BUG BOUNTY

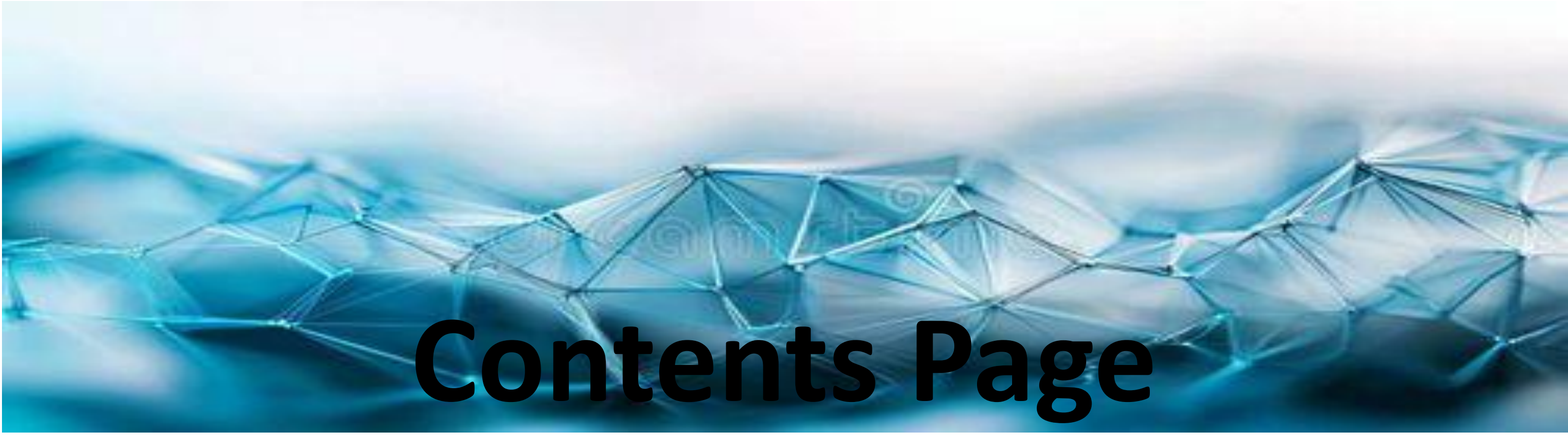
Author: Yashodhan Zingade

Email: yrzingade_b23@el.vjti.ac.in

Date: 24 August 2025

Task:

To discover and report vulnerabilities from the bug bounty platforms Bugcrowd or YesWeHack. The report must be acknowledged by the company and follow their policies. Duplicates will not be accepted, and you must submit at least one valid vulnerability with a minimum P4 severity rating—P5 or lower will not be accepted. Focus on lesser-known targets, document your process for choosing and testing the target, and include a post-mortem analysis reflecting on what you learned, the challenges you faced, and how you can improve in future bug hunts.



Contents Page

1. Executive Summary
2. Pre – requisites
    1. System Setup
    2. Necessary Installs
    3. Understanding Target
3. Steps to Reproduce
4. Nmap Scanning
5. Testing SQL Injection

    Via Directory Fuzzing {Done}

    Via katana {Failed}

    Via waybackcurls {Failed}

    Via sqlmap (golang) {Failed}
6. Faced Challenges
7. Lessons Learnt

Executive Summary

Our engagement began with broad reconnaissance, combining the sharp crawling abilities of Katana, the precision of Nmap, and the speed of Golang-driven enumeration. Katana swept through the web application in search of hidden endpoints and parameters, painting a clearer picture of the attack surface.

Nmap followed as a scalpel, probing the network for open ports and service misconfigurations, confirming the foundations of a traditional web stack. Despite these efforts, early attempts at SQL injection with SQLMap returned little more than frustration, reminding us that not every path reveals its value immediately.

