**Lab Objective: Perform SQL injection on http://testphp.vulnweb.com/ by using Havij Tool.**

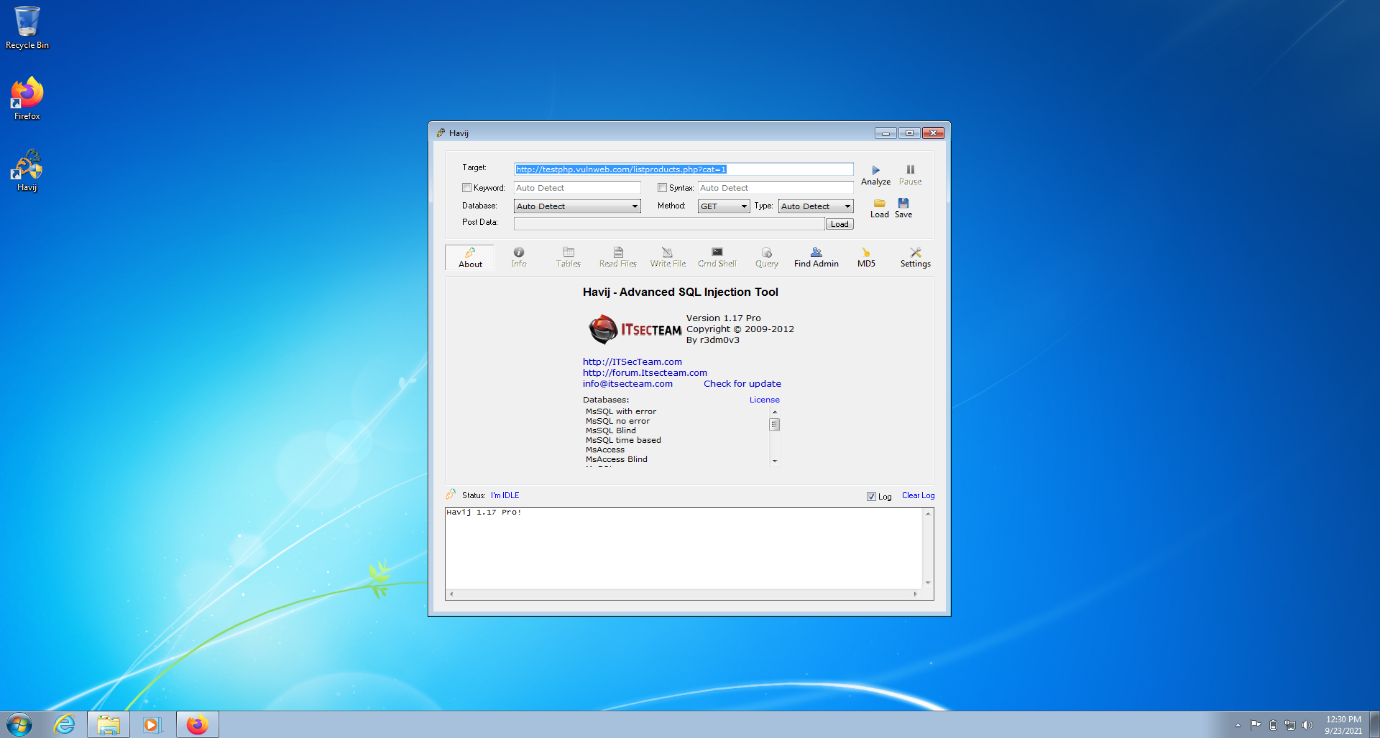
**Equipments Used:**

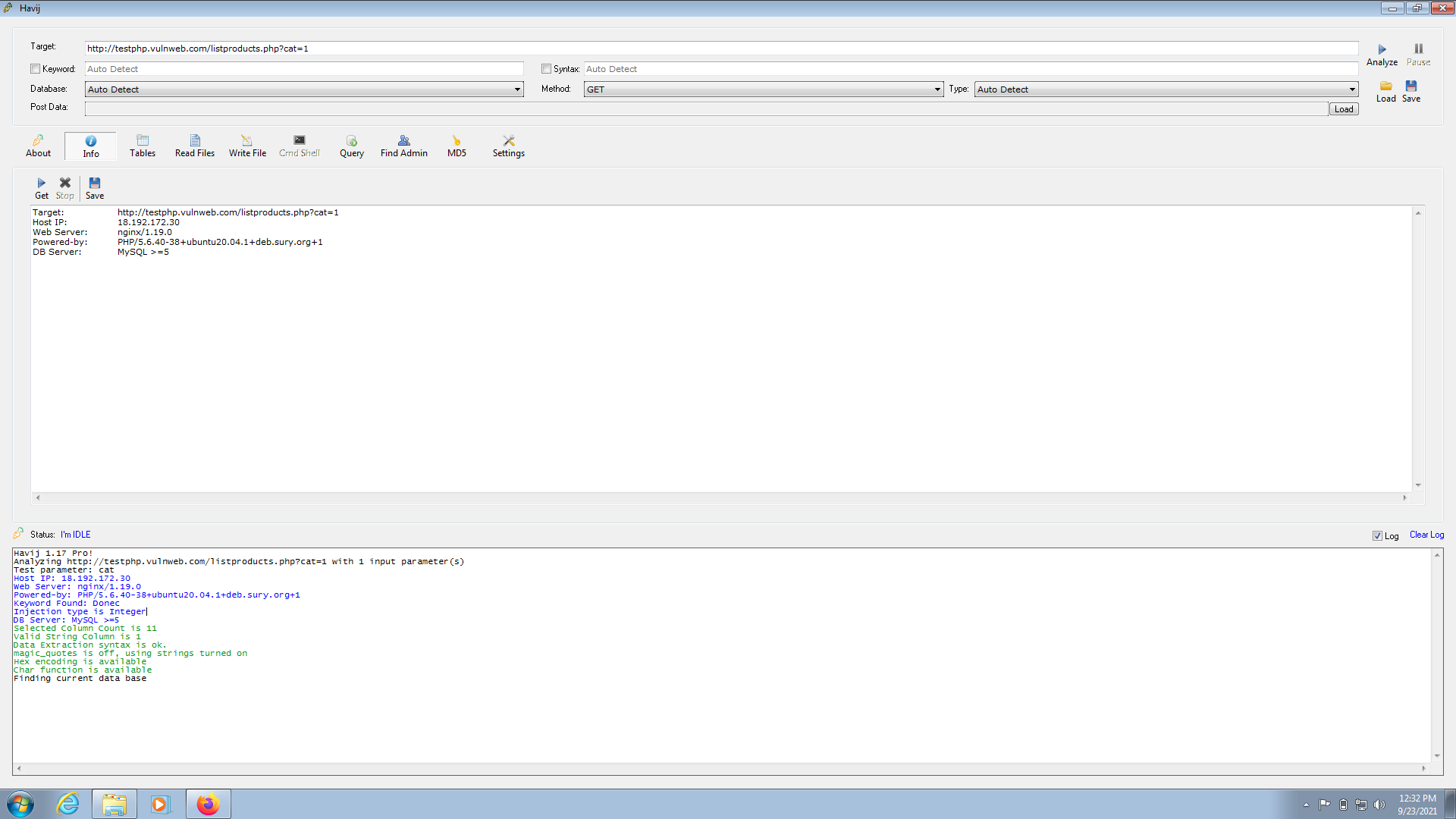
**Havij Tool**

Havij is an automated SQL Injection tool that helps **penetration testers** to find and exploit SQL Injection vulnerabilities on a web page. It can take advantage of a vulnerable web application. By using this software, user can perform back-end database fingerprinting, retrieve DBMS login names and password hashes, dump tables and columns, fetch data from the database, execute SQL statements against the server, and even access the underlying file system and execute operating system shell commands.

