# Exploring Seeker Features


ACCURACY

CLARITY

SIMPLICITY

# Table of Contents

# 1 Introduction

Seeker is a leading application security testing solution which integrates into the software development life cycle to identify vulnerabilities that pose a real threat to business critical data.

Seeker provides accurate results in a clear and simple manner. The unique technology in Seeker allows it to fully analyze the application, as it runs, and correlates code and data flow with simulated attacks. More information on Seeker's technology is available at http://www.quotium.com/prod/security.php .

Accurate and relevant results are provided for each function in the organization in precisely the way they need it. Seeker delivers results tailored for clear visual information ranging from the big picture to the smallest details.

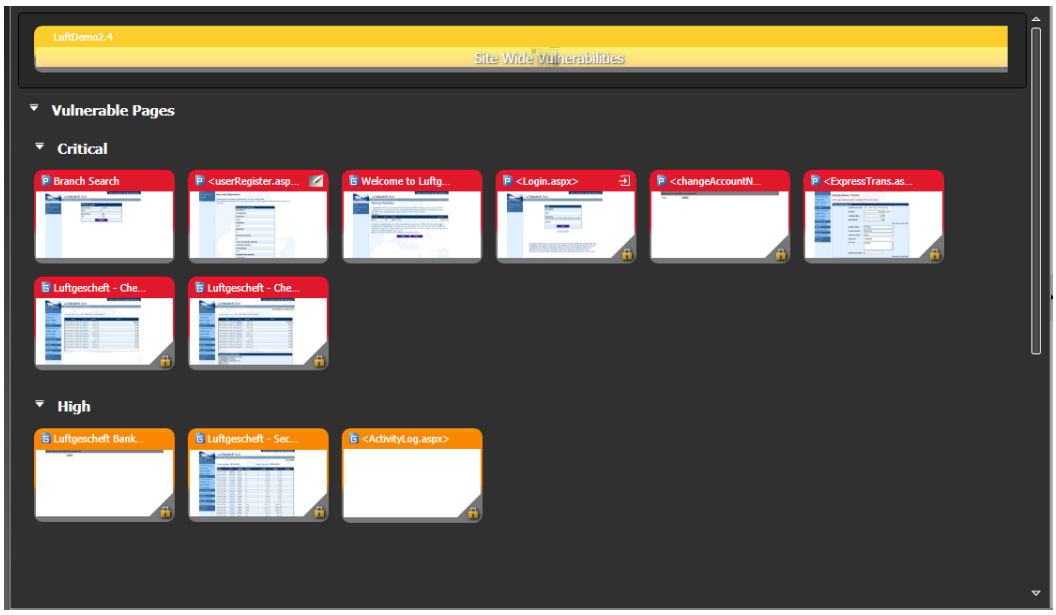



**Fig.1 – high level overview of application security status.**

Seeker's user interface is application-screen-oriented. Following a test, one of the views Seeker provides for the user is a birds-eye view of the severity of vulnerabilities identified in each of the application screens. This allows a quick overview of the security state of the application.

## 1.1 A Natural, Holistic Testing Process

Seeker mimics the work of a human tester. The process of ***Observe & Analyze → Test & Verify → Document*** is embedded into existing SDLC processes in the organization.

Seeker first observes normal application behavior, then analyzes the observed information and uses it in testing. Following testing, Seeker verifies and documents the results.

# 2 Observing Real-Application Use-Cases

Organizations invest a lot of effort to establish effective quality assurance and development tests. These cover use-cases of the application, business flows and end-to-end transactions. Seeker observes testing performed on the application as part of development and QA – these could be QA automatic tests, unit testing conducted by developers, QA manual testing and more. These cases are used by Seeker for security testing purposes.

To perform these procedures, a set of easy-to-learn tools are incorporated into Seeker.

## 2.1 Automated Learning

Seeker is designed to learn automatically about the application by integrating with existing automated (and manual) testing, using a variety of techniques.

### *2.1.1 Recording via a Proxy*

The proxy recording option built into Seeker allows a user to record application use cases using any external browser. This allows the user to record by using browsers other than Internet Explorer, or more importantly, to record an external tool driving requests to the application.

For example, in cases where automatic QA testing takes place, Seeker can be used to record the requests sent by the automatic QA tool and then use those for testing by Seeker.

