Deploying SQL Injection Attacks in

POST Method Data to Hack Websites

It is easier to dump the database if SQL Injection Vulnerability exists in the GET Method Parameters than in the case of POST Method Parameters. Havij or SQLmap, Tools cannot substitute knowledge. Here we explore how we can exploit an SQL Injection Vulnerability existing in the POST Parameter with the help of a case study.

SQL Injection Vulnerabilities have been exploited since they were first discovered by 'rain.forest.puppy' who wrote a paper about it in the 'Phrack' magazine in 1998. It has been a favorite of Hackers to gain access into the database which contains vital information about users like 'user-ids', 'passwords', 'emails' etc. There are numerous blogs and websites which detail the procedure to gain access into the website's database when such vulnerability exists in the GET parameter. But there aren't as many resources discussing the POST parameter SQL injection. Before we can move any further it becomes incumbent to comprehend the difference between GET and POST Methods.

The Two HTTP Request Methods: GET and POST

In HTML, one can specify two different submission methods for a form. The method is specified inside a FORM element, using the `METHOD` attribute. The difference between `METHOD="GET"` (the default) and `METHOD="POST"` is primarily defined in terms of form data encoding.

GET Method

The GET method means retrieve whatever information (in the form of an entity) is identified by the Request-URL Example of GET Parameter data:

```
GET /<website>/?parameter_name1=value1&parameter_
                    name2=value2 HTTP/1.1
```

Because query strings are transferred openly in GET requests, they are not used when dealing with sensitive information being passed in web-forms.

POST Method

The POST method is used to request that the origin server accept the entity enclosed in the request as a new subordinate of the resource identified by the Request-URI in the Request-Line

Usually POST method is used because the query stings are not passed openly in URL and it helps keep the URL short and simple while sending a request. Example of POST Parameter data:

```
POST /website/ HTTP/1.1
parameter_name1=value1&parameter_name2=value2
```

SQL Injection Attacks

A SQL injection attack consists of insertion or "injection" of a SQL query via the input data from the client to the application. A successful SQL injection exploit can read sensitive data from the database.

When an SQL Injection Vulnerability exists in the GET parameter, it's relatively easy to exploit it because we can clearly observe the parameters where we can inject the SQL queries. For example in the GET request above, we can identify two parameters:

```
parameter_name1 and parameter_name2
```

When it comes to POST data however, the Parameters are not so obvious since they are passed in the background and are never really displayed on-screen while an HTTP Request is passed. Without the knowledge of the Parameters, a Hacker will have problems figuring out where to Inject SQL queries. The Tools available to exploit SQL Injections are no wiser, as they have to be explicitly pointed to where the SQL queries are to be inject-

ed. Hence it is up to the Hacker to somehow locate the exact names of the POST Parameters.

Typically this can be done by viewing the source code of the webpage. The source code could be searched for forms where POST parameters are being passed. However on bulky pages with long lines of HTML, viewing source code and locating the POST parameters can slow a Hacker down. However a [Control+F] search for the term 'password' or 'login' or 'username' can take us to the right spot where the POST Parameters are named (Listing 1).

A much better way to determine information about these Parameters would be to intercept ongoing requests from the browser in a local proxy. These requests can then be studied to determine which POST parameters are being passed.

In our case we have a website where the usercode and password fields are making the use of HTTP POST Method. The first thing we do is try to manually check for the presence of SQL Injection Vulnerabilities. A very common way to do this is to put a 'Single Quote' in one of the fields (Figure 1)and forward the request. In this case we receive an error message as shown in Figure 2, which indicates that the website is vulnerable to SQL Injection Attacks. The input in the password field is not validated or filtered before passing to the Database and hence the MSSQL 2005 server generated the error message. Information of this sort is vital for Hackers.

Once we have checked for the presence of the SQL Vulnerability we can go ahead and try to exploit it. To do this we seek assistance from some well known tools. One of these Tools is Burpsuite.

Burpsuite has become a prominent tool in the arsenal of Hackers. It provides a wide range of functionality in a GUI Environment. Burpsuite has a 'Proxy' module that runs on a specified port on the system and listens for connections (Default Port is 8080). A browser can be configured to connect to Burpsuite Proxy at port 8080. If the Burpsuite Proxy is 'running' and the 'Intercept is on', the request from the browser would be intercepted in the middle by the Burpsuite proxy which allows us to modify requests and construct attacks.

