Development System of Application Security Testing

Abdul Razaque[1], Saule Amanzholova[1], Amir Akimbayev[1], Emil Kovalenko[1], and Dilnaz Ashimzhanova[1]

[1] *International Information Technology University, Manas St. 34/1, Almaty, 050040, Kazakhstan*

Abstract

The main objective of the scientific work was to use theoretical and practical knowledge gained during studying at the university, as well as their further application in the development of a scientific work on the system of application security testing - AST. Various statistical data were collected on which products of this type are present on the world market, as well as on the market of Kazakhstan, analysis of the security of banks' web resources, and much more. Also, to achieve this goal, available resources associated with the algorithms of the work of statistical and dynamic analysis were studied, as well as piloting similar products from foreign vendors, which are activelyused in Kazakhstan banks. In this work, libraries and tools such as Django, Libsast, Bandit, Semgrep, and other dependencies were used. During the scientific work, a corporate-level web application was developed that scans and analyzes the repository of the downloaded application to further display statistics and a summary of the vulnerabilities found, as well as methods for solvingthem in a single JSON format. This scientific work is aimed at implementation in the banks of the Republic of Kazakhstan and its use by security departments and code review.

Keywords

AST, SAST, DAST, Django, Libsast, Bandit, code review, Semgrep, JSON

1. Introduction

The relevance. The entry level of attackers is the application. Each application has vulnerabilityin cases that we are currently observed. Companies waste money on ineffective Pentest audits, which aren't effective, against which they order a system that can find and resolve all vulnerabilities. Even the largest companies have poor quality and security of source code [1].

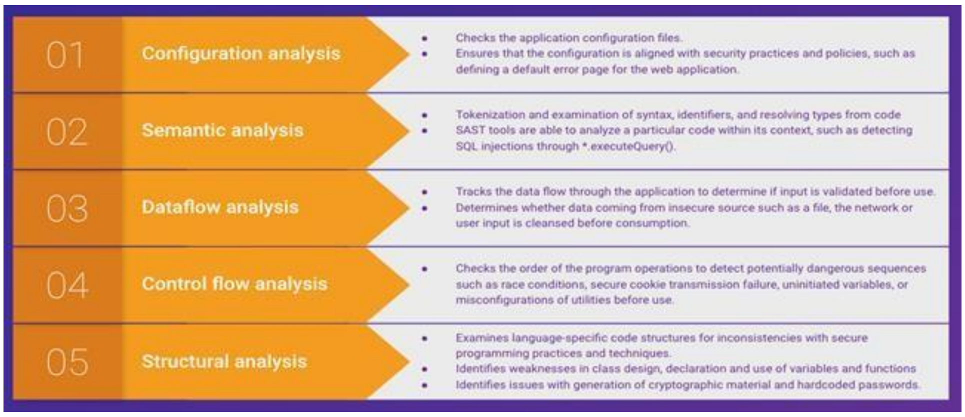
The scientific novelty. The existing approaches through the use of manual auditing are too outdated, but the approach of Static Code AST is the most modern way to do this. This product will adhere to modern application security standards, as well as be able to flexibly customize the functionality for the task set by the company and the platform on which this product will be used, and integration tools. There are only a few vendors offer tools for static code analysis tools. It’s easy to implement, to add a static scanner to your development pipeline and provide feedback on a potential problem.

2. Main part

The prepared scientific work from our team is aimed at testing application security, but we don'twant to limit ourselves to SAST, or other AST. We want to present a product near to ASTO - Application security testing orchestration. Main goal of this scientific work is to provide a comprehensive

Proceedings of the 7th International Conference on Digital Technologies in Education, Science and Industry (DTESI 2022), October 20-21, 2022, Almaty, Kazakhstan
EMAIL: a.razaque@edu.iitu.kz (Abdul Razaque); s.amanzholova@iitu.edu.kz (Saule Amanzholova); 25106@edu.iitu.kz (Amir Akimbayev); kovv.emil@gmail.com (Emil Kovalenko); ashimzhanova00@mail.ru (Dilnaz Ashimzhanova)
ORCID: 0000-0003-0409-3526 (Abdul Razaque); 0000-0002-6779-9393 (Saule Amanzholova)

protection by scanning applications for vulnerabilities, starting from static scanning, and ending with other types of scanning. Integration with software development life cycle gives neededefficiency of product, it also includes analytics of the scans.[2] Process of testing code will be delegated by modules starting from SAST - static scanning codeon vulnerabilities by white box testing, source code will be scanned by parsers, and scanners. SAST, it's a very old and stable type of scan, it takes responsibility for undeclared possibilities. Our SAST willemploy predetermined rules, such as coding errors in the source code that must be corrected. Idea of SAST is to scan but not to start or run it. Use of open-source vulnerabilities databases will increase efficiency, and time by time, for example if some critical vulnerability will born into world, and start toinfect all computers, our solution will send update to servers to find and destroy this type of vulnerabilities, or use WAF, IDS, IPS system to close this gateway before vulnerability can be used tohack to system, so it’s update, upgrade and control system [3].