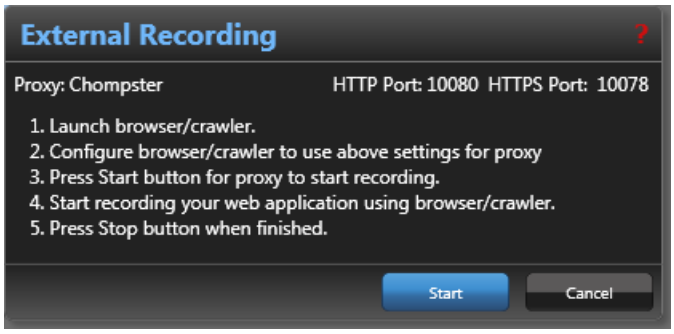



**Fig 2: First step of the external recording process though Seeker's UI.**

Seeker shows the user which functionality to activate. External recording can also be initiated from the command line to allow automatic invocation without manual intervention.

### 2.1.2 The Classic – Automatic Crawler

Seeker provides a fully functional automatic crawler, including full JavaScript support. By using the built in crawler a user can generate a website recording map with a few clicks. The automatic crawler can be used on its own to map the application, or as a supplement to other recording methods.

### 2.1.3 Browser Plug-In

Seeker provides a browser plugin which allows the user to perform recording of application use-cases while using Internet Explorer. The user simply activates a recording directly from the plugin without initiating the Seeker interface. A recording takes place in the background and then saved by the user in a Seeker project.

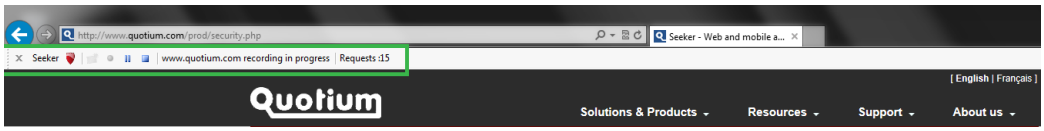



**Fig 3: Internet Explorer with a Seeker recording toolbar.**

## 2.2 Additional Manual Testing

In addition to automated testing, Seeker allows users to manually record additional components or even the entire application. This can be done by utilizing the above-mentioned browser plug-in or the built-in browser.

### 2.2.1 Built-In Browser

Seeker features an embedded browser that fully simulates Internet Explorer and allows the user to manually guide Seeker through the test scenarios. The user simply browses to the locations of the application that are to be tested. Seeker records these in the background. This functionality is intended for situations where a full security test is to take place and the user wants to ensure the specific components are tested by Seeker. This is usually done for third party software or full-blown pre-production application tests.

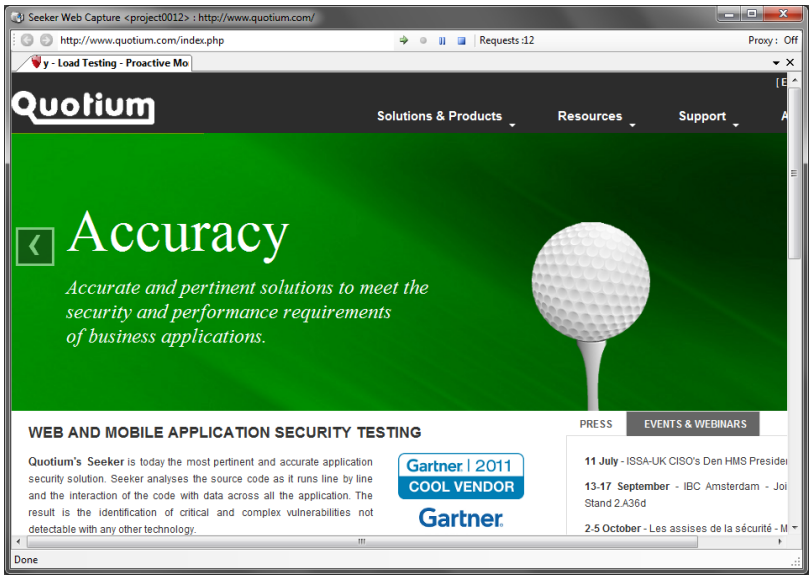



**Fig 4: Seeker's embedded browser.**

A recording can be paused or resumed as needed and a request count is shown for the user's convenience.

## 2.3 Application Business Logic

Seeker automatically analyzes recorded data and allows the user to add more information, if needed.

Requests are grouped and filtered for duplicates using rule-based as well as behavior-based analysis techniques. Sensitive information is then automatically identified and mapped (examples of sensitive information could be credit card data, user authentication data and more).

If needed, pages can be excluded from testing. Seeker also supports URL rewriting, Anti-CSRF tokens, custom request headers and more.

### *2.3.1 Special Pages*

After the automated detection by Seeker of special pages, Users are also given an opportunity to manually review identified special pages and sensitive data, and also configure any additional user-specific sensitive data (which could be any type of data considered sensitive according to the application business logic).