Listing 1. *Viewing Webpage Source code to locate POST Parameters*

```
<font color="#972FFF" size="4">
  <span lang="HI" style="font-family: Mangal; ">
     पासवर्ड
  </span>
  <span style="font-family: Mangal; ">

  </span>
</font>
<span lang="HI" style="font-family: Mangal">
  <font color="#972FFF" size="4">
    :
  </font>
</span>
</td>
<td width="26%" height="19" align="center"  >
  <strong>
    <b>
      <input name="PasswdText" size="10"
style="float: left; font-family: Times New
Roman; font-size: 14pt; font-weight: bold"
tabindex="2" type="password">
    </b>
  </strong>
</td>
<td width="43%" height="19" align="center">
   
</td>
</tr>
```

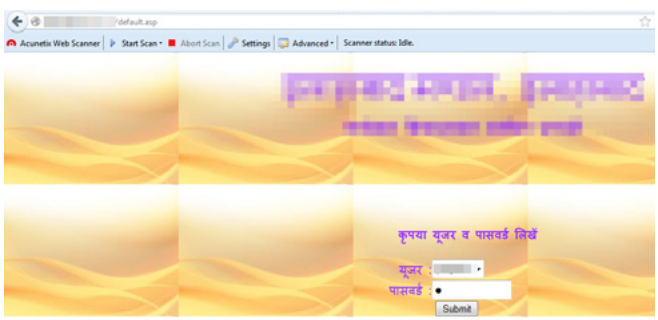

Figure 1. *Sending a 'Single Quote' in Password Field*

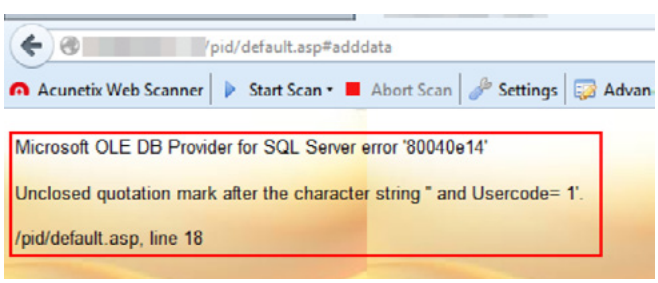

Figure 2. *Received Error Message from the Database Server*

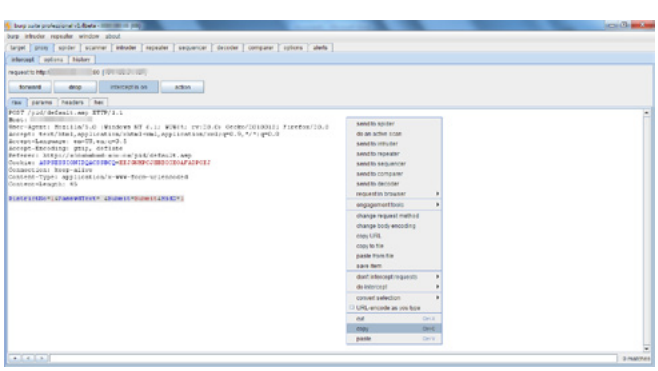

Figure 3. *POST Parameters intercepted by Burpsuite*

Shown in Figure 3, are the POST method parameters being held in Burpsuite proxy. At this point the POST Parameters can be grabbed and their 'values' can be changed. For our purpose we will try to inject SQL Injection Attack Queries in these Parameter values and see if we get success in enumerating available databases.

Havij is an advanced SQL Injection Tool available for windows. It has easy to use GUI and brings exploiting SQL Injection vulnerabilities down to simple point-and-click. In case of GET Request, you can specify the URL containing the GET Parameters and SQL would do its job. But in Case of POST Parameters, Havij needs to know where to inject SQL Injection queries. This location is provided to Havij by `%Inject _ Here%` as shown in Figure 4.

In this case, we know that an SQL Injection Vulnerability lies in `PasswordText` Parameter (as concluded by the error message shown to us by entering a single quote in the Password field), so we `%Inject _ Here%`.