01	Configuration analysis	• Checks the application configuration files. • Ensures that the configuration is aligned with security practices and policies, such as defining a default error page for the web application.
02	Semantic analysis	• Tokenization and examination of syntax, identifiers, and resolving types from code • SAST tools are able to analyze a particular code within its context, such as detecting SQL injections through *.executeQuery().
03	Dataflow analysis	• Tracks the data flow through the application to determine if input is validated before use. • Determines whether data coming from insecure source such as a file, the network or user input is cleansed before consumption.
04	Control flow analysis	• Checks the order of the program operations to detect potentially dangerous sequences such as race conditions, secure cookie transmission failure, uninitiated variables, or misconfigurations of utilities before use.
05	Structural analysis	• Examines language-specific code structures for inconsistencies with secure programming practices and techniques. • Identifies weaknesses in class design, declaration and use of variables and functions • Identifies issues with generation of cryptographic material and hardcoded passwords.

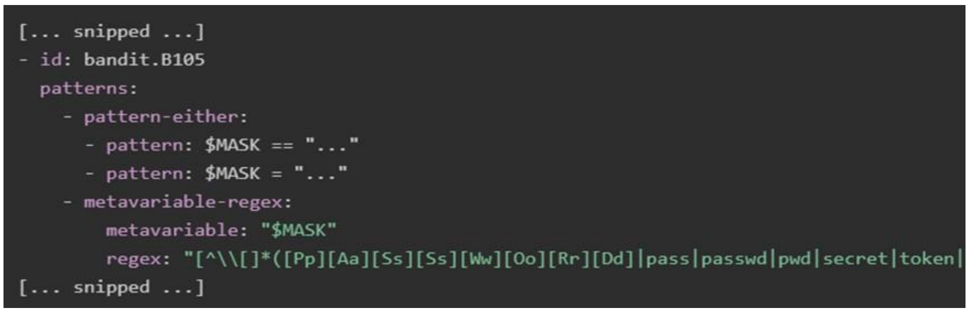
Figure 1: Mechanism of SAST analysis

At the final stage a system engine will be used that will work by list of vulnerabilities to find false positives and delegate them. It will be very useful for security officers who have not so much timeto look at each vulnerability every time.

Profit of a scientific work product that is not framed by method of scanning application, will give effort in Kazakhstan, and worldwide against other big solutions in the market. Against comparisonof other product, it will have more efficiency and more possibilities to defense scientific work by the fact that it’s complex solution that customer can buy and be sure about security safety of applications [4].

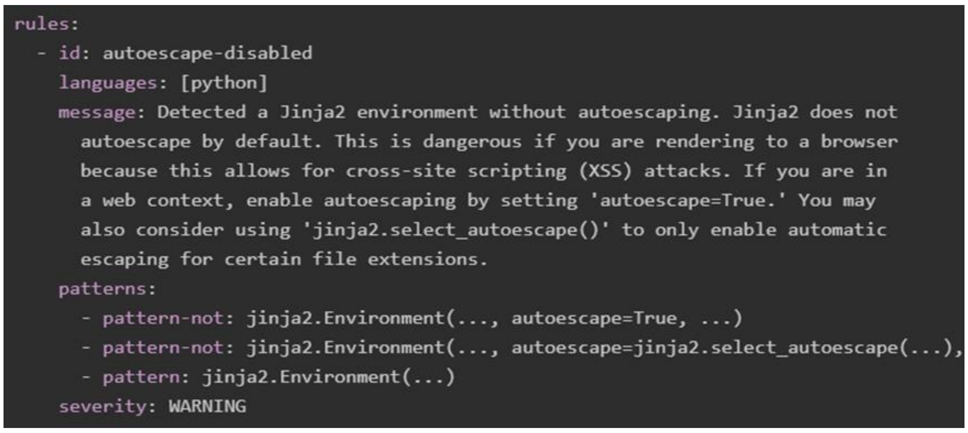
3. Technical implementation

In addition to the main Django library, also used two libraries for the regex-based pattern matching - Libsast and Bandit. To implement the semantic analysis, the Semgrep libraries were used, which can only be used in the Linux environment[5]. For them, self-written various rules were independently written that meet the standards of modern actual vulnerabilities such as Log4J, NPM, etc.



```
[... snipped ...]
- id: bandit.B105
  patterns:
    - pattern-either:
      - pattern: $MASK == "..."
      - pattern: $MASK = "..."
    - metavariable-regex:
        metavariable: "$MASK"
        regex: "[^\\[]*([Pp][Aa][Ss][Ss][Ww][Oo][Rr][Dd]|pass|passwd|pwd|secret|token|
[... snipped ...]
```

Figure 2: Example of B105 - hardcoded_password_string Custom Rule in Bandit



```yaml
rules:
  - id: autoescape-disabled
    languages: [python]
    message: Detected a Jinja2 environment without autoescaping. Jinja2 does not
      autoescape by default. This is dangerous if you are rendering to a browser
      because this allows for cross-site scripting (XSS) attacks. If you are in
      a web context, enable autoescaping by setting 'autoescape=True.' You may
      also consider using 'jinja2.select_autoescape()' to only enable automatic
      escaping for certain file extensions.
    patterns:
      - pattern-not: jinja2.Environment(..., autoescape=True, ...)
      - pattern-not: jinja2.Environment(..., autoescape=jinja2.select_autoescape(...),
      - pattern: jinja2.Environment(...)
    severity: WARNING
```

Figure 3: Example of autoescape-disabled Custom Rule in Semgrep

In order to ensure the smooth operation of users from the very beginning, it was necessary to design the login screen as intuitively as possible. This screen is usually the first step for the user when getting acquainted with the product, moreover, security is important when authenticating users.