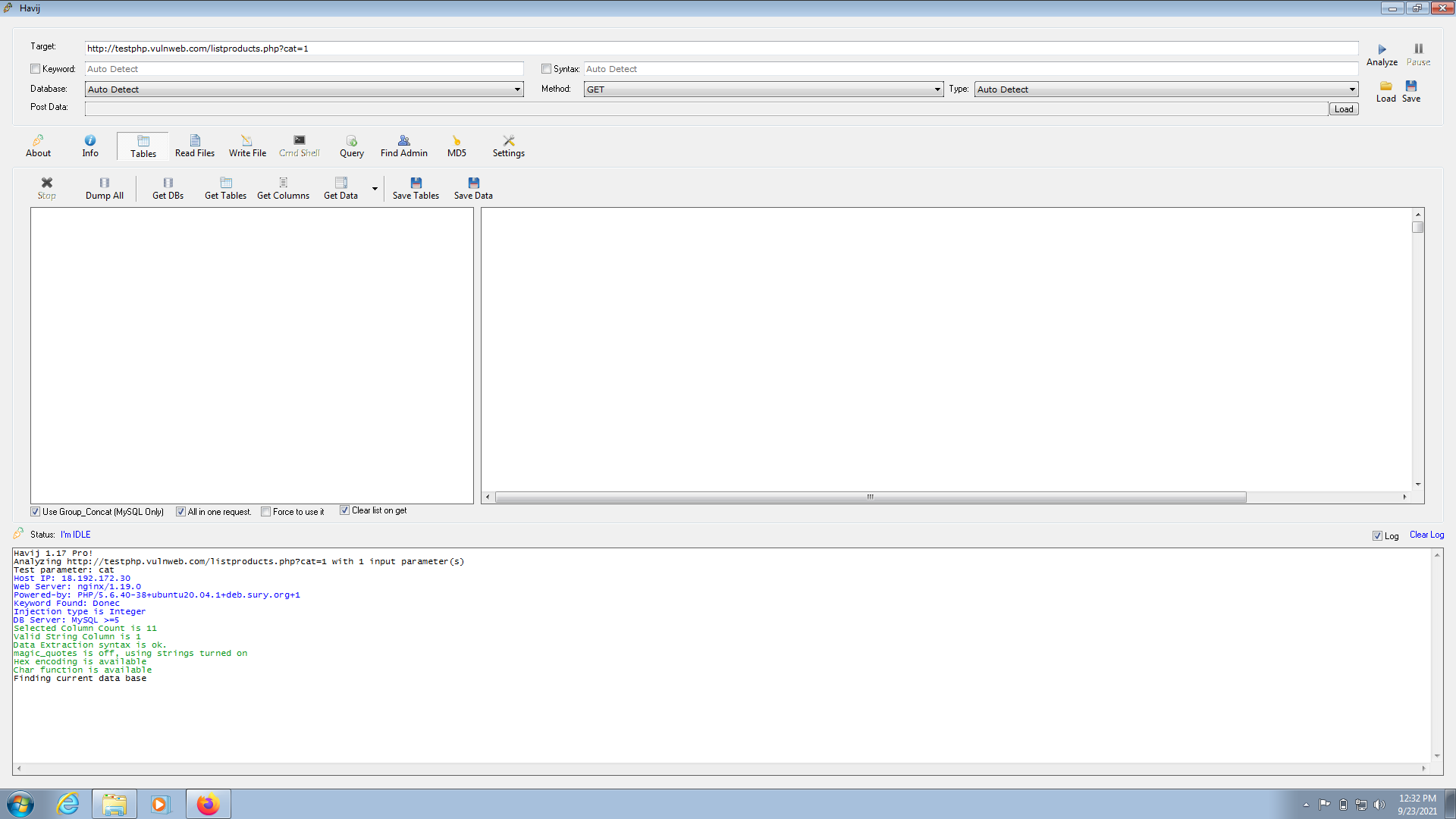
**Procedure:**

1. Download Havij from Google and install it. Then open it and enter the vulnerable page url in the target column

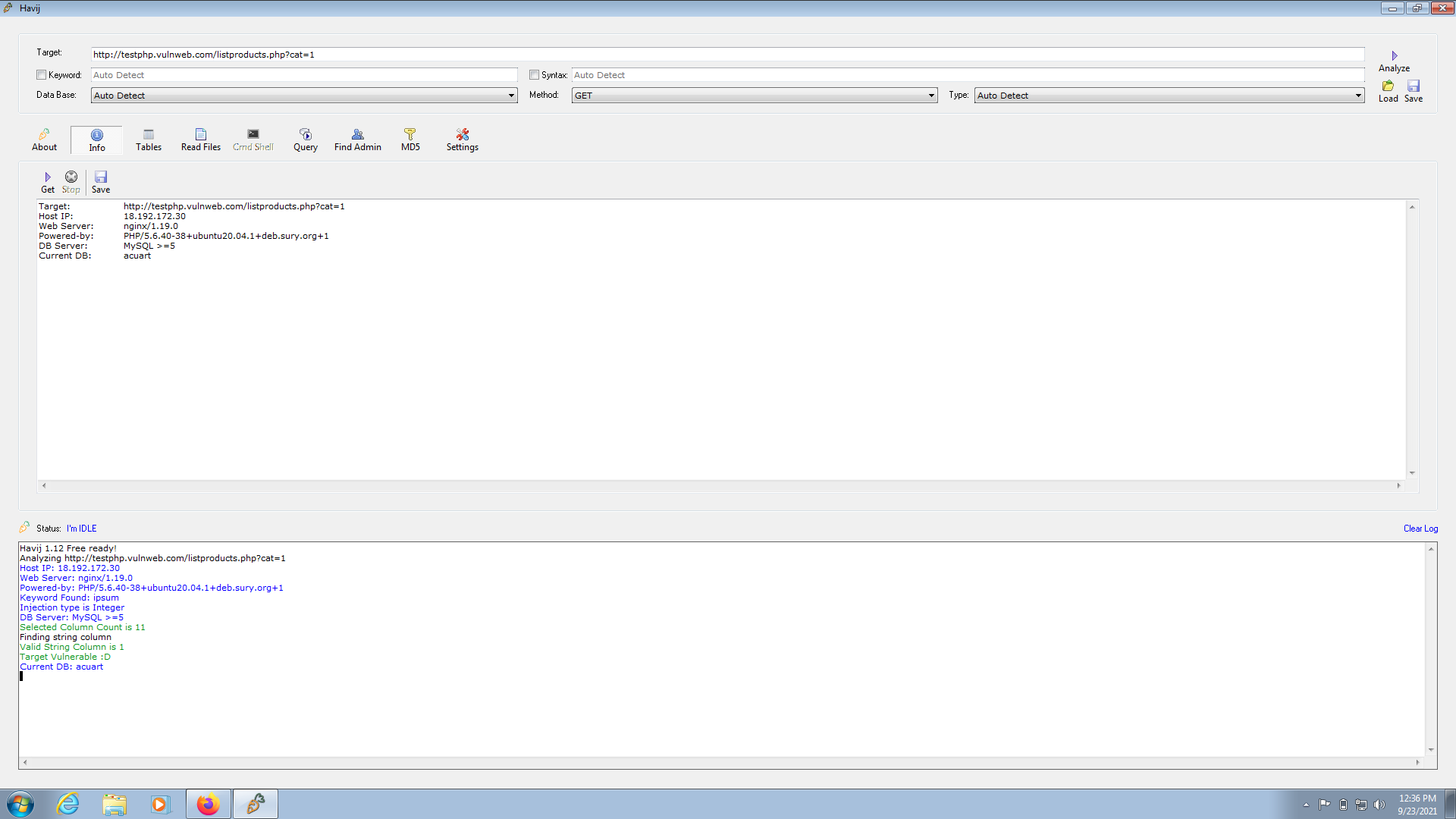