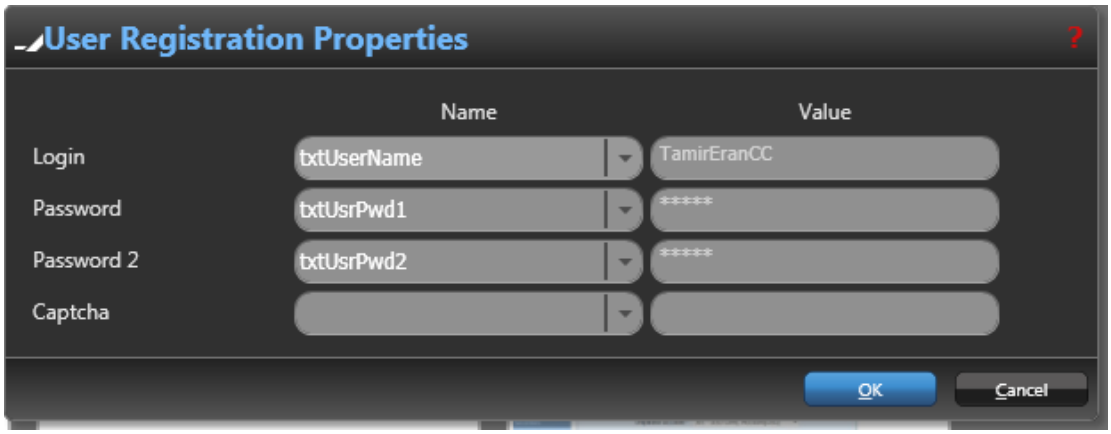



**Fig 5: automatically identified special page properties.**

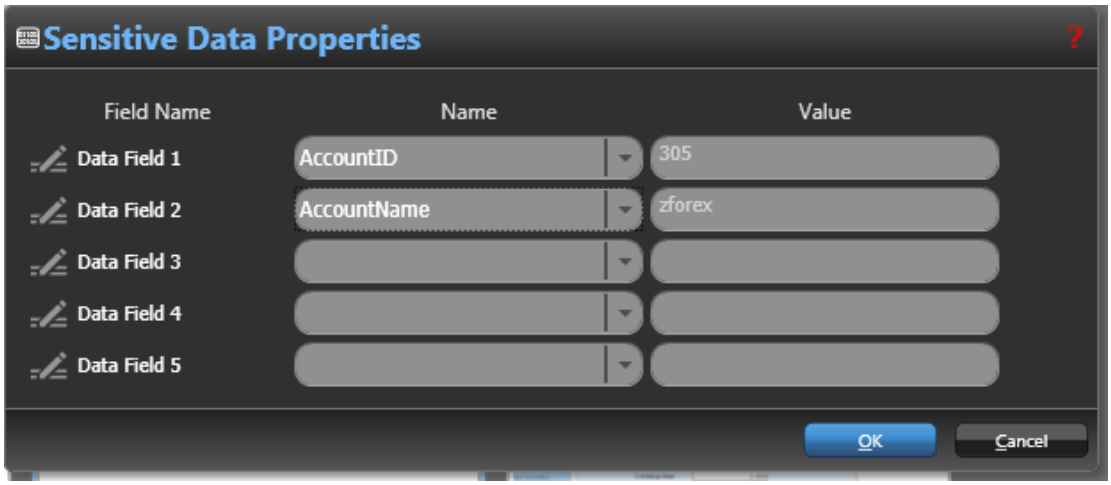



**Fig 6: user-defined special page properties**

### *2.3.2 Project Templates*

To support automation and to avoid re-configuring special pages, Seeker enables the user to create project templates.

A project template is used at the start of a new project (either from the Seeker graphical interface or from the command line interface). The template contains information about the login mechanism of the application, any special pages in it, recording properties and settings and any other information the user needs.

# 3 The Testing Process

Seeker analyzes the application code and data as it runs, in response to simulated attacks. Seeker monitors application behavior and data flow across modules, components, tiers and servers to accurately identify application threats.

See additional information on Seeker's technology at http://www.quotium.com/prod/security.php .

## 3.1 Fully Automatic, No Manual Intervention Needed

The testing process is fully automatic and does not require manual intervention. Once it learns the application behavior, Seeker does not need the user to point out the location of the vulnerabilities. All the links and connections in the application are mapped by Seeker and used in the testing process. Seeker knows when a different component, tier or data repository are invoked and monitors them automatically. The process is recorded in Seeker's detailed execution log which shows the steps that took place.