The user interface has been designed to be user-friendly. The initial login page uses two-factor authentication and without the possibility of registration (determined by internal security protection purposes) in this case, using the product license, a regulated number of accesses is issued. The 2FA application of Google Code Authenticator works perfectly and it correctly uses the reading of the generated tokens corresponding to the generation on the server. In the case of corporate product development, there is a field input check, which will give the user, first of all, a convenient start with the platform [6].

The EternalSec user interface looks and works as familiar to most users. The navigation step structure on the left makes it easier to move inside the platform, for a more user-friendly functionality, the ability to switch to dark-mode has been developed[7]. The top panel displays information about the workspace or task selected in the left panel. Multiple users can simultaneously access the user interface through web browsers. In the same system-wide database, users have access to the same informationdetails. The permissions provided to each user control access to tasks and objects.

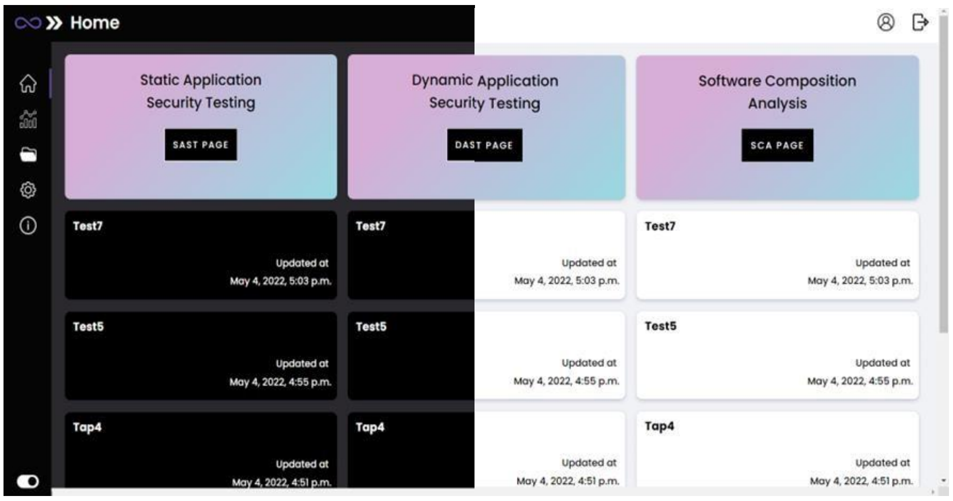

Home
Static Application Security Testing
SAST PAGE
Dynamic Application Security Testing
DAST PAGE
Software Composition Analysis
SCA PAGE
Test7
Updated at May 4, 2022, 5:03 p.m.
Test5
Updated at May 4, 2022, 4:55 p.m.
Tap4
Updated at May 4, 2022, 4:51 p.m.

Figure 4: Dark/Light modes

The home page can save the mode that the user prefers in a session variable. The view switchesthe mode that the user wants to use - in turn, the binding tag references this view to switch the css mode. Saving the user's theme in localStorage and every time the page loads, js-code is launched, which sets the css of the user's choice. To use the default theme, the information is stored in cookies and allows

Django to deliver pages with the right theme instead of relying on js speed [8].

The Analytics section allows you to see basic statistics for all scans and compare the results ofthe analysis of the most common vulnerabilities.

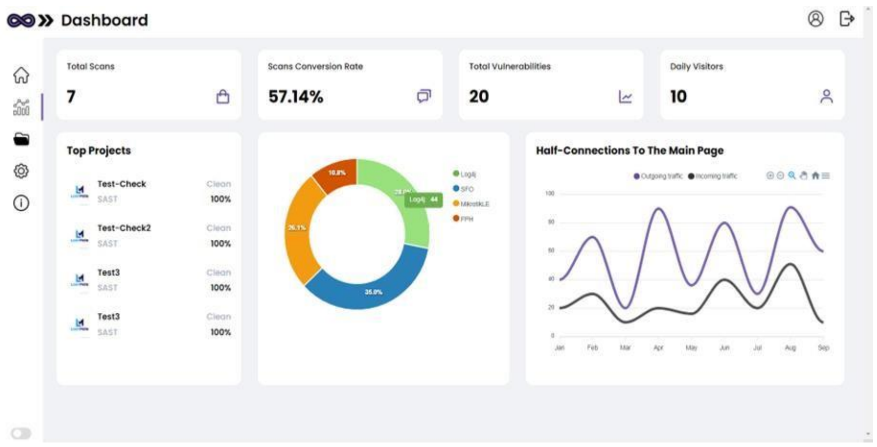


Figure 5: Dashboard Page

1. See the list — the most recent five scans are displayed, followed by a link to the Scans page, which contains the full list of scans;
2. View statistics of the number of scans, taking statuses of the average value for all past scans;
3. The number of vulnerabilities (taking into account the level of criticality);
4. View the load while scanning web applications to account for half connections in case of afailure.

The Overview page provides the following information:

- Scan duration;
- Repository diagram in the form of a hierarchical tree;
- A diagram with the number of vulnerabilities of each level of criticality in the scan;
- Navigation menu.