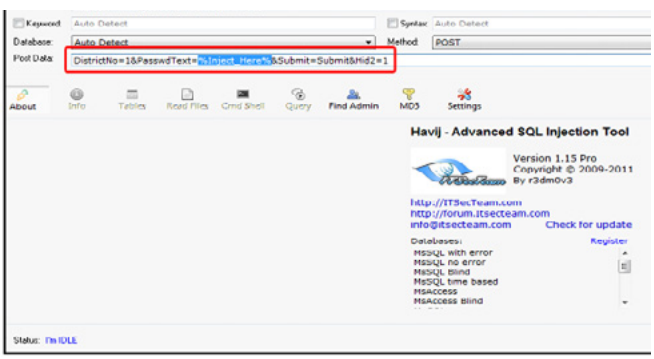

Figure 4. *Passing on POST Parameters to Havij*

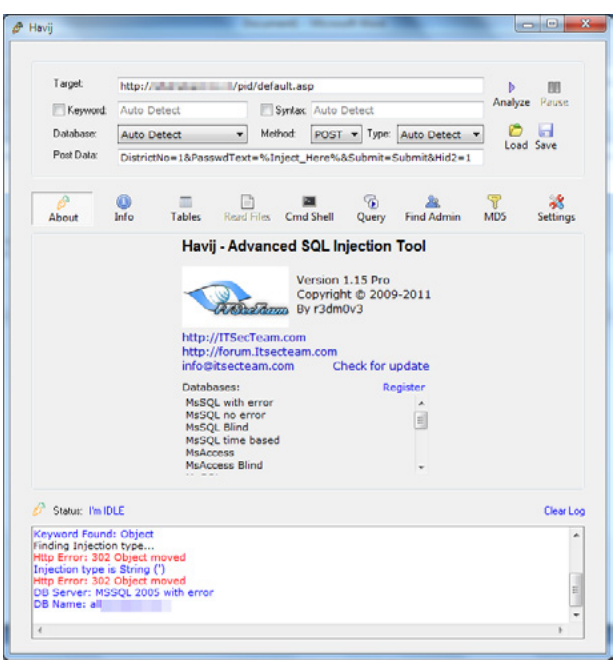

Figure 5. *Database Server Recognized as MSSQL 2005 and DB Name Enumerated*

Havij immediately recognizes by the error message shown, that the Database Server running is 'MSSQL 2005' and grabs the name of the available database (Figure 5). Next we can pull tables, columns and data from this database (Figure 6). In one of these tables we find the one table that contains the 'UserCode' and 'Password' columns (Figure 7). We pull the data in this table and notice that passwords are stored as Plaintext in the Database. This is a serious Negligence on part of the Website Administrator. If the Passwords were stored in hashed format, the Hacker would have to crack the Hashes. But in this case, it is not so. All 5 Users are compromised and we can use these 5

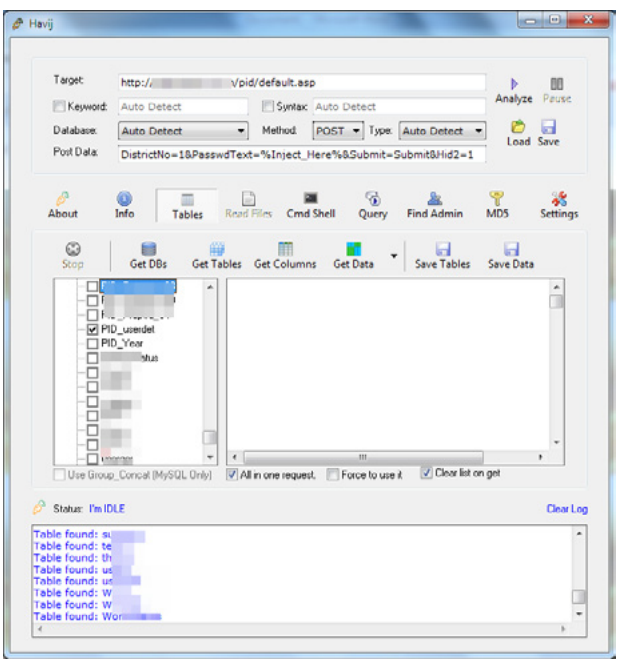

Figure 6. *Pulling Tables from the Database*

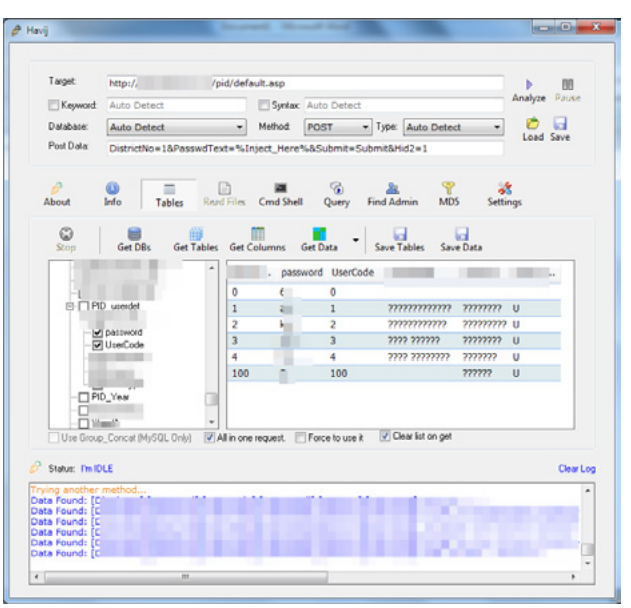

Figure 7. *'password' field found in one of the tables 'PID_ userdat'*

Passwords to log into each of their accounts (Figure 8). SQLmap is another great tool that can be used for SQL Injections for both GET and POST data. However this is a command line tool and provides no GUI. It is available for Linux Platform. Figure 9 shows the same SQL Vulnerability being exploited by SQLmap.