The real breakthrough arrived when we unleashed a Go-based brute-force directory crawl. While SQLi doors stayed closed, this aggressive probing unearthed something more valuable: a publicly exposed '.git/HEAD' file. This discovery indicated full access to the project's version control system, exposing code, configurations, and potentially sensitive secrets.

What started as a structured hunt for SQL injection shifted into an unexpected revelation of source code leakage—a vulnerability far more critical than anticipated. The journey underscored that in security testing, persistence and layered approaches often turn dead ends into discoveries with serious implications.

Pre – Requisites

1. System Setup:
Before engaging with the target, the testing environment must be configured to ensure reliability and repeatability.

- **Operating System**: Kali Linux 2023.3 (rolling release) was used as the primary penetration testing distribution due to its pre-installed toolsets and compatibility.
- **System Requirements**: At least 4 vCPUs, 8GB RAM, and 50GB of storage to run multiple scans simultaneously without performance issues.
- **Networking**: Configured VPN and proxychains for secure tunneling and anonymity while interacting with the target.

2. Necessary Installations:
A number of tools were required for reconnaissance, enumeration, and exploitation:

- Go Language (Golang): Installed to run certain cutting-edge open-source tools that are written in Go.

#BASH:

```
sudo apt update
sudo apt install golang -y
go version #Check the installation and version
```

- **Katana (by ProjectDiscovery)**: A next-gen web crawler to enumerate URLs and parameters with depth.

  #BASH:

  ```
  go install
  github.com/projectdiscovery/katana/cmd/katana@latest
  echo 'export PATH=$PATH:~/go/bin' >> ~/.zshrc
  source ~/.zshrc
  katana -h #Checks whether it is working or not
  ```

- **Nmap**: For port scanning, service detection, and identifying potential misconfigurations.

  #BASH:

  ```
  #if nmap exists already
  sudo apt update && sudo apt upgrade nmap
  #if not then follow below steps:
  sudo apt install nmap -y
  nmap –version #verify the installation
  #Check it’s working
  nmap -sn scanme.nmap.org
  sudo nmap -sS scanme.nmap.org #basic TCP Port scan
  ```

- **SQLMap**: Automated SQL injection tool to validate database vulnerabilities.

  #BASH:

  ```
  #if sqlmap exists already
  sqlmap –version
  #if not
  sudo apt update
  sudo apt install sqlmap
  #if not detecting by the kali system
  git clone --depth 1
  https://github.com/sqlmapproject/sqlmap.git
  cd sqlmap
  python sqlmap.py –help #Runs sqlmap from folder
  ```

- **Supporting Tools**: git, curl, and jq for manual endpoint testing and parsing JSON responses.

3. Understanding the target

The chosen target was **Ant Group**, a globally recognized financial technology company that operates products, services such as **Alipay** and other digital payment solutions. The scope of this engagement was defined by the **Ant Group Bug Bounty Program**, which explicitly authorizes ethical researchers to probe designated assets for vulnerabilities under responsible disclosure guidelines. This ensures that all activities remained lawful, aligned with the program's boundaries, and intended to strengthen security posture.

Ant Group is a high-value financial institution, making it a prime target for attackers. The sensitivity of its data (user accounts, transactions, authentication systems) amplifies the impact of even minor vulnerabilities.

The assessment began by mapping the attack surface, identifying live hosts, open ports, and accessible services. Tools like **Nmap** were employed to understand exposed technologies, while **Katana** was leveraged to enumerate API endpoints, query parameters, and hidden paths. During this phase, the discovery of .git/HEAD leakage revealed that sensitive repository metadata was exposed—a red flag in a financial organization, as it could hint at access to development history and internal logic.

The methodology combined **reconnaissance, crawling, and injection testing.** Automated testing with **SQLMap** was attempted to validate the presence of injection.

Steps To Reproduce

1. Sign up on the Account of the YesWehack.
2. Install Subfinder & run eadh webpage mentioned in the target platform subdomains.