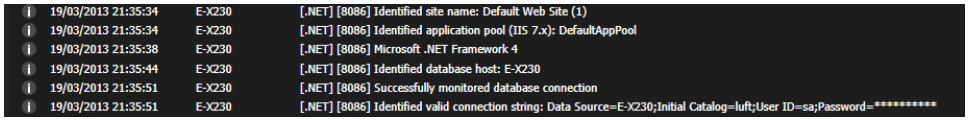



**Fig 7: Seeker's execution log shows identified processes, tiers, connection strings and more.**

### *3.1.1 Fully Customizable, if Needed*

Seeker allows the user to create custom testing policies or use pre-defined ones to launch more granular application security testing.

## 3.2 Verification and Exploitation of Vulnerabilities

Seeker verifies and/or exploits identified vulnerabilities. This is performed to verify the finding, assess the severity of the finding and its impact on business data, as well as to allow for proper documentation of the vulnerability with all relevant data. All vulnerabilities are documented alongside information about exploitation or verification.

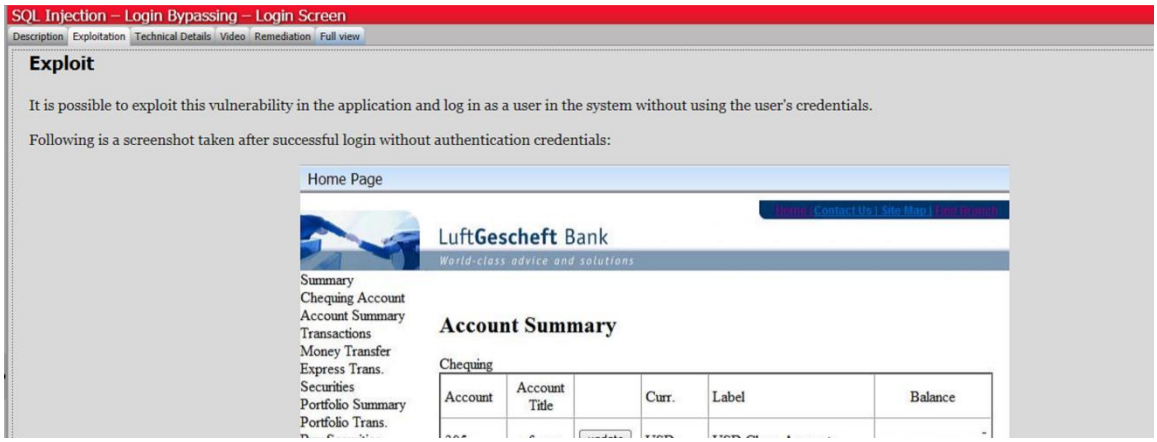



**Fig 8: Vulnerability business risk and exploitation information**

Seeker also displays the exploitation process in a video clip. This allows the user to fully understand the business impact of the vulnerability, as well as easily reproduce it.

