_sql_select.txt                   5_python_is_object_identity_operator_general.txt
4_general_sqli_generic.txt                 5_python_is_object_identity_operator_right.txt
4_general_swear_stupid.txt                 6_android_access_query.txt
4_js_node_get_generic.txt                  6_cryptocred_hash.txt
4_python_float_equality_general.txt        6_cryptocred_hexdigest.txt
5_cryptocred_authentication.txt            6_general_update.txt
5_cryptocred_credentials_wide.txt          6_java_sql_execute.txt
5_general_backticks.txt                    6_modsecurity_block.txt
5_general_base64_content.txt               6_php_echo_high_volume.txt
5_general_base64_urlsafe.txt               6_php_print_high_volume.txt
5_general_bypass.txt                       7_cryptocred_ciphers_des.txt
5_general_deny.txt                         7_php_type_unsafe_comparison.txt
5_general_eval_wide.txt                    8_general_ip-addresses.txt

fenago@student1 MINGW64 ~/Desktop/DevSecOps-course/lab03/grep-output (master)
$
```

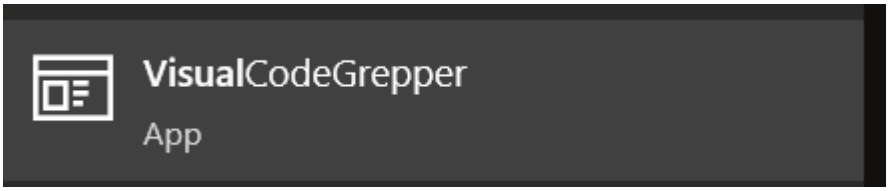
Task: Analyze all these files one by one:

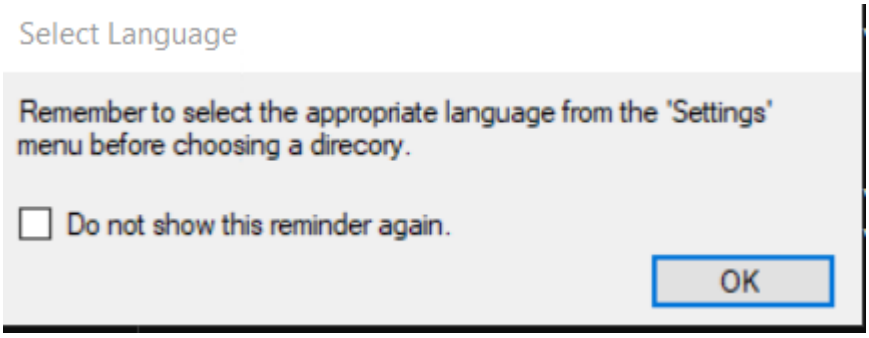
```
fenago@student1 MINGW64 ~/Desktop/DevSecOps-course/lab03/grep-output (master)
$ cat 6_java_sql_execute.txt
# Info: The syntax for SQL executions start with execute and this should as well catch generic execute calls.
# Filename 6_java_sql_execute.txt
# Example: executeBlaBla(
# False positive example: FALSE_POSITIVES_EXAMPLE_PLACEHOLDER
# Grep args:
# Search regex: execute.{0,20}\(
./vulnerable-api/ansible/roles/api/files/vAPI.py-67-    # no sql parameterization
./vulnerable-api/ansible/roles/api/files/vAPI.py-68-    user_query = "SELECT * FROM users WHERE username = '%s' A
ND password = '%s'" % (
./vulnerable-api/ansible/roles/api/files/vAPI.py-69-        username, password)
./vulnerable-api/ansible/roles/api/files/vAPI.py:70:    c.execute(user_query)
./vulnerable-api/ansible/roles/api/files/vAPI.py-71-    user = c.fetchone()
--
./vulnerable-api/ansible/roles/api/files/vAPI.py-77-        # removing them...