****

****

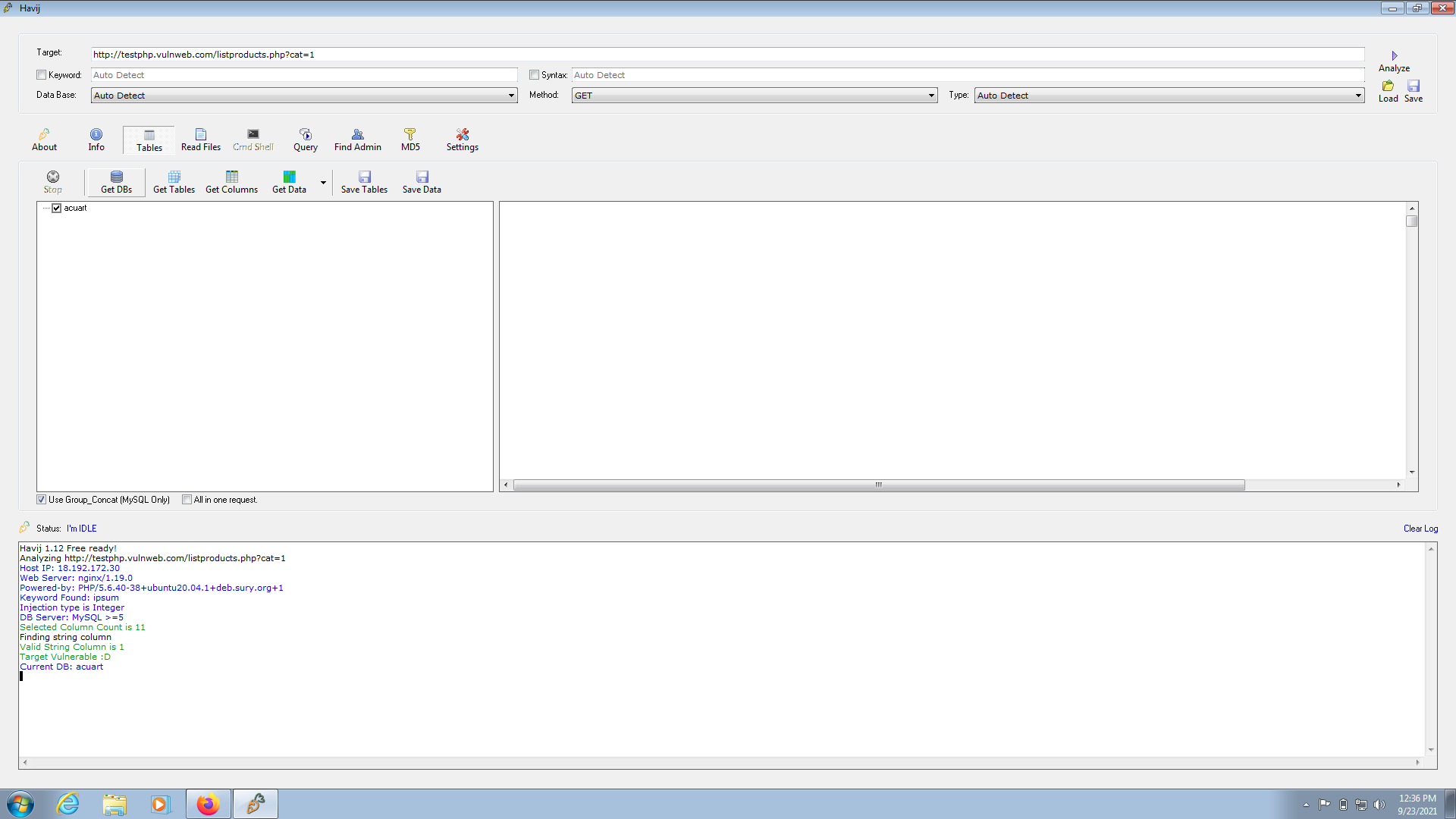
1. Set the database option to ‘auto detect‘ and hit analyze. This should show you the current database name as shown below.