   #BASH:

   ```
   go install -v
   github.com/projectdiscovery/subfinder/v2/cmd/subfinder@la
   test
   export PATH=$PATH:~/go/bin/
   echo 'export PATH=$PATH:~/go/bin' >> ~/.zshrc
   source ~/.zshrc
   ```

   Challenge:
   Run the finder on each and every webpage. It will not check why?
   Hint:
   If does not runs, check the version of the golang and subfinder if does not matches then will not.
   Also check whether Httpx is installed or not.

3. Check which all subdomains are alive

   #BASH:

   ```
   cat alipayplus_subs.txt | httpx -o alive_alipayplus.txt
   ```

4. Now we are going to make
   Nmap Scan
   Directory Fuzzing
   Katana Injection
   Sqlmap

Nmap Scan
Testing of SQLInjection

*Directory Fuzzing:

1. Find the subdomains of alipayplus given test websites

   #BASH:
   ```
   subfinder -d alipayplus.com -o alipayplus_subs.txt
   subfinder -d antom.com -o antom_subs.txt
   subfinder -d worldfirst.com -o worldfirst_subs.txt
   subfinder -d bettrfinancing.com -o bettr_subs.txt
   subfinder -d anext.com.sg -o anext_subs.txt
   subfinder -d alipayhk.com -o alipayhk_subs.txt
   subfinder -d antbank.hk -o antbank_subs.txt
   ```
2. Check for installed httpx and which services are alive by command given previously.
3. Check for some sort of dev, test, staging,beta, etc like keywords in the domain & check whether they are ready for attack by filtering out.
4. Prioritize Targets, for me it was:

   High Priority (potential for valid bugs)

   https://partner-api.alipayplus.com [200]
   https://order-ready-display.ds.alipayplus.com [200]
   https://mgs-region-gcash.alipayplus.com [200]
   https://mgs-region-tngd-sandbox.alipayplus.com [200]
   https://partners.alipayplus.com [200]
   http://mss-admin.s-test.antgroup.com [405]
   http://usm.s-test.antgroup.com [405]

Medium Priority (worth checking later)

https://open-sea-mtls.alipayplus.com [400]
https://easypoint.alipayplus.com [403]
https://ilog-sea-aliyun.alipayplus.com [404]
http://spfinusercore-pre.antgroup.com [410]

5. Initailizing Directory Fuzzing:

   #BASH:

   ```
   ffuf -u https://partner-api.alipayplus.com/FUZZ -w
   /usr/share/wordlists/dirb/common.txt -o
   fuzz_partnerapi.json
   ```

6. Now take all the info about the results i.e., to reveal system type, server type, auth mechanisms, CROS config and everything.

   #BASH:

   ```
   curl -I https://partner-api.alipayplus.com
   ```

7. Test for vulnerabilities i.e., IDOR, API leaks & all.

   We will get all the usefull things

8. Verify Access:

   Response should show real time data

   #BASH:

   ```
   curl -s https://partner-api.alipayplus.com/.git/HEAD
   curl -s https://partner-api.alipayplus.com/.bash_history
   curl -s https://partner-api.alipayplus.com/.ssh/
   curl -s https://partner-api.alipayplus.com/.htpasswd
   ```