./vulnerable-api/ansible/roles/api/files/vAPI.py-78-        token_query = "SELECT * FROM tokens WHERE userid = '%
s' ORDER BY expires DESC" % (user[
./vulnerable-api/ansible/roles/api/files/vAPI.py-79-
                                0])
./vulnerable-api/ansible/roles/api/files/vAPI.py:80:        c.execute(token_query)
./vulnerable-api/ansible/roles/api/files/vAPI.py-81-        token_record = c.fetchone()
--
./vulnerable-api/ansible/roles/api/files/vAPI.py-88-                token = hashlib.md5(expire_date).hexdigest()
./vulnerable-api/ansible/roles/api/files/vAPI.py-89-                # we'll parameterize this one because we need
 this serious
./vulnerable-api/ansible/roles/api/files/vAPI.py-90-                # functionality
```

Automatic secure code inspection tools for Windows

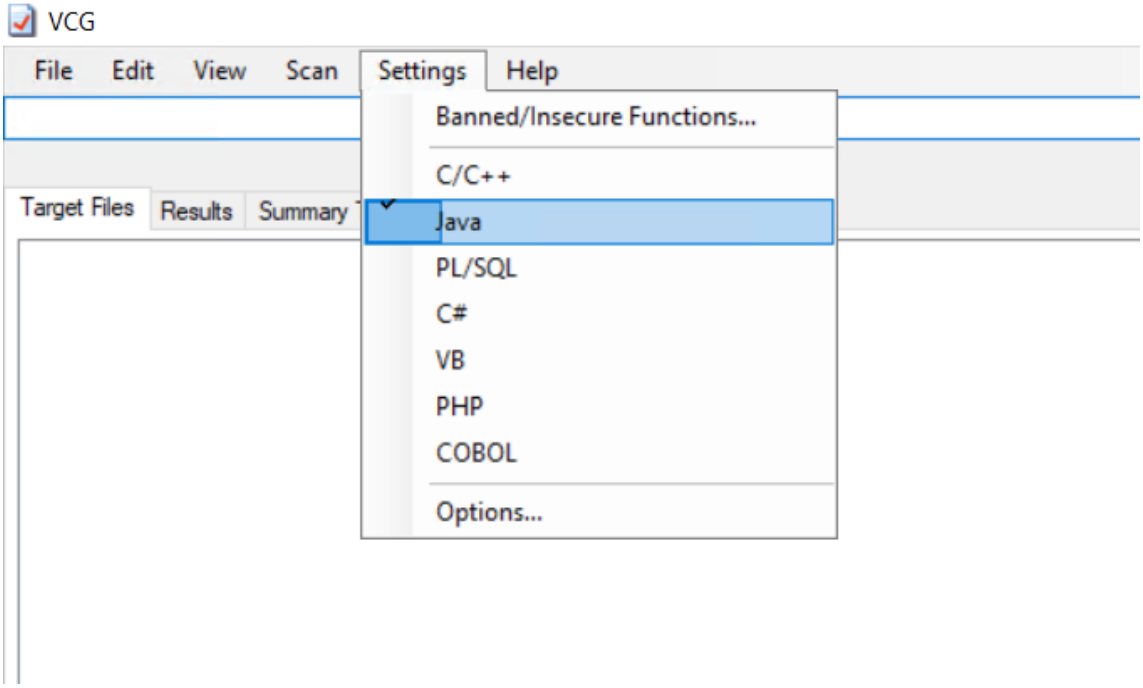
Step: Executing VCG

Search `VisualCodeGrepper` to directly launch VCG in GUI mode:

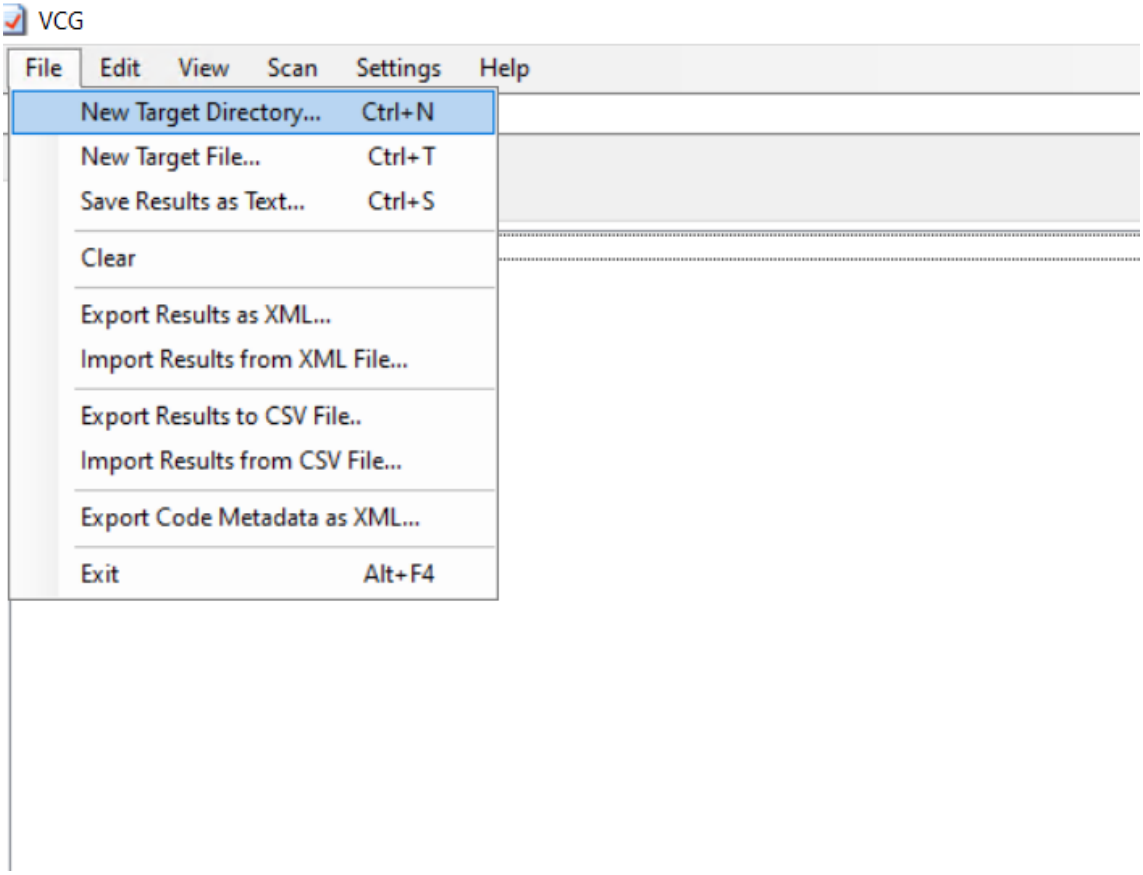


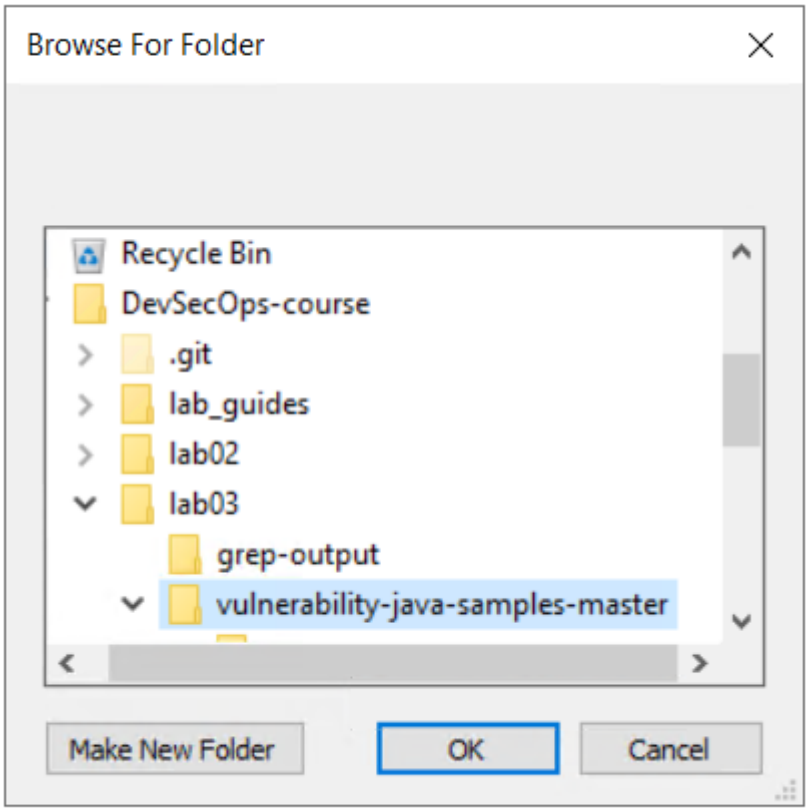


Click **Settings** and select `Java` :

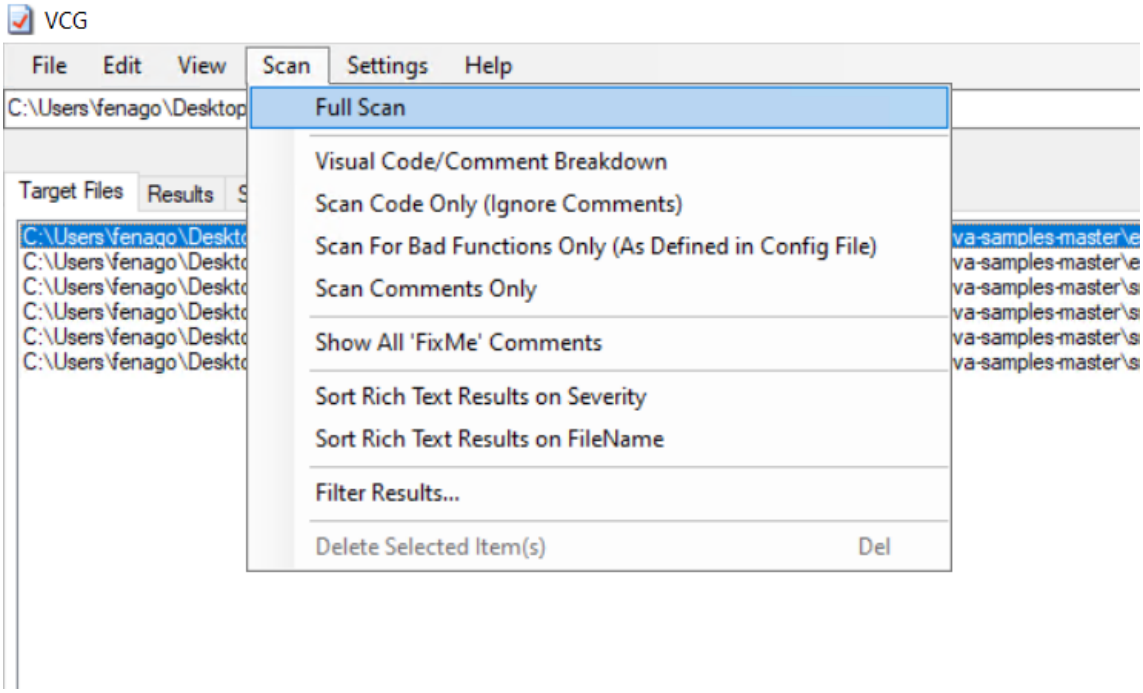


Click **File** and select `New Target Directory`:



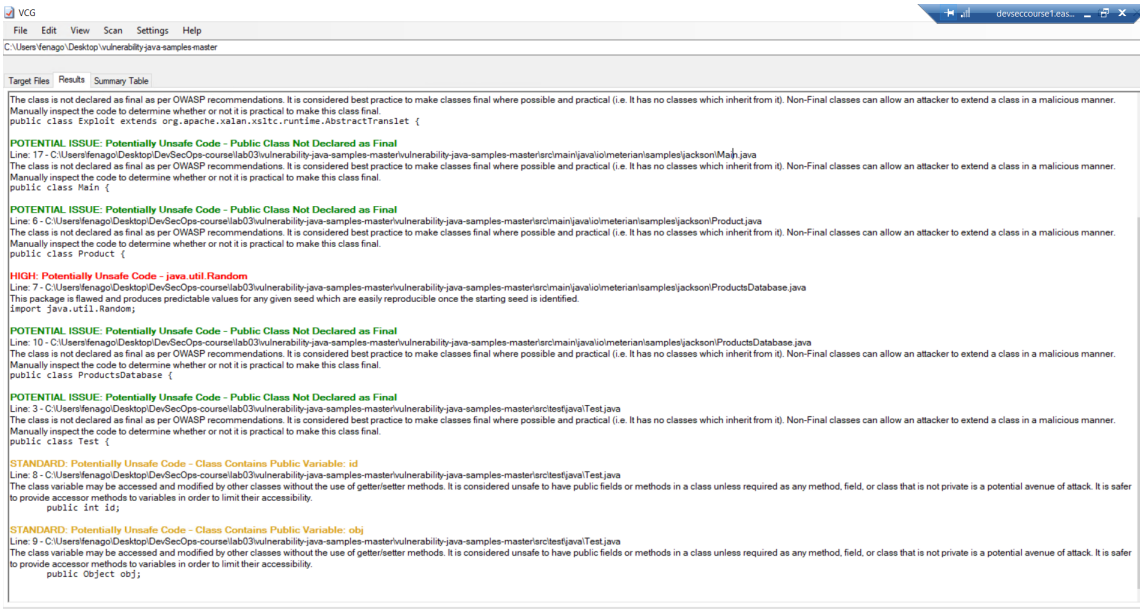


Click **Scan** and select `Full Scan`:



Step 3: Reviewing the VCG scanning results

Output:



Click **File** and select `Export Results to CSV file`

You may use the [VCG GUI] | [File] | [Import Results from CSV File] | [test1.csv] to review the results with highlighted colors.

Summary

In a case study of this lab, we demonstrated the use of CRASS to scan vulnerable Python APIs. Furthermore, we also introduced another generic general secure coding inspection tool, VCG.

In the coming lab, we will apply similar code inspection techniques to look for sensitive information leakage and privacy security issues.