****

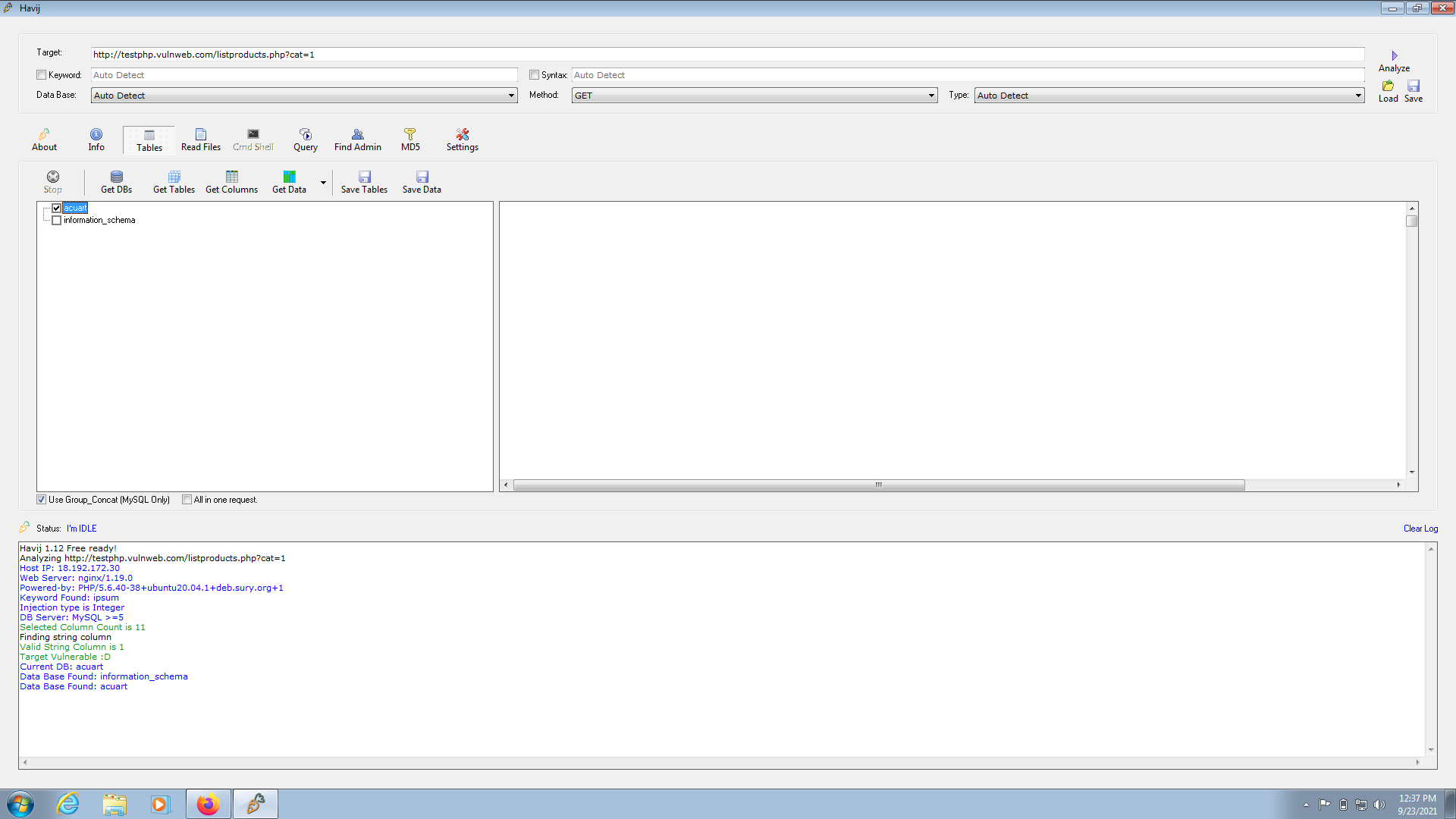
1. Click on the “***info”*** tab. This will show you information about the victim’s system. We can see information like Host IP address, web server version etc.

****

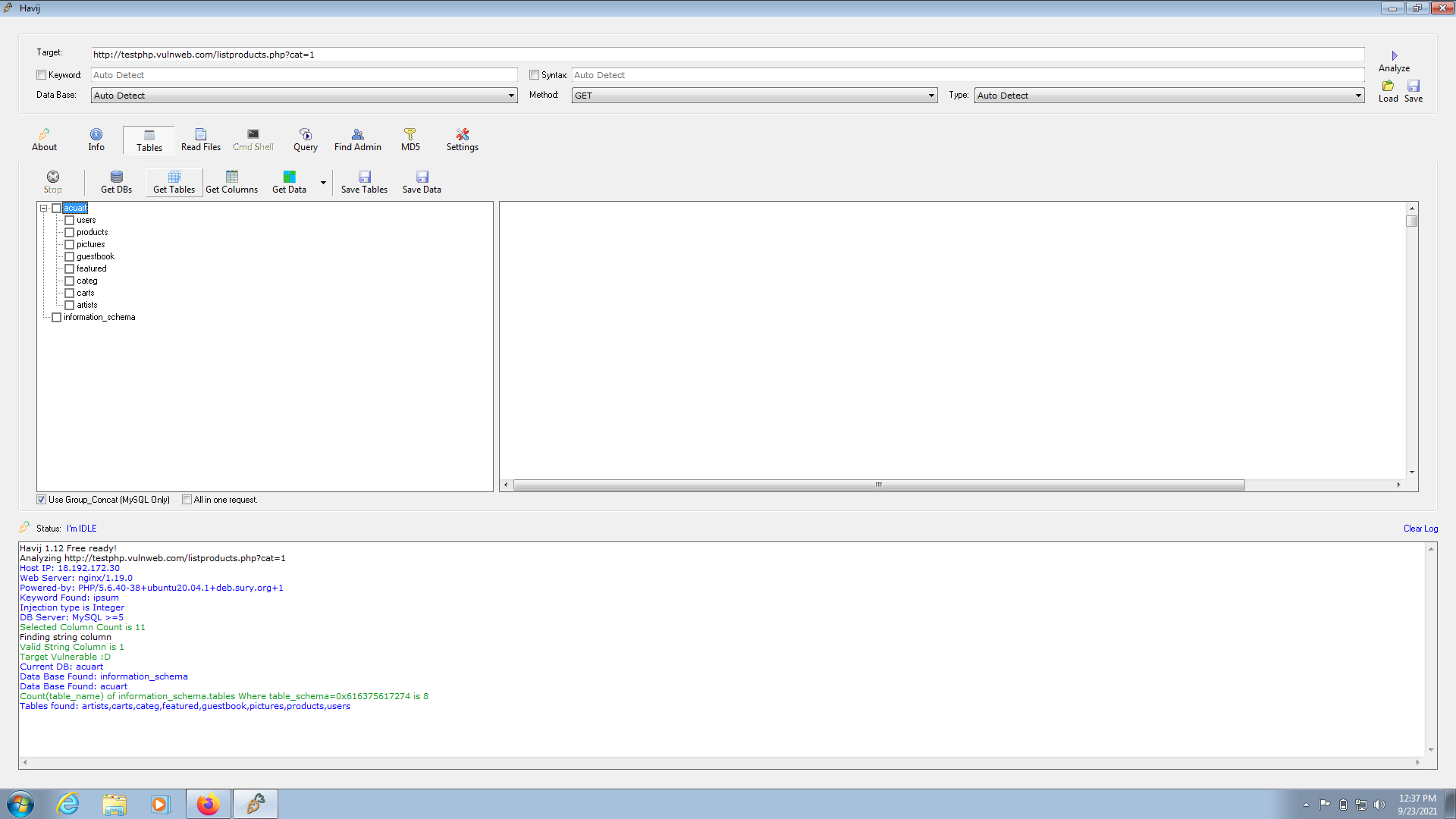
1. Click on the “***Tables”*** tab.