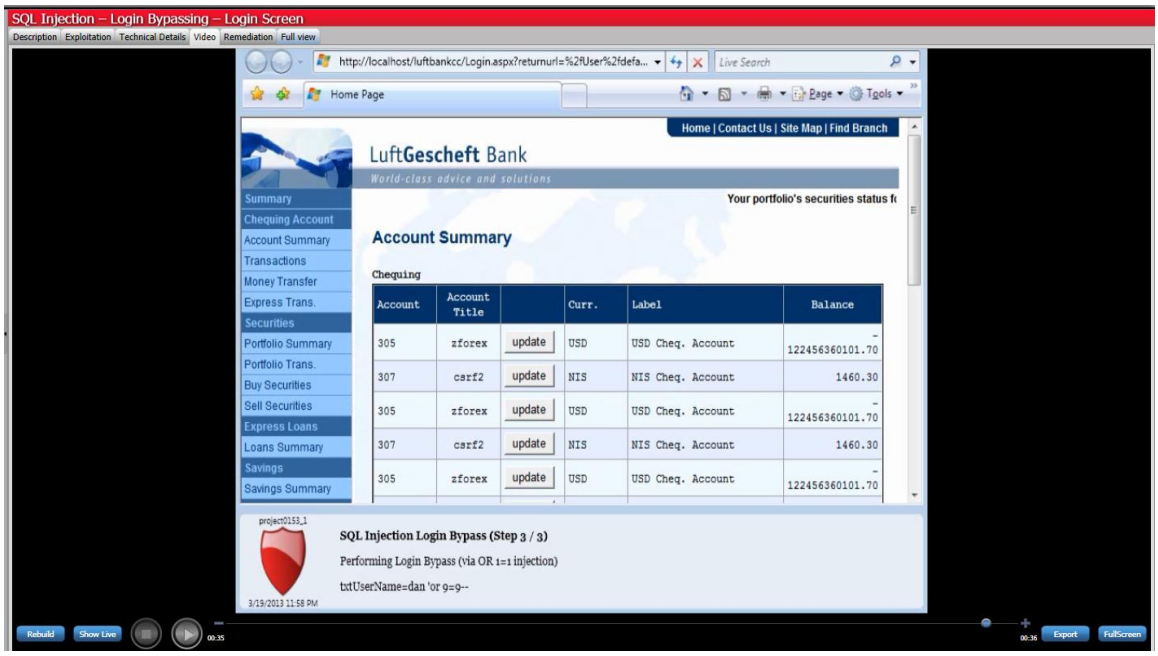



**Fig 9: a video clip.**

In this clip the user can see the actual step-by-step exploitation process of a vulnerability identified in the application, accompanied by explanations of the steps performed.

# 4 Test Results Delivered in a Tailored Manner

Seeker delivers results after analyzing business risk, eliminating false positives and ranking the severity and risk level of each vulnerability.

A results-view option is available for each function in the organization.

### *4.1.1 High Level Overview and Summary*

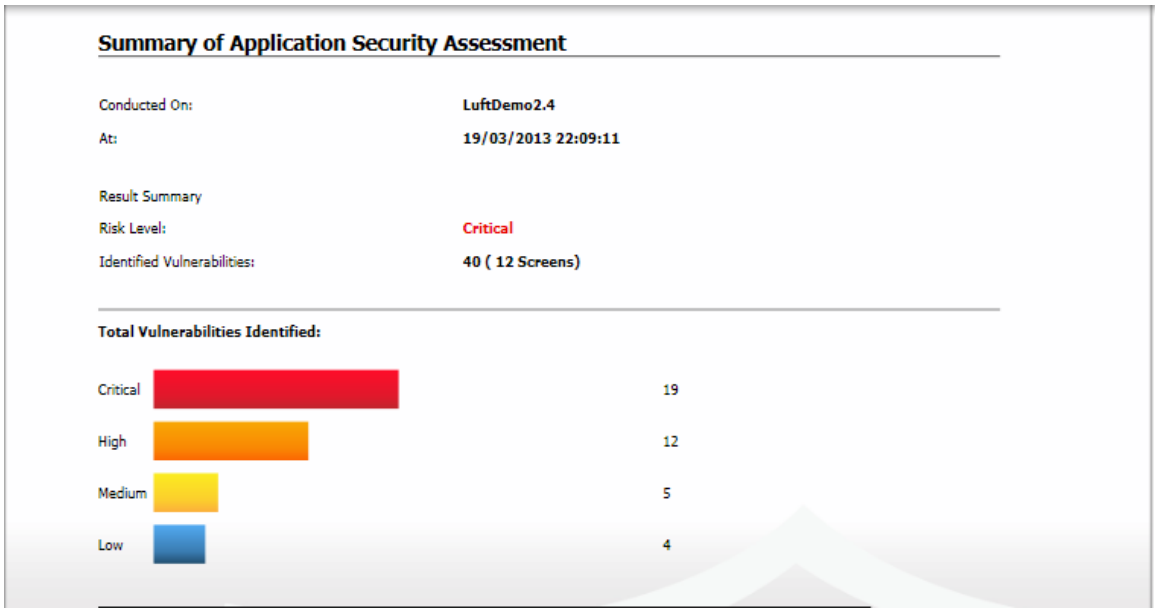



**Fig 10: Overview of an individual test's results**

### 4.1.2 Tailored Reports


Fig 11: Customization of Seeker's reports

## 4.2 Test Comparison and Security Status over Time

Seeker allows users who prefer a high level view to see the application status and compare the security progress of a specific application over time.

| URL | Parameter | Test1 20/3/2013 8:37:57 | Test2 1/7/2013 10:25:44 | |
|---|---|---|---|---|
| localhost:80/luftbankcc/userRegister.aspx | | | | |
| Persistent Cross Site Scripting (P-XSS) | txtUsrLastName | | | |
| Persistent Cross Site Scripting (P-XSS) | txtUsrFirstName | | | |
| SQL Injection | ccnumber | | | |
| SQL Injection | txtUserName | | | |
| SQL Injection | cvv | | | |
| Passwords Stored as Cleartext in Database | txtUsrPwd1 | | | |
| Reflected Cross Site Scripting (XSS) | txtUsrFirstName | | | |
| Reflected Cross Site Scripting (XSS) | txtUsrLastName | | | |
| Insecure Handling of Credit Card Information Vulnerability | ccnumber | | | |
| Weak Password Policy | (txtUsrPwd1, txtUsrPwd2... | | | |
| Username Enumeration | txtUsrPwd1 | | | |
| Insufficient SSL Enforcement | | | | |