What is better, Havij or SQLmap?

While Havij is a proprietary tool, SQLMAP is open source. Havij is available for Windows Only where as SQLmap runs on any OS running a Python Compiler. Havij has the upper hand in terms of providing ease-of-use to the new inexperienced user because of its point and click GUI that SQLmap lacks. However Hackers argue that SQLmap is much more flexible than Havij. It's impossible to choose a winner unless you work on both and figure out what works best for your needs.

Conclusion

SQL Injection in POST data can be a little tricky if you don't know where to inject the queries. However with the aid of tools like Burpsuite Proxy we can intercept such POST Parameters with ease and then supply these to an automatic SQL Injection Tool and sit back to see if it can enumerate the databases and dump data. SQL Injection attacks have been widely misused since 1998 and it's amazing to see how many websites out there are still vulnerable to such attacks and many of them are downright negligent when they store sensitive information like passwords in plaintext. The need for securing websites against SQL Injection attacks can't be overlooked. Some of the points to be kept in mind to secure a website against such attacks are:

- Input Validation
- Parameterized Queries
- Using Stored Procedures
- Suppressing Error Messages that reveal information to the Hacker
- Installing Database by using a Least-Privileges account
- Encrypting Sensitive data (Passwords)

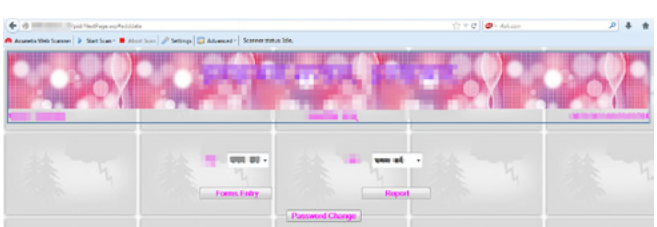

Figure 8. *Logging into Hacked Accounts*

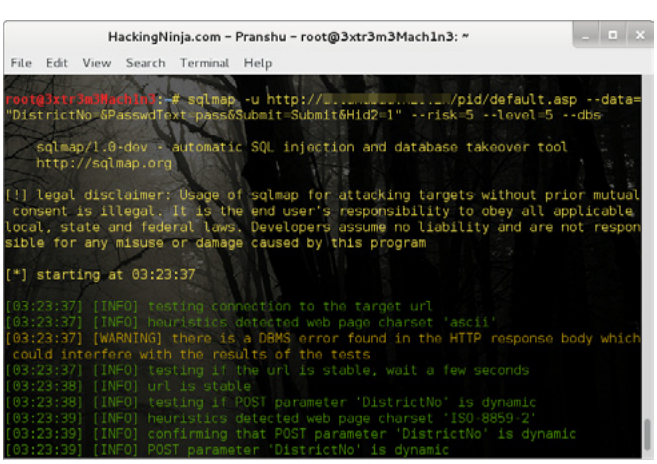

Figure 9. *Exploiting the SQL Injection Vulnerability in the POST Parameter using SQLmap*

On the Web

- *http://itsecteam.com/products/havij-advanced-sql-injection/* – Havij
- *http://sqlmap.org/* – SQLmap
- *http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html* – HTTP Methods

PRANSHU BAJPAI

Pranshu Bajpai is a Computer Security Professional specialized in 'Systems, Network and Web Penetration Testing'. He is completing his Master's in Information Security from the Indian Institute of Information Technology. Currently he is also working as a Freelance Penetration Tester on a Counter-Hacking Project in a Security Firm in Delhi, India, where his responsibilities include'Vulnerability Research', 'Exploit kit deployment', 'Maintaining Access' and 'Reporting'. He is an active speaker and author with a passion for Information security.

SAURABH MISHRA

Saurabh Mishra is a Cyber Security Professional. His area of interest includes Network & Web Penetration Testing, Attack Research, Defense Strategies, Post Exploitation Research and malware analysis. He has years of experience in Penetration Testing of many Government Organizations of India and Global Corporate Giants. He has discovered vulnerabilities in hundreds of Govt. and corporate websites & servers. Currently he is working as a Penetration Tester on a Hack-Defense Project in Cyber Security Division of National Informatics Centre in Delhi, India with a wide-range of responsibilities.