****

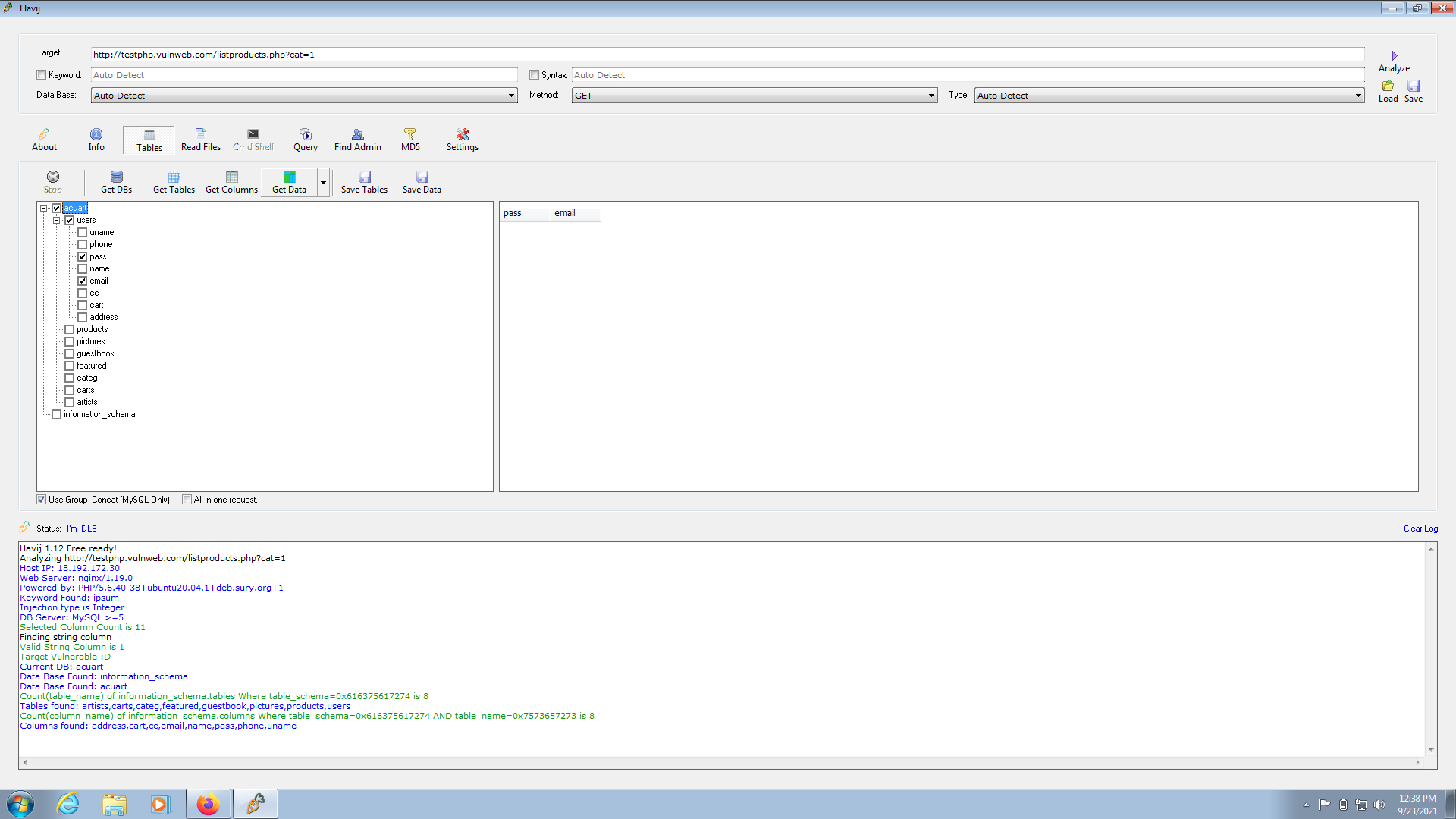
1. Click on “***Get DBs”*** option. This will list all the databases as shown below.

****

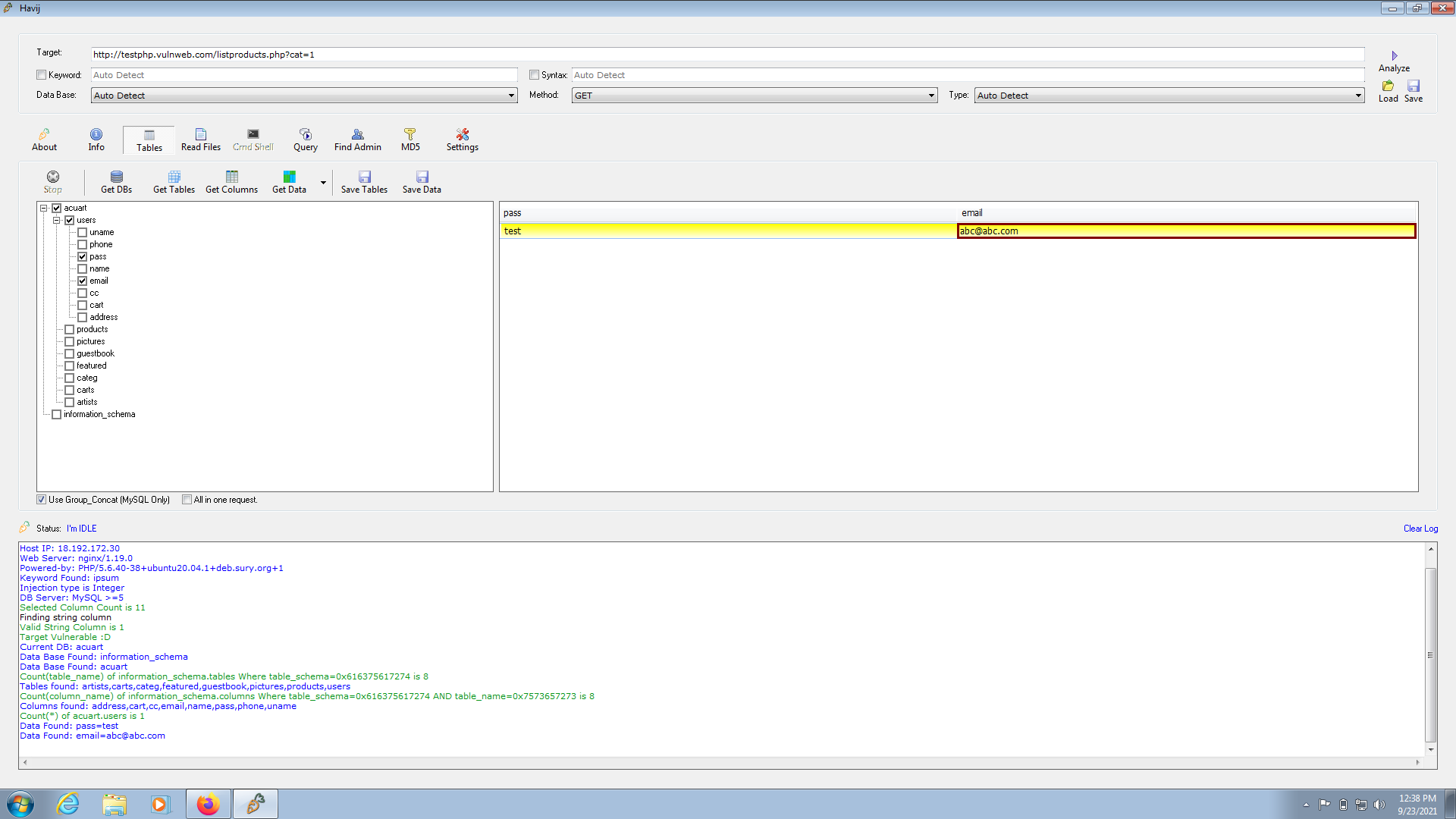
1. To get tables in a specific database, select the database and click on “***Get Tables”***. This will list all the tables present in the selected database. I selected database “***acurat”*** here.