Fig 12: Application vulnerability status compared between two tests

| Name | Test1<br>20/3/2013<br>8:37:57 | Test2<br>1/7/2013<br>10:25:44 |
|---|---|---|
| A1 - Injection | Fail | Fail |
| A2 - Cross Site Scripting (XSS) | Fail | Fail |
| A3 - Broken Authentication and Session Management | Fail | Fail |
| A4 - Insecure Direct Object References | Fail | Fail |
| A5 - Cross Site Request Forgery (CSRF) | Fail | Fail |
| A6 - Security Misconfigurations | Fail | Fail |
| A7 - Insecure Cryptographic Storage | Fail | Fail |
| A8 - Failure to Restrict URL Access | Fail | Fail |
| A9 - Insufficient Transport Layer Protection | Fail | Fail |
| A10 - Unvalidated Redirects and Forwards | Fail | Fail |

**Fig 13: compliance status compared between two tests**

## 4.3 Detailed Technical Reports

A user can view a detailed technical report of all the vulnerabilities identified in the application. Each vulnerability is provided with information on exploitation/verification, as well as all the relevant information necessary to understand, reproduce and remedy the vulnerability.

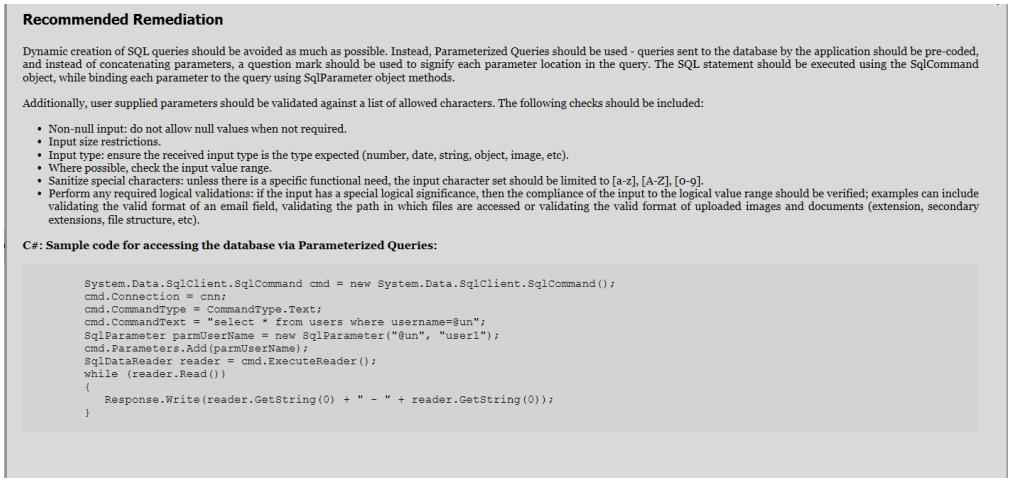
The source code that uses this parameter in the page is:

```
Code located on machine E-X230, at 127.0.0.1

Assembly Path:C:\Windows\Microsoft.NET\Framework\v4.0.30319\Temporary ASP.NET Files\luftbankcc\939ecb21\8ff99ef6\assembly\dl3\c85f093e
\791de0f8_56d9cd01\Luft.General.dll

Pseudo source code of execution from Luft.General.GeneralDB.searchBankBranch :

            {
                text += " and ";
            }
            text = text + " Address like '%" + p_address + "%'";
        }
        DbCommand sqlStringCommand = database.GetSqlStringCommand(text);
        database.AddInParameter(sqlStringCommand, "@ID", DbType.Int32, p_code);
        database.AddInParameter(sqlStringCommand, "@Type", DbType.Int32, p_typeCode);
        database.AddInParameter(sqlStringCommand, "@City", DbType.Int32, p_cityCode);
        database.AddInParameter(sqlStringCommand, "@Address", DbType.String, p_address);
        return database.ExecuteDataSet(sqlStringCommand);

    }
```

**Fig 14: Root cause analysis of the vulnerability showing vulnerable source code.**

### 4.3.1 *Extensive Context-Based Remediation Instructions*

Seeker provides remediation instructions based on the vulnerability variant and the coding language of the application.