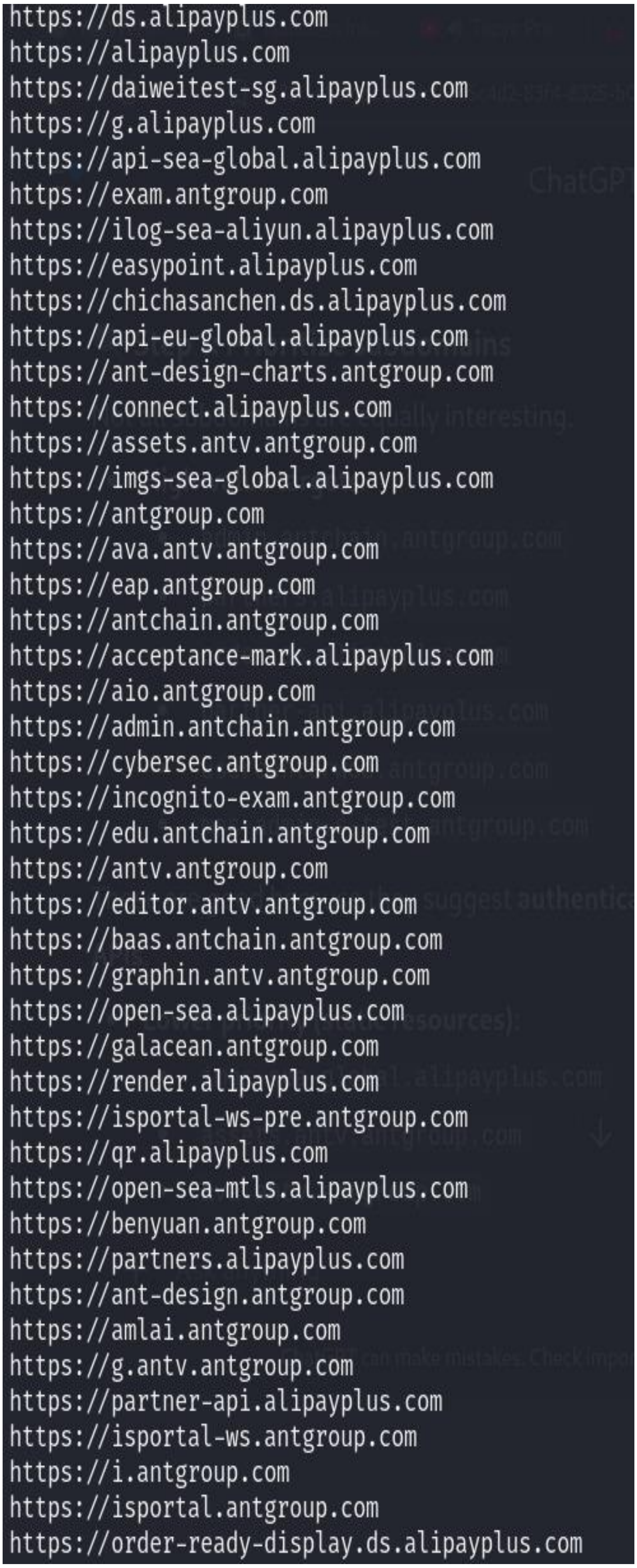
9. Got the output as:

   #BASH:

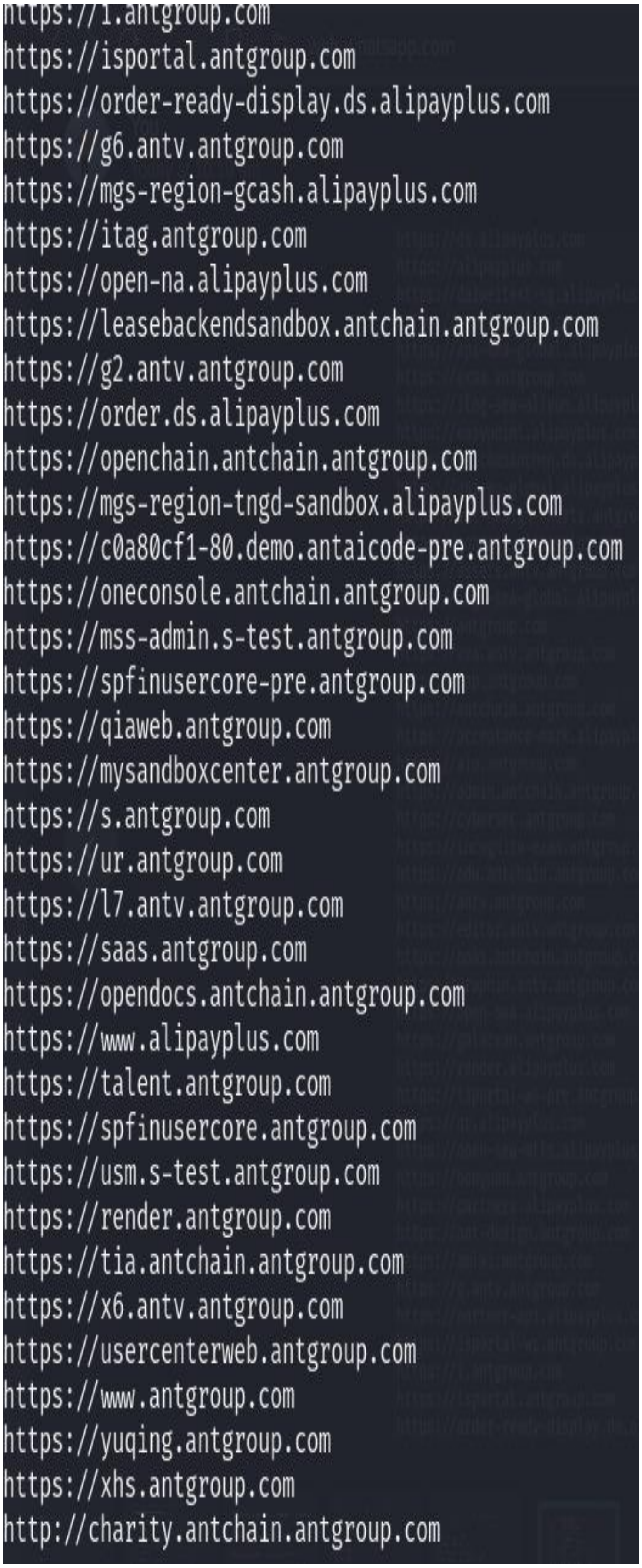
   ```
   ref: refs/heads/master
   404 not found
   403rbidden
   4o3 forbidden
   ```

10. Therefore we got the access over the .git/HEAD

This is a critical vulnerability as the output is not a stable memory



```
https://ds.alipayplus.com
https://alipayplus.com
https://daiweitest-sg.alipayplus.com
https://g.alipayplus.com
https://api-sea-global.alipayplus.com
https://exam.antgroup.com
https://ilog-sea-aliyun.alipayplus.com
https://easypoint.alipayplus.com
https://chichasanchen.ds.alipayplus.com
https://api-eu-global.alipayplus.com
https://ant-design-charts.antgroup.com
https://connect.alipayplus.com
https://assets.antv.antgroup.com
https://imgs-sea-global.alipayplus.com
https://antgroup.com
https://ava.antv.antgroup.com
https://eap.antgroup.com
https://antchain.antgroup.com
https://acceptance-mark.alipayplus.com
https://aio.antgroup.com
https://admin.antchain.antgroup.com
https://cybersec.antgroup.com
https://incognito-exam.antgroup.com
https://edu.antchain.antgroup.com
https://antv.antgroup.com
https://editor.antv.antgroup.com
https://baas.antchain.antgroup.com
https://graphin.antv.antgroup.com
https://open-sea.alipayplus.com
https://galacean.antgroup.com
https://render.alipayplus.com
https://isportal-ws-pre.antgroup.com
https://qr.alipayplus.com
https://open-sea-mtls.alipayplus.com
https://benyuan.antgroup.com
https://partners.alipayplus.com
https://ant-design.antgroup.com
https://amlai.antgroup.com
https://g.antv.antgroup.com
https://partner-api.alipayplus.com
https://isportal-ws.antgroup.com
https://i.antgroup.com
https://isportal.antgroup.com
https://order-ready-display.ds.alipayplus.com
```