****

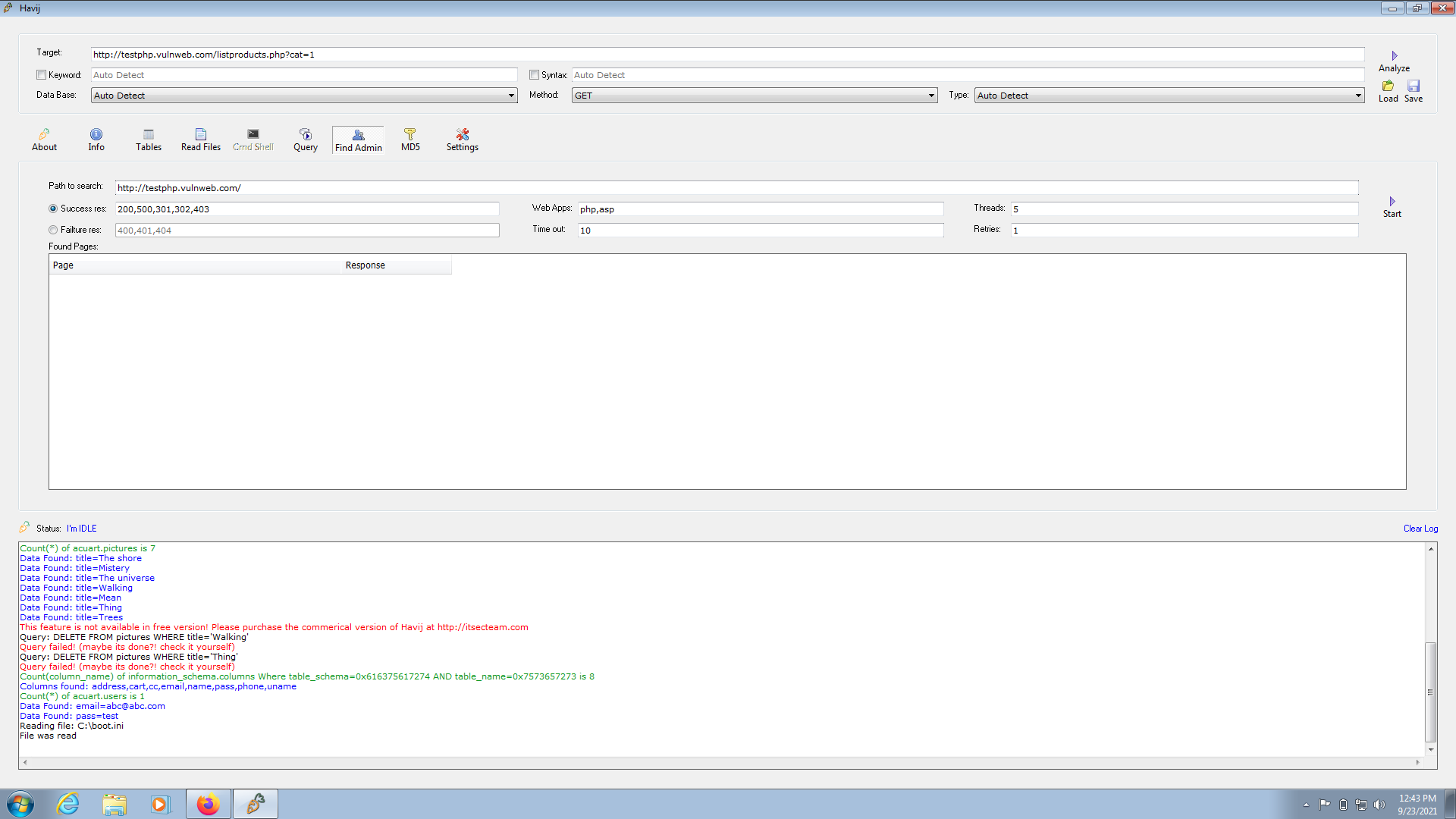
1. We can see that there is on table ‘users’ in our database ‘acurat’ .To get columns , select the table ‘ users’ and click on “***Get Columns”***.