**Recommended Remediation**

Dynamic creation of SQL queries should be avoided as much as possible. Instead, Parameterized Queries should be used - queries sent to the database by the application should be pre-coded, and instead of concatenating parameters, a question mark should be used to signify each parameter location in the query. The SQL statement should be executed using the SqlCommand object, while binding each parameter to the query using SqlParameter object methods.

Additionally, user supplied parameters should be validated against a list of allowed characters. The following checks should be included:

- Non-null input: do not allow null values when not required.
- Input size restrictions.
- Input type: ensure the received input type is the type expected (number, date, string, object, image, etc).
- Where possible, check the input value range.
- Sanitize special characters: unless there is a specific functional need, the input character set should be limited to [a-z], [A-Z], [0-9].
- Perform any required logical validations: if the input has a special logical significance, then the compliance of the input to the logical value range should be verified; examples can include validating the valid format of an email field, validating the path in which files are accessed or validating the valid format of uploaded images and documents (extension, secondary extensions, file structure, etc).

**C#: Sample code for accessing the database via Parameterized Queries:**

```
System.Data.SqlClient.SqlCommand cmd = new System.Data.SqlClient.SqlCommand();
cmd.Connection = cnn;
cmd.CommandType = CommandType.Text;
cmd.CommandText = "select * from users where username=@un";
SqlParameter parmUserName = new SqlParameter("@un", "user1");
cmd.Parameters.Add(parmUserName);
SqlDataReader reader = cmd.ExecuteReader();
while (reader.Read())
{
   Response.Write(reader.GetString(0) + " - " + reader.GetString(0));
}
```

**Fig 15: remediation recommendations**

## 4.4 Exports

In addition to full reports and integration with bug tracking systems, users can export results in multiple formats as required by the organization (XML, CSV and more). This functionality can be used to view Seeker results using external tools that exist in the organization.

## 4.5 Compliance with Industry Standards

Seeker provides built-in reporting options to comply with leading industry standards such as OWASP top 10 or SANS top 25. It is possible to generate reports according to these standards, view specific projects' improvement of compliance over time, launch tests that check for compliance with specific standard categories and more.

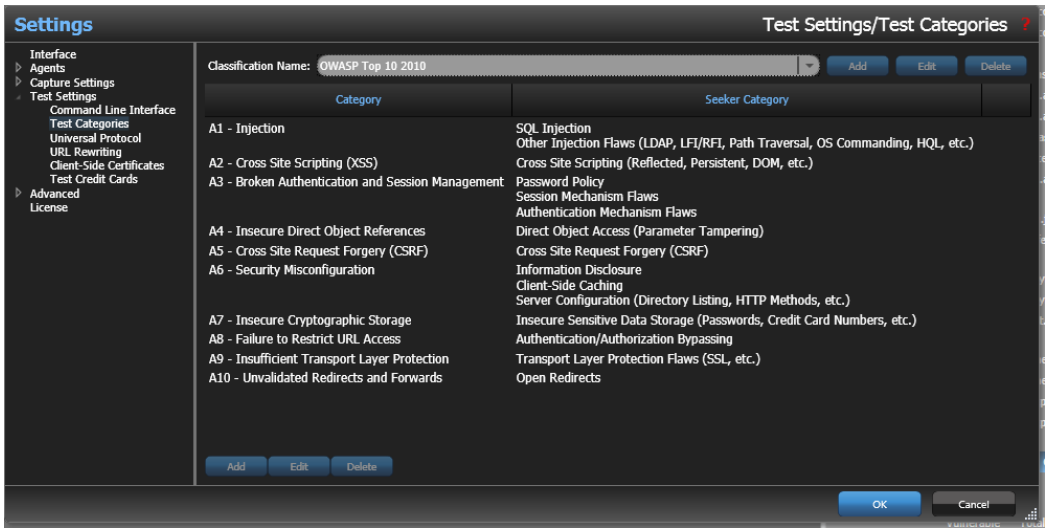



**Fig 16: Mapping between OWASP top 10 and vulnerability testing performed by Seeker**

The custom classification categories can be used in reporting or as a baseline for custom testing policies.