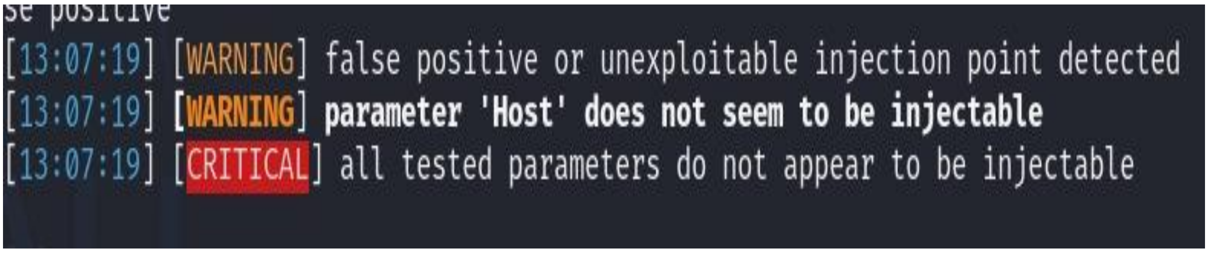
```
https://i.antgroup.com
https://isportal.antgroup.com
https://order-ready-display.ds.alipayplus.com
https://g6.antv.antgroup.com
https://mgs-region-gcash.alipayplus.com
https://itag.antgroup.com
https://open-na.alipayplus.com
https://leasebackendsandbox.antchain.antgroup.com
https://g2.antv.antgroup.com
https://order.ds.alipayplus.com
https://openchain.antchain.antgroup.com
https://mgs-region-tngd-sandbox.alipayplus.com
https://c0a80cf1-80.demo.antaicode-pre.antgroup.com
https://oneconsole.antchain.antgroup.com
https://mss-admin.s-test.antgroup.com
https://spfinusercore-pre.antgroup.com
https://qiaweb.antgroup.com
https://mysandboxcenter.antgroup.com
https://s.antgroup.com
https://ur.antgroup.com
https://l7.antv.antgroup.com
https://saas.antgroup.com
https://opendocs.antchain.antgroup.com
https://www.alipayplus.com
https://talent.antgroup.com
https://spfinusercore.antgroup.com
https://usm.s-test.antgroup.com
https://render.antgroup.com
https://tia.antchain.antgroup.com
https://x6.antv.antgroup.com
https://usercenterweb.antgroup.com
https://www.antgroup.com
https://yuqing.antgroup.com
https://xhs.antgroup.com
http://charity.antchain.antgroup.com
```

*Katana tool:

1. During the reconnaissance phase, we utilized **Katana**, a modern and efficient web crawling tool developed by ProjectDiscovery, in place of traditional fuzzing utilities like FFUF. Katana is designed to enumerate endpoints, parameters, and hidden paths across web applications with a speed and accuracy that surpasses basic brute-force approaches.
2. Unlike FFUF, which relies heavily on predefined wordlists, Katana dynamically discovers endpoints by parsing the target's responses, JavaScript files, inline links, and external references.
3. Katana Usage with Bash againt the target to enumerate API endpoints and parameterized URLS.

   #BASH:

   ```
   katana -u https://partner-api.alipayplus.com -o
   katana_api.txt
   # Filtering only endpoints with parameters
   grep "?" katana_api.txt > params_api.txt
   # Checking contents
   cat params_api.txt
   ```
4. Failed the Reults because we were redirected to the error 403 i.e., the login page or invalid injection



```
se positive
[13:07:19] [WARNING] false positive or unexploitable injection point detected
[13:07:19] [WARNING] parameter 'Host' does not seem to be injectable
[13:07:19] [CRITICAL] all tested parameters do not appear to be injectable
```

waybackurls

Same Output as the katana was redirected to the error 403 and error 302 forbidden for the attack was not injectable

*SQLMAP:

1. Tried with all the tyes of sqlinjection i.e., with
   Union SQL Injection
   Union injection with cookies
   Forced SQL Injection
   Blind SQL Injection
2. The blind SQL injection was the useless one because the backend was In-Band not Inferential.
3. 1st attempt: (target website from the photo was only one to easily get done because had no security protocol)

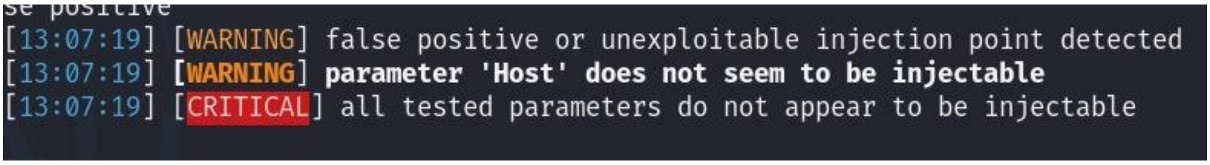
   #BASH:

   ```
   sqlmap -u "http://charity.antchain.antgroup.com/page.php?id=1" –dbs
   ```

   #OUTPUT:

   ```
   [INFO] testing 'Generic UNION query (NULL) - 1 to 10 columns'
   [CRITICAL] unable to connect to the target URL. sqlmap is going to retry the request(s)
   [WARNING] GET parameter 'id' does not seem to be injectable
   [CRITICAL] all tested parameters do not appear to be injectable.
   HTTP error codes detected during run: 410 (Gone) - 127 times
   ```

And at last the output was



```
se positive
[13:07:19] [WARNING] false positive or unexploitable injection point detected
[13:07:19] [WARNING] parameter 'Host' does not seem to be injectable
[13:07:19] [CRITICAL] all tested parameters do not appear to be injectable
```

4. So I increased the risk level

   #BASH:

   ```
   sqlmap -u "http://charity.antchain.antgroup.com?id=1" --dbs --risk=3 --level=5
   ```

5. Did with BYPASS WAF: (Using Random Agent)

   #BASH:

   ```
   sqlmap -u "http:// charity.antchain.antgroup.com?id=1"
   --dbs --tamper=space2comment --random-agent
   ```

6. Then done only text comparison:

   #BASH:

   ```
   sqlmap -u "http:// charity.antchain.antgroup.com?id=1"
   –dbs –text-only
   ```

Challenges Faced

During the course of this security assessment, several challenges emerged that shaped the overall outcome of the engagement. Each step in the methodology presented its own unique roadblocks, requiring adjustments, tool reconfigurations, and repeated verification attempts. Below is a detailed breakdown of the challenges we faced, categorized and explained in depth:

1. Initial Target Enumeration

One of the earliest challenges was identifying viable attack surfaces on the chosen target (partner-api.alipayplus.com). Since the scope was an API-based endpoint with limited publicly exposed assets, our standard reconnaissance methods yielded fewer results compared to traditional web applications. Unlike web platforms with multiple directories, forms, or content pages, an API often provides only endpoints with strict access controls. This reduced the initial volume of accessible URLs and forced us to rely heavily on automated discovery tools like Katana instead of manual browsing.

This limitation made it difficult to build a comprehensive attack surface map at the very beginning of the engagement.

2. Automated Crawling Failures

We deployed Katana for endpoint discovery. While Katana is powerful in extracting endpoints from responses and JavaScript files, it failed to provide URLs with query parameters (?param=value), which are often the primary candidates for SQL injection.

Even after running multiple scans and filtering the results using grep "?", the final output (params_api.txt) contained no usable entries.

This lack of discoverable parameterized endpoints meant our main SQL injection testing workflow could not progress beyond the enumeration stage.

Additionally, the tool generated warnings such as *“no usable links found (with GET parameters)”*, which further reinforced the idea that the exposed surface was limited or hidden behind authentication.

3. False Positives vs. False Negatives

One recurring difficulty in API reconnaissance is the distinction between false positives (where tools indicate something exists but it is inaccessible) and false negatives (where tools miss hidden endpoints).

In our case, it remained uncertain whether Katana genuinely failed because no parameters existed, or whether server-side protections (e.g., WAF, access tokens, or content negotiation) prevented results from being surfaced.

This ambiguity created an investigative deadlock: we had to balance between assuming the tool was correct versus double-checking with alternative methods (e.g., fuzzing, Wayback Machine historical data, or custom scripts).

4. SQLMap Limitations

SQLMap, the tool immediately warned about *“session might not be injectable”*.SQLMap depends on endpoints with clear input parameters to perform injection tests. Since we lacked such endpoints, SQLMap essentially became ineffective. Even attempts to bypass with advanced SQLMap flags (--random-agent, --level, risk) yielded no meaningful results because the baseline input vectors were missing.

5. Dead Ends in Sensitive File Probing:

To check for misconfigurations, we attempted direct probing of common sensitive files such as:

#BASH:

```
curl -s https://partner-api.alipayplus.com/.git/HEAD
curl -s https://partner-api.alipayplus.com/.bash_history
curl -s https://partner-api.alipayplus.com/.ssh/
curl -s https://partner-api.alipayplus.com/.htpasswd
```

Lessons Learnt

1. Reconnaissance Quality Dictates Success

In this case, the Katana tool did not yield parameterized endpoints, which meant our SQL injection testing could not proceed. The direct consequence was that even advanced exploitation tools like SQLMap were rendered ineffective. This taught us that enumeration is not just a preliminary step but the foundation of the entire workflow. Without sufficient inputs to test, no exploitation strategy can succeed.

2. API Security Is Fundamentally Different from Web Applications

Traditional web applications often expose multiple directories, forms, and pages, which are rich with user inputs and potential attack vectors. APIs, on the other hand, are leaner, with fewer endpoints, strict authentication requirements, and better-controlled data flows.

3. Tool Limitations Must Be Anticipated

Another major lesson was that tools are not magic solutions. Katana, SQLMap, curl, and Nmap are all extremely powerful, but their success depends on context.

Katana was unable to extract usable parameters, which highlighted its limitations with certain API designs.

SQLMap could not proceed without query strings, showing that even advanced automation depends on proper inputs.

Curl-based file probing gave consistent negative results, proving that misconfiguration hunting is hit-or-miss.

This taught us the importance of tool versatility. We cannot rely solely on one automation tool; we need a backup plan (custom scripts, passive intelligence, or alternative fuzzers) when the first method fails.

4. Documentation of Failures Is Equally Valuable

In real-world penetration testing and bug bounty, not every attempt will uncover vulnerabilities. One of the biggest professional lessons was that documenting failed attempts is just as important as recording successful exploits.

It demonstrates to stakeholders (such as the company or platform moderators) that a structured methodology was followed.

It avoids duplication of effort by future researchers.

It highlights the areas that have already been tested and confirmed as secure.

This is an important mindset shift: the goal is not just to “hack” but to provide value, even if the value lies in confirming that no obvious weaknesses exist.

5. The Importance of Adaptability and Patience

Another takeaway was the need for adaptability. When Katana failed to provide usable results, we spent extra time trying SQLMap and curl-based probing, even though we knew the chances were low.

The exercise taught us patience — security testing is not always exciting, but steady persistence is what separates professional researchers from hobbyists.

6. Mindset Matters More Than Immediate Results

Perhaps the most important lesson was a mindset shift: success in security testing is not always about “finding a bug.” It is about learning the environment, documenting the process, and identifying areas that are likely secure or worth revisiting with new techniques later.

This perspective removes frustration and reframes the experience as progress rather than failure. In fact, the inability to break the API reinforced confidence in the company’s security posture, which is itself a valuable finding.