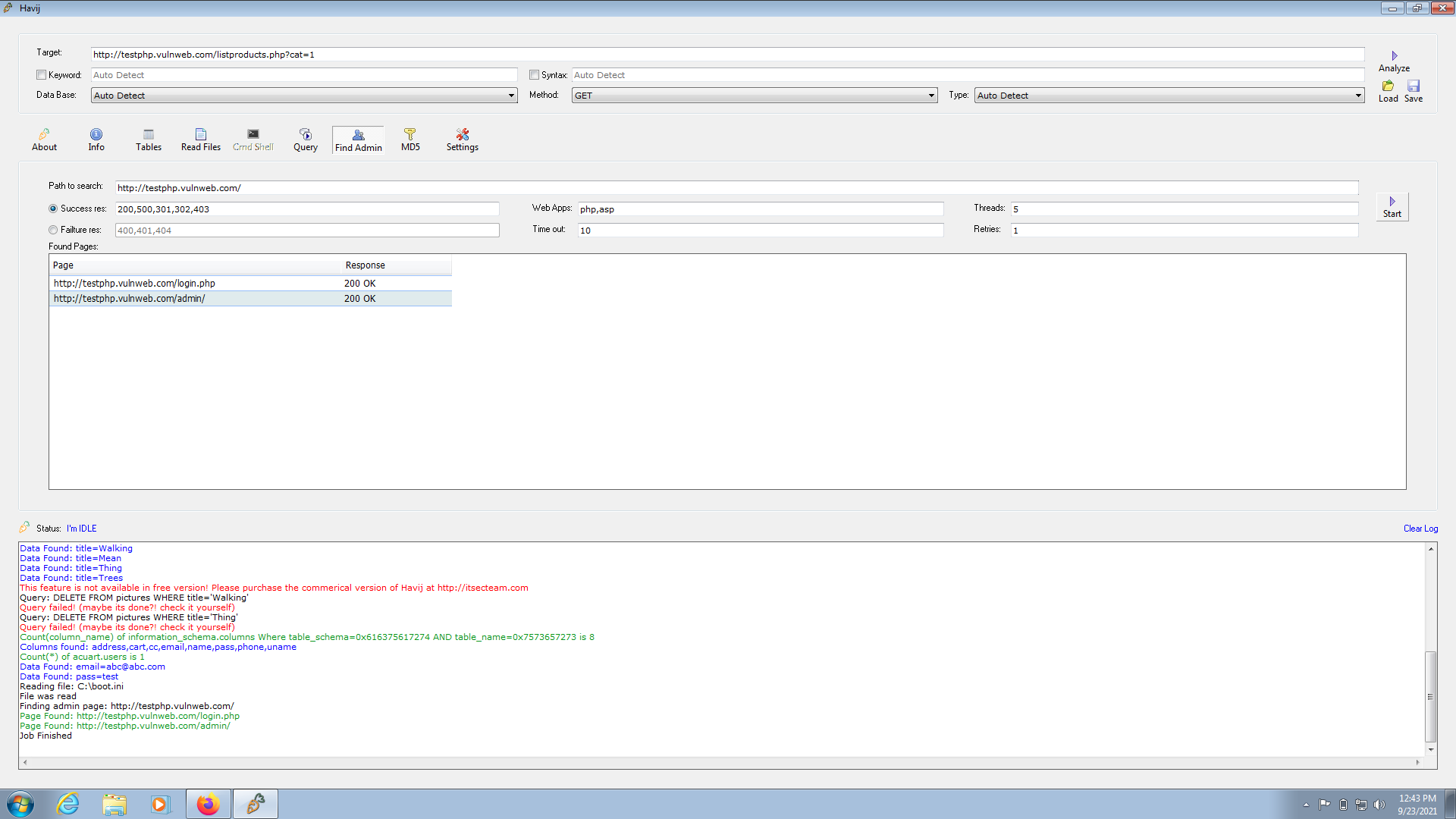
****

1. This will list all the columns in the table. We can see that we have eight columns in the table ‘users’.all the columns. It’s time to dump the values of columns. Select the columns whose data we want to dump and click on “***Get data”***. Here I selected pass and email columns.

****

1. Click on “***Find admin”***. This option finds the admin  page of the website automatically. When it finds the admin page, you can try the username and passwords to get access to the website.

****

****

**Preventive steps to avoid SQL injections:**

### 1. Validate User Inputs

A common first step to preventing SQL injection attacks is validating user inputs. First, identify the essential SQL statements and establish a [whitelist](https://www.esecurityplanet.com/trends/ibm-appscan-takes-aim-at-input-validation/) for all valid SQL statements, leaving unvalidated statements out of the query. This process is known as [input validation](https://www.esecurityplanet.com/trends/ibm-appscan-takes-aim-at-input-validation/) or query redesign. Additionally, you should configure inputs for user data by context. For example, input fields for email addresses can be filtered to allow only the characters in an email address, such as a required “@” character. Similarly, phone numbers and social security numbers should only be filtered to allow the specific number of digits for each.  While this action alone won’t stop SQLi attackers, it is an added barrier to a common fact-finding tactic for SQL injection attacks.

### 2. Sanitize Data by Limiting Special Characters

Another component of safeguarding against SQL injection attacks is mitigating inadequate [data sanitization](https://www.esecurityplanet.com/endpoint/prevent-web-attacks-using-input-sanitization/). Because SQLi attackers can use unique character sequences to take advantage of a database, sanitizing data not to allow string concatenation is critical. One way of doing this is configuring user inputs to a function such as MySQL’s mysql\_real\_escape\_string(). Doing this can ensure that any dangerous characters such as a single quote **‘** is not passed to a SQL query as instructions. A primary method of avoiding these unauthenticated queries is the use of prepared statements.

### 3. Enforce Prepared Statements and Parameterization

Sadly, input validation and data sanitization aren’t fix-alls. It’s critical organizations also use prepared statements with parameterized queries, also known as variable binding, for writing all database queries. By defining all SQL code involved with queries, or parameterization, you can distinguish between user input and code.  While dynamic SQL as a coding technique can offer more flexible application development, it can also mean SQLi vulnerabilities as accepted code instructions. By sticking with standard SQL, the database will treat malicious SQL statements inputted like data and not as a potential command.

### 4. Use Stored Procedures in the Database

Similar to parameterization, using stored procedures also requires variable binding. Unlike the prepared statements approach to mitigating SQLi, stored procedures reside in the database and are called from the web application. Stored procedures are also not immune to vulnerabilities if dynamic SQL generation is used.  Organizations like [OWASP](https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html) say only one of the parameterized approaches is necessary, but neither method is enough for optimal security. Crafting parameterized queries should be done in conjunction with our other recommendations.

### 5. Actively Manage Patches and Updates

Vulnerabilities in applications and databases that are exploitable using SQL injection are regularly discovered and publicly identified. Like so many cybersecurity threats, it’s vital organizations stay in tune with the most recent news and [apply patches](https://www.esecurityplanet.com/threats/microsoft-patches-massive-remote-access-vulnerability/) and updates as soon as practical. For SQLi purposes, this means keeping all web application software components, including database server software, frameworks, libraries, plug-ins, and web server software, up to date. If your organization struggles to consistently patch and update programs, a [patch management solution](https://www.esecurityplanet.com/products/patch-management-software/) might be worth the investment.

### 6. Raise Virtual or Physical Firewalls

We strongly recommend using a software or appliance-based [web application firewall (WAF)](https://www.esecurityplanet.com/networks/-security/application-firewalls.html) to help filter out malicious data.  Firewalls today, including [NGFW](https://www.esecurityplanet.com/products/top-ngfw/) and [FWaaS](https://www.esecurityplanet.com/cloud/firewalls-as-a-service-fwaas/) offerings, have both a comprehensive set of default rules and the ease to change configurations as needed. If a patch or update has yet to be released, WAFs can be handy.  A popular example is the free, open-source module [ModSecurity](https://modsecurity.org/), available for Apache, Microsoft IIS, and nginx web servers. ModSecurity provides a sophisticated and ever-evolving set of rules to filter potentially dangerous web requests. Its SQL injection defenses can catch most attempts to sneak SQL through web channels.