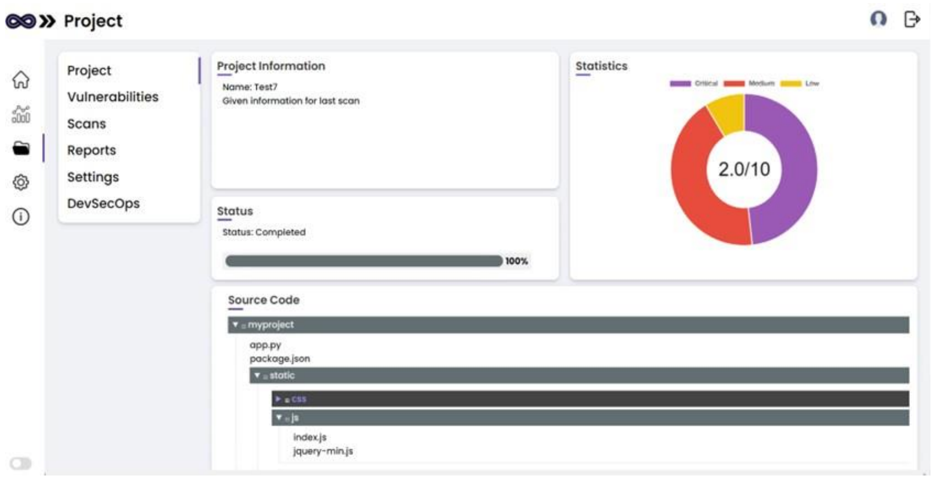
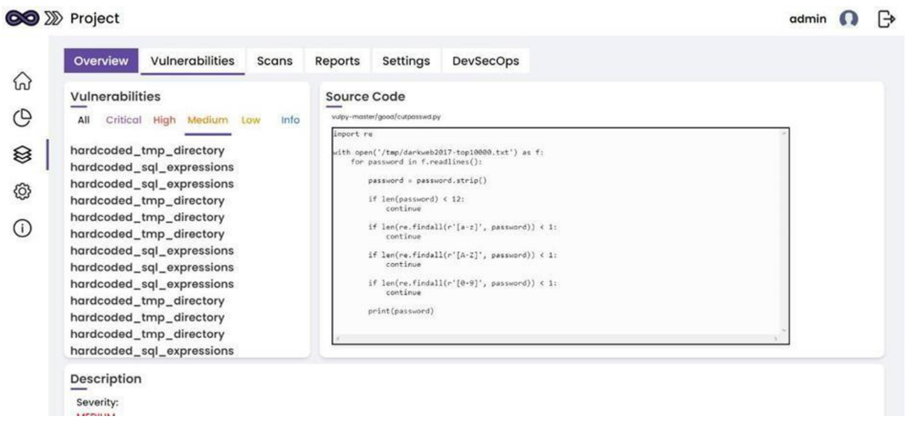


Figure 6: Scan information

The vulnerabilities tab contains extended information about them, including default parameters (CWE, id, description, severity), classifications as links to relevant items in CWE, HIPAA, FSTEC database, OWASP, CWE/SANS Top 25, PCI DSS, taking into account the output onthe right side of the processed file [9].



hardcoded_tmp_directory
hardcoded_sql_expressions
hardcoded_sql_expressions
hardcoded_tmp_directory
hardcoded_tmp_directory
hardcoded_tmp_directory
hardcoded_sql_expressions
hardcoded_sql_expressions
hardcoded_sql_expressions
hardcoded_tmp_directory
hardcoded_tmp_directory
hardcoded_tmp_directory
hardcoded_sql_expressions

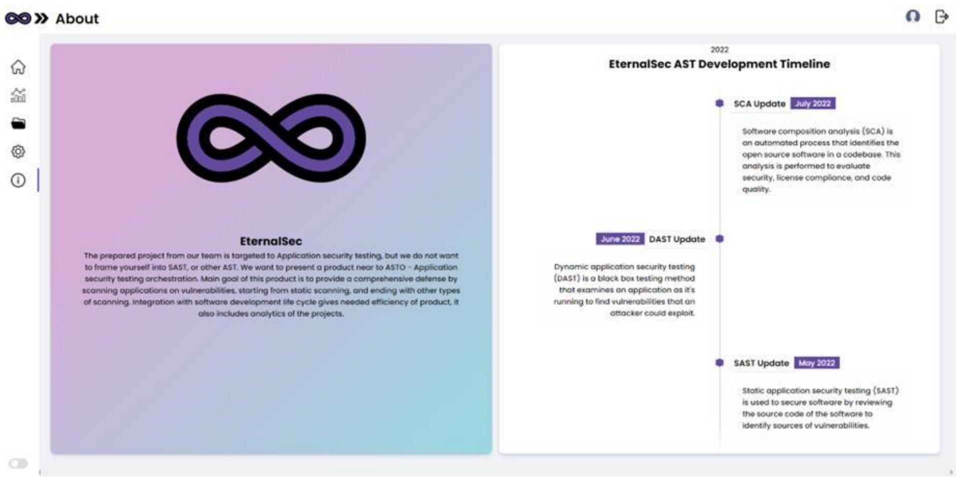
vulpy-master/good/cutpasswd.py

```
import re

with open('/tmp/darkweb2017-top10000.txt') as f:
    for password in f.readlines():

        password = password.strip()

        if len(password) < 12:
            continue

        if len(re.findall(r'[a-z]', password)) < 1:
            continue

        if len(re.findall(r'[A-Z]', password)) < 1:
            continue

        if len(re.findall(r'[0-9]', password)) < 1:
            continue

        print(password)
```

Severity:

Figure 7: Scan vulnerabilities list

The report page allows the user to upload data in the most readable form(.json), it was also taken into account that most often the internal security structure uses files of this type to select themin SIEM systems and scan signatures with machine code [10].



EternalSec

The prepared project from our team is targeted to Application security testing, but we do not want to frame yourself into SAST, or other AST. We want to present a product near to ASTO - Application security testing orchestration. Main goal of this product is to provide a comprehensive defense by scanning applications on vulnerabilities, starting from static scanning, and ending with other types of scanning. integration with software development life cycle gives needed efficiency of product. it also includes analytics of the projects.

2022
EternalSec AST Development Timeline

SCA Update July 2022

Software composition analysis (SCA) is an automated process that identifies the open source software in a codebase. This analysis is performed to evaluate security, license compliance, and code quality.

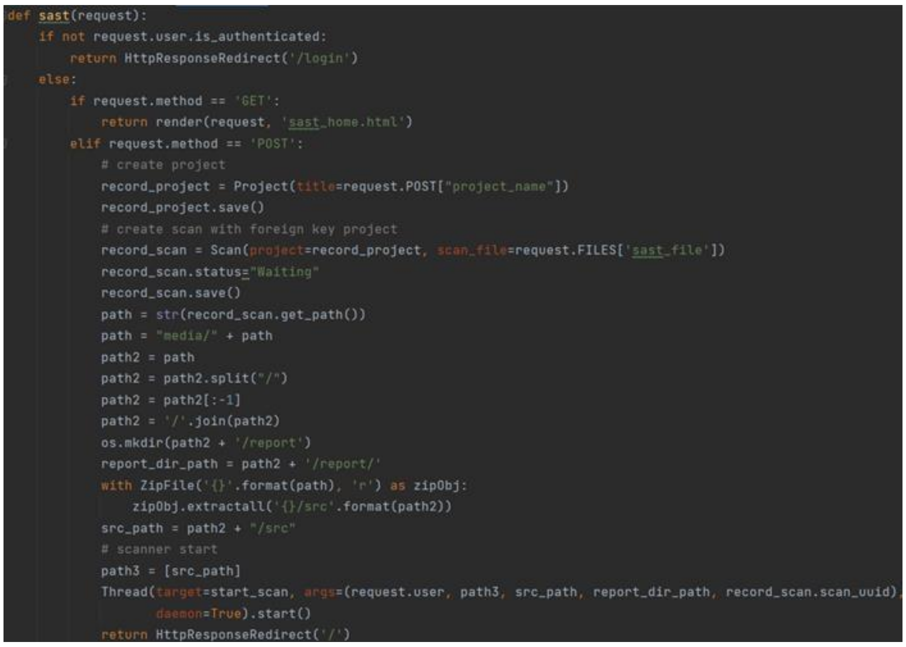
June 2022 DAST Update

Dynamic application security testing (DAST) is a black box testing method that examines an application as it's running to find vulnerabilities that an attacker could exploit.

SAST Update May 2022

Static application security testing (SAST) is used to secure software by reviewing the source code of the software to identify sources of vulnerabilities.

Figure 8: About



```
def sast(request):
    if not request.user.is_authenticated:
        return HttpResponseRedirect('/login')
    else:
        if request.method == 'GET':
            return render(request, 'sast_home.html')
        elif request.method == 'POST':
            # create project
            record_project = Project(title=request.POST["project_name"])
            record_project.save()
            # create scan with foreign key project
            record_scan = Scan(project=record_project, scan_file=request.FILES['sast_file'])
            record_scan.status="Waiting"
            record_scan.save()
            path = str(record_scan.get_path())
            path = "media/" + path
            path2 = path
            path2 = path2.split("/")
            path2 = path2[:-1]
            path2 = '/'.join(path2)
            os.mkdir(path2 + '/report')
            report_dir_path = path2 + '/report/'
            with ZipFile('{}'.format(path), 'r') as zipObj:
                zipObj.extractall('{}/src'.format(path2))
            src_path = path2 + "/src"
            # scanner start
            path3 = [src_path]
            Thread(target=start_scan, args=(request.user, path3, src_path, report_dir_path, record_scan.scan_uuid),
                   daemon=True).start()
            return HttpResponseRedirect('/')
```

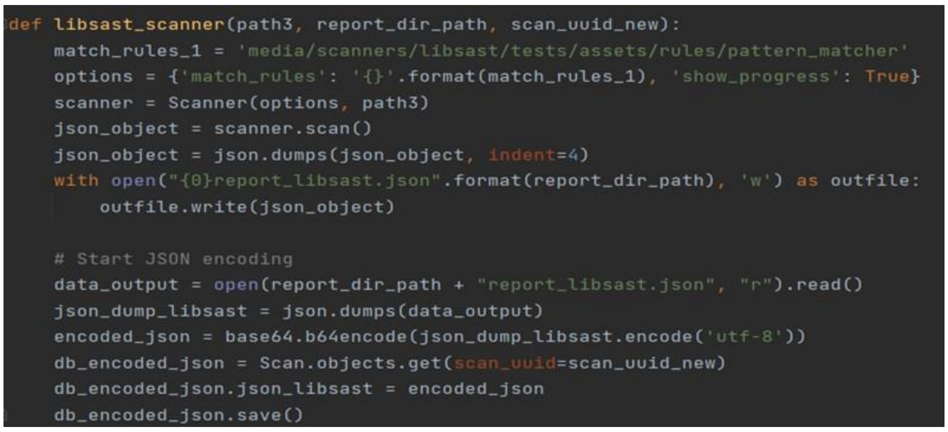
Figure 9: Start scan structure

The about section provides a timeline and general information about the beginning of product development, and planned updates, so the user is always aware of the current status of the product.

ASTO starts SAST scan by request of the user from the web interface. Users send source codeof file in ZIP, RAR format, with other information, for example, name, link to a git repository.

As a server, it takes this source code zip and adds it to our database field - “FileField”, and creates a new repository, with a new scan. The status function here needs to track the status of scanning to show it to the user interface. Using zip libraries, it unpacks the src path to see the sourcecode in the needed format.

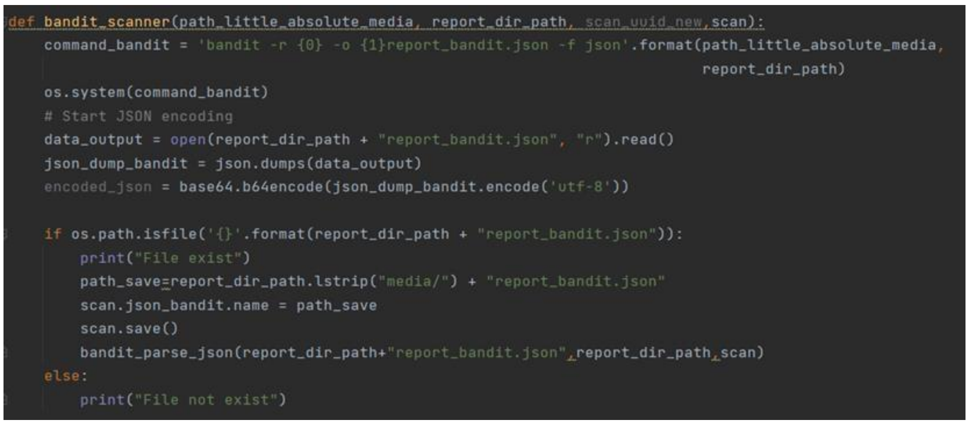
By using thread function, the application can do scanning function in the thread with the daemon, the process of scanning is in parallel, it needs technical requirements for that, but it gives the best efficiency of the functionality [11]. Libraries that helped to build the scanner were: Bandit, Libsast, Semgrep, JSON.



```python
def libsast_scanner(path3, report_dir_path, scan_uuid_new):
    match_rules_1 = 'media/scanners/libsast/tests/assets/rules/pattern_matcher'
    options = {'match_rules': '{}'.format(match_rules_1), 'show_progress': True}
    scanner = Scanner(options, path3)
    json_object = scanner.scan()
    json_object = json.dumps(json_object, indent=4)
    with open("{0}report_libsast.json".format(report_dir_path), 'w') as outfile:
        outfile.write(json_object)

    # Start JSON encoding
    data_output = open(report_dir_path + "report_libsast.json", "r").read()
    json_dump_libsast = json.dumps(data_output)
    encoded_json = base64.b64encode(json_dump_libsast.encode('utf-8'))
    db_encoded_json = Scan.objects.get(scan_uuid=scan_uuid_new)
    db_encoded_json.json_libsast = encoded_json
    db_encoded_json.save()
```

Figure 10: Libsast structure

Libsast library uses a pattern matcher to find vulnerabilities, this helps to scan any source code language and to increase functionality it uses python regex rule patterns to create custom patterns [12]. Manypopular scanners nowadays are based on the libsast library, for instance, njsscan, MobSF, etc. Results of libsast are based on a regex test of the given source code, it shows match lines, position, file path andresults can be fully customized by different options. It uses JSON format to output findings.

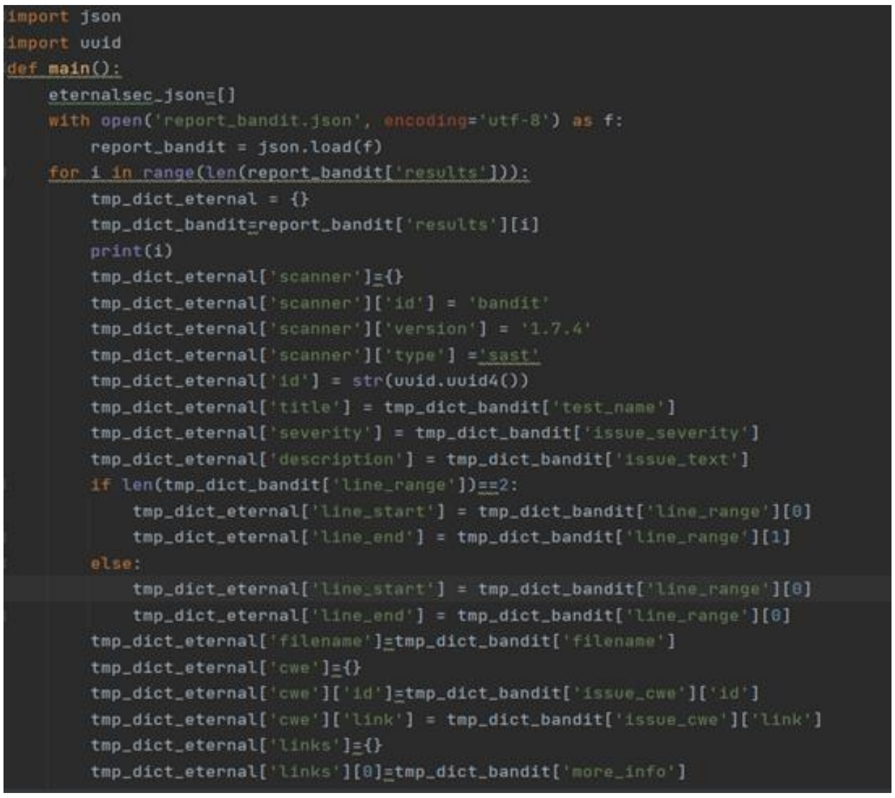


```python
def bandit_scanner(path_little_absolute_media, report_dir_path, scan_uuid_new,scan):
    command_bandit = 'bandit -r {0} -o {1}report_bandit.json -f json'.format(path_little_absolute_media,
                                                                              report_dir_path)
    os.system(command_bandit)
    # Start JSON encoding
    data_output = open(report_dir_path + "report_bandit.json", "r").read()
    json_dump_bandit = json.dumps(data_output)
    encoded_json = base64.b64encode(json_dump_bandit.encode('utf-8'))

    if os.path.isfile('{}'.format(report_dir_path + "report_bandit.json")):
        print("File exist")
        path_save=report_dir_path.lstrip("media/") + "report_bandit.json"
        scan.json_bandit.name = path_save
        scan.save()
        bandit_parse_json(report_dir_path+"report_bandit.json",report_dir_path,scan)
    else:
        print("File not exist")
```

Figure 11: Bandit structure

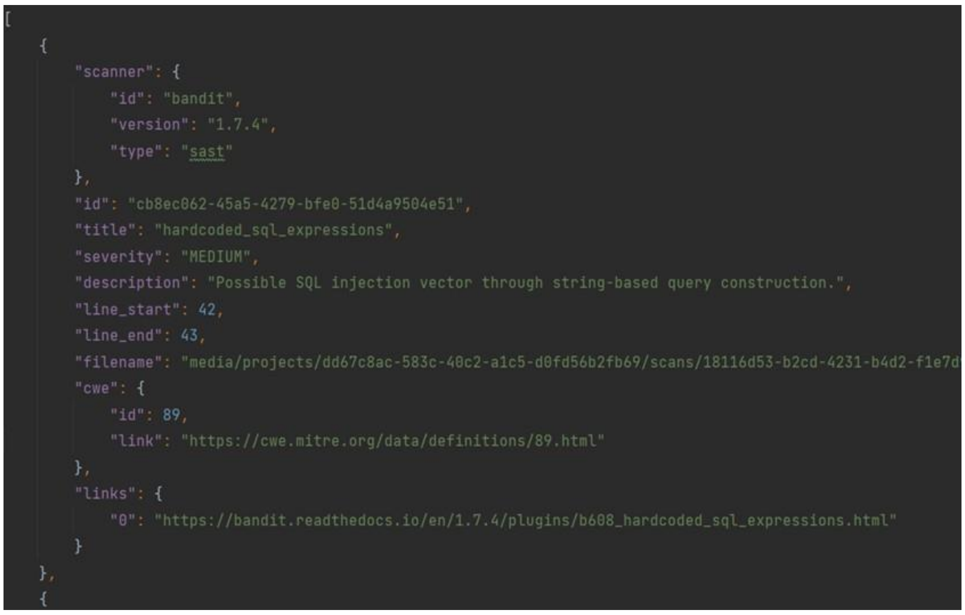
Bandit library finds security vulnerabilities only in Python code, but it can be modified to increase the number of supported languages. Using the os system package helps to start a scan and track the process, and check the conditions that you should pay attention to. Results of bandit are based on a test of abstract syntax trees of given source code, it shows match lines, position, file path, level of confidence and severity, etc. It uses JSON format to output findings. Users see results by webinterface in section vulnerabilities, and source code in the right grid with a window of code. Using section reports, users or analysts can download reports by format JSON from the response of request with the name of the scan in name of the file.

Parser eternalsec_json was developed to use results from libraries in the product, and to add new features.



```python
import json
import uuid
def main():
    eternalsec_json=[]
    with open('report_bandit.json', encoding='utf-8') as f:
        report_bandit = json.load(f)
    for i in range(len(report_bandit['results'])):
        tmp_dict_eternal = {}
        tmp_dict_bandit=report_bandit['results'][i]
        print(i)
        tmp_dict_eternal['scanner']={}
        tmp_dict_eternal['scanner']['id'] = 'bandit'
        tmp_dict_eternal['scanner']['version'] = '1.7.4'
        tmp_dict_eternal['scanner']['type'] ='sast'
        tmp_dict_eternal['id'] = str(uuid.uuid4())
        tmp_dict_eternal['title'] = tmp_dict_bandit['test_name']
        tmp_dict_eternal['severity'] = tmp_dict_bandit['issue_severity']
        tmp_dict_eternal['description'] = tmp_dict_bandit['issue_text']
        if len(tmp_dict_bandit['line_range'])==2:
            tmp_dict_eternal['line_start'] = tmp_dict_bandit['line_range'][0]
            tmp_dict_eternal['line_end'] = tmp_dict_bandit['line_range'][1]
        else:
            tmp_dict_eternal['line_start'] = tmp_dict_bandit['line_range'][0]
            tmp_dict_eternal['line_end'] = tmp_dict_bandit['line_range'][0]
        tmp_dict_eternal['filename']=tmp_dict_bandit['filename']
        tmp_dict_eternal['cwe']={}
        tmp_dict_eternal['cwe']['id']=tmp_dict_bandit['issue_cwe']['id']
        tmp_dict_eternal['cwe']['link'] = tmp_dict_bandit['issue_cwe']['link']
        tmp_dict_eternal['links']={}
        tmp_dict_eternal['links'][0]=tmp_dict_bandit['more_info']
```

Figure 12: Parser structure



```json
[
    {
        "scanner": {
            "id": "bandit",
            "version": "1.7.4",
            "type": "sast"
        },
        "id": "cb8ec062-45a5-4279-bfe0-51d4a9504e51",
        "title": "hardcoded_sql_expressions",
        "severity": "MEDIUM",
        "description": "Possible SQL injection vector through string-based query construction.",
        "line_start": 42,
        "line_end": 43,
        "filename": "media/projects/dd67c8ac-583c-40c2-a1c5-d0fd56b2fb69/scans/18116d53-b2cd-4231-b4d2-f1e7d
        "cwe": {
            "id": 89,
            "link": "https://cwe.mitre.org/data/definitions/89.html"
        },
        "links": {
            "0": "https://bandit.readthedocs.io/en/1.7.4/plugins/b608_hardcoded_sql_expressions.html"
        }
    },
    {
```

Figure 13: JSON structure

Vulnerability structure is shown in Figure 13. It is based on python dictionaries.

4. Conclusion

In conclusion, to summarize the development of the ASTO scientific work, a web-based information system, and functions were created that will scan applications for information security, show how to solve issues and vulnerabilities, and determine the level and criticality of the application and be complex to customers.

In the technical part, a lot of work has been done to develop this product, various frameworks have been used like Django, tools like various packages, and libraries like libsast, bandit, and semgrep. More than 2-3 thousand lines of code have been written. Own rules for static and semantic analysis were written without being tied to a specific programming language, which makes this product cross-platform and allows it to be deployed both on Windows and Linux. The web interface also complies with all modern UI/UX design standards, providing a pleasant user experience and rich functionality. Existing resources have been extended by adding the necessary checks for their functionality.

This enterprise-level product is recommended for use in security departments of banks, regulators, as well as in integrator companies and software developers. Since this product is a modular orchestrator, it will be easy to customize it to the needs of the company, as well as integrate it into any stage of the development process.

As a result, we have a product that has assembled all the necessary things to ensure the information security of an application, and which economic efficiency is very positive. In the future, this product can compete with the current market leaders.

5. References

[1] URL: https://www.vaadata.com/blog/pentest-statistics-and-most-frequent-vulnerabilities/.

[2] Ranking.kz, Information security incident statistics, January 2021. URL: http://ranking.kz/ru/a/infopovody/kolichestvo-kiberatak-v-kazahstane-uvelichilos-pochti-v-3-raza-do- 3-tysyach-incidentov-90-iz-nih-prihoditsya-na-botnety.

[3] Imperva, Application Security Testing, 2021. URL: https://www.imperva.com/learn/application-security/application-security-testing/#:~:text=Application%20security%20testing%20(AST)%20is,started%20as%20a%20manual%20process.

[4] J. Garbajosa, X. Wang, Myths and Facts About Static Application Security Testing Tools: An Action Research at Telenor Digital (2018) 24-48.

[5] P. Johnson, What You Need To Know About Application Security Testing Orchestration, December 10, 2020. URL: https://www.mend.io/resources/blog/asto-application- security-testing-orchestration/.

[6] C. Nabe, Impact of COVID-19 on Cybersecurity, 2021. URL: https://www2.deloitte.com/ch/en/pages/risk/articles/impact-covid-cybersecurity.html.

[7] PT Application Inspector, official description. URL: https://www.ptsecurity.com/ru-ru/products/ai/.

[8] SonarQube, official description. URL: https://www.sonarqube.org/.

[9] Solar appScreener, official description. URL: https://rt-solar.ru/products/solar_appscreener/.

[10] HCL AppScan, official description. URL: https://www.hcltech.com/brochures/software/hcl-appscan-standard.

[11] Profit.kz, The number of cyberattacks in the Republic of Kazakhstan increased by 20% over the year, 8 July 2021. URL: https://profit.kz/news/61600/Kolichestvo-kiberatak-v-RK-viroslo-na-20-za-god/.

[12] ZAO Deloitte & Touche CIS, Cyber risk assessment in banks of Kazakhstan by Deloitte on forum "Making an Impact That Matters, June 2021.