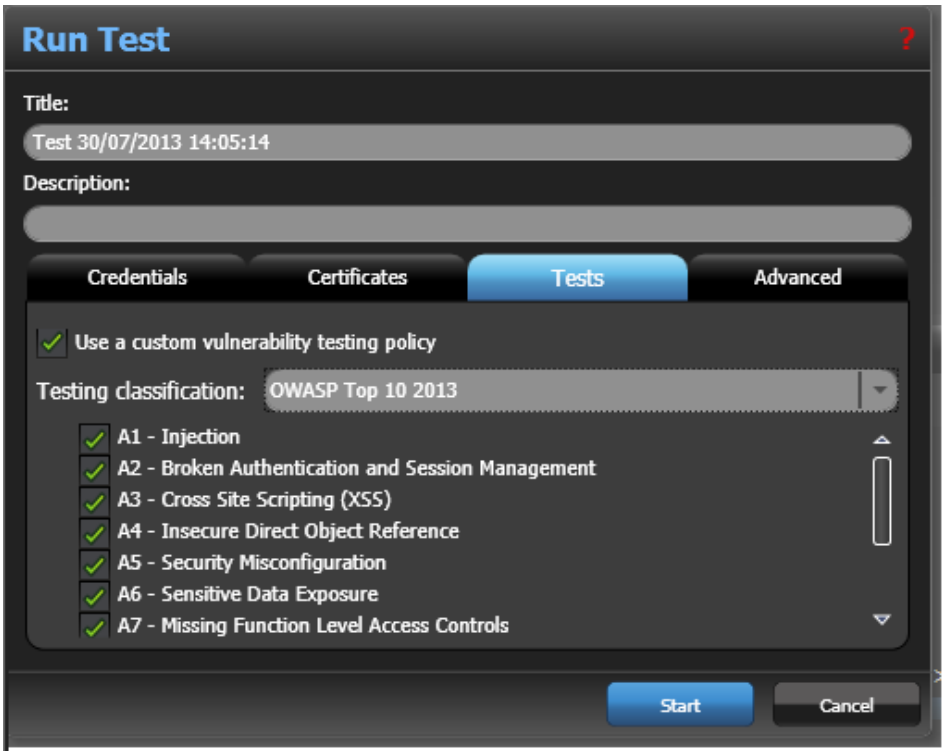



**Fig 17: Using common standards such as OWASP as a baseline for test policies**

### 4.5.1 *Application Security –Organizational Standards*

Seeker allows the user to create custom test categories and classifications. These will be used by Seeker to classify findings and display compliance information in reports, test comparisons, and more.

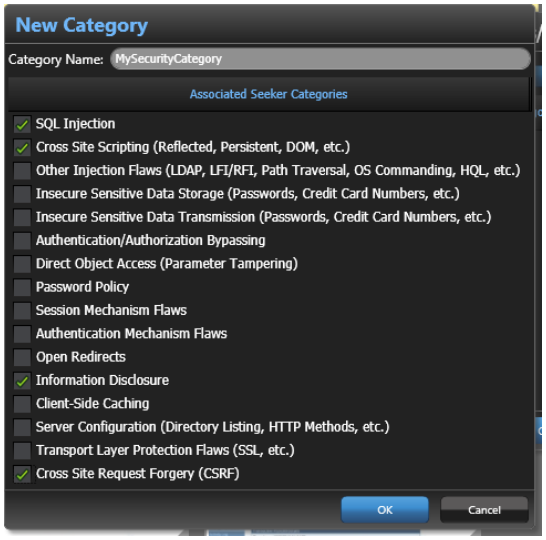



**Fig 18: Creation of a custom vulnerability classification scheme**

**OWASP Top 10 2010 Compliance**

| Threat | Status | Severity | # Vuln |
|---|---|---|---|
| A1 - Injection | Fail | Critical | 8 |
| A2 - Cross Site Scripting (XSS) | Fail | Critical | 14 |
| A3 - Broken Authentication and Session Management | Fail | Critical | 4 |
| A4 - Insecure Direct Object References | Fail | Critical | 1 |
| A5 - Cross Site Request Forgery (CSRF) | Fail | Medium | 2 |
| A6 - Security Misconfiguration | Fail | Low | 3 |
| A7 - Insecure Cryptographic Storage | Fail | High | 4 |
| A8 - Failure to Restrict URL Access | Fail | Critical | 1 |
| A9 - Insufficient Transport Layer Protection | Fail | High | 2 |
| A10 - Unvalidated Redirects and Forwards | Fail | Medium | 1 |

**MySecurityClassification Compliance**

| Threat | Status | Severity | # Vuln |
|---|---|---|---|
| SQL Injection & XSS | Fail | Critical | 21 |
| Injections | Fail | High | 1 |
| Insecure Handling of Data | Fail | High | 6 |
| Authentication & Authorization | Fail | Critical | 5 |
| Client Side Flaws | Fail | Low | 2 |
| Server Configuration | Fail | Low | 1 |
| Insecure Redirects | Fail | Medium | 1 |
| CSRF | Fail | Medium | 2 |
| Session and Passwords | Fail | Critical | 1 |

** This version of Seeker only tests application vulnerabilities