### 7. Harden Your OS and Applications

This step goes beyond mitigating SQL injection attacks in ensuring your entire physical and virtual framework is working intentionally. With the big news of [supply chain compromises in 2020](https://www.esecurityplanet.com/threats/guarding-against-solorigate-ttps-solarwinds-hack/), many are looking to NIST and other industry-standard security checklists to harden operating systems and applications.  Adopting application vendor security guidelines can enhance an organization’s defensive posture and help identify and disable unnecessary applications and servers.

### 8. Reduce Your Attack Surface

In cybersecurity, an attack surface refers to the array of potential entry points for attackers. So in the context of SQLi attacks, this means disposing of any [database](https://www.esecurityplanet.com/products/database-security-tools/) functionalities that you don’t need or further safeguarding them.

One such example is the [xp\_cmdshell](https://docs.microsoft.com/en-us/sql/relational-databases/system-stored-procedures/xp-cmdshell-transact-sql?view=sql-server-ver15) extended stored procedure in the Microsoft SQL Server. This procedure can spawn a Windows command shell and pass a string for execution. Because the Windows process generated by xp\_cmdshell has the same security privileges as the SQL Server service account, the attacker can cause severe damage.

### 9. Establish Appropriate Privileges and Strict Access

Given the power SQL database holds for an organization, it’s imperative to enforce [least privilege access policies](https://www.esecurityplanet.com/applications/privileged-access-management-pam/) with strict rules. If a website only requires the use of SELECT statements for a database, there’s no reason it should have additional INSERT, UPDATE, or DELETE privileges.   Further, your database should only be accessed with admin-level privileges when necessary, nevermind granting others access. Using a limited access account is far safer for general activity and ultimately limits an attacker’s access if the less-privileged credential is compromised.

### 10. Limit Read-Access

Connected to the principle of least privilege for SQL injection protection is configuring [read-access to the database](https://www.esecurityplanet.com/applications/tips-for-privileged-access-management-pam/). If your organization only requires active users employing read-access, it’s undoubtedly easier to adopt. Nevertheless, this added step is imperative for stopping attackers from altering stored information.

### 11. Encryption: Keep Your Secrets Secret

It’s best to assume internet-connected applications are not secure. Therefore [encryption](https://www.esecurityplanet.com/threats/tokenization-vs-encryption/) and hashing passwords, confidential data, and connection strings are of the utmost importance.

Encryption is almost universally employed as a data protection technique today and for a good reason. Without appropriate encryption and hashing policies, sensitive information could be in plain sight for an intruder. While only a part of the security checklist, [Microsoft notes](https://docs.microsoft.com/en-us/archive/msdn-magazine/2003/november/safeguarding-database-connection-strings-and-other-settings) encryption, “transforms the problem of protecting data into a problem of protecting cryptographic keys.”

### 12. Deny Extended URLs

Another tactic by SQLi attackers is sending excessively long URLs causing the server to fail at logging the complete request. In 2013, [*eSecurityPlanet* reported](https://www.esecurityplanet.com/networks/s/highly-critical-security-flaw-found-in-foxit-reader/) on how attackers exploited Foxit by sending users long URLs that would trigger a stack-based buffer overflow.

Microsoft IIS, as another example, is built to process requests over 4096 bytes long. However, the web server software fails to place the contents of the request in the log files. Attackers can then go undetected while performing queries. To avoid this, set a limit of 2048 bytes for URLs.

### 13. Don’t Divulge More Than Necessary in Error Messages

SQL injection attackers can learn a great deal about database architecture from error messages, ensuring that they display minimal information. Use of the “RemoteOnly” customErrors mode (or equivalent) can display verbose error messages on the local machine while ensuring that an external attacker gets nothing more than the fact that his or her actions resulted in an unhandled error. This step is critical in safeguarding the organization’s internal database structure, table names, or account names.

### 14. No Shared Databases or User Accounts

Shared databases by multiple websites or applications can be a recipe for disaster. And the same is true for user accounts that have access to various web applications. This shared access might provide flexibility for the managing organization or administrator, but it also unnecessarily poses a more significant risk. Ideally, any linked servers have minimal access to the target server and can only access the mission-critical data. Linked servers should have distinct logins from any process on the target server.

### 15. Enforce Best Practices for Account and Password Policies

While it might go without saying, organizations must follow the best account and [password policies](https://www.esecurityplanet.com/networks/s/password-failure-ceos-have-been-pwned/) for foolproof security. Default and built-in passwords should be changed upon receipt and before usage, with regularly scheduled password updates. Suitable passwords in length and character complexity are essential for all SQL server administrator, user, and machine accounts.

### 16. Continuous Monitoring of SQL Statements

Organizations or third-party vendors should continually monitor all SQL statements of database-connected applications for an application, including documenting all database accounts, prepared statements, and stored procedures. With visibility into how SQL statements function, it’s much easier to identify rogue SQL statements and vulnerabilities. In this continued review, admins can delete and disable unnecessary accounts, prepared statements, and stored procedures.  Monitoring tools that utilize machine learning and behavioral analysis like [PAM](https://www.esecurityplanet.com/products/privileged-access-management-pam-software/) and [SIEM](https://www.esecurityplanet.com/products/siem-tools/) can be excellent add-ons to your network security.

### 17. Perform Regular Auditing and Penetration Testing

Regular audits of your database and application security are becoming increasingly necessary, including auditing logs for suspicious activity, group and role memberships privileges, and variable binding terms.  Just as crucial as auditing for malicious behavior is conducting [penetration tests](https://www.esecurityplanet.com/products/best-penetration-testing/) to see how your defenses respond to an array of potential attacks, including SQLi. Most penetrating testing companies can find threats such as [cross-site scripting](https://www.esecurityplanet.com/endpoint/how-to-prevent-cross-site-scripting-xss-attacks/), retired software, unpatched vulnerabilities, injections, and insecure passwords.

### 18. Code Development & Buying Better Software

In the vast market of [software solutions](https://www.esecurityplanet.com/products/database-security-tools/), there’s certainly a hierarchy of solutions. While enterprise organizations can cover the cost of expensive third-party solutions and might even develop the software further in-house, smaller organizations rightfully work with less or consider free, open-source tools.  Though, to a great extent, vendor code writers are ultimately responsible for [flaws in custom applications](https://www.esecurityplanet.com/products/securing-web-application-code-at-the-source/) for a client. Organizations considering vendors must be keen on this and ensure the contract terms reflect this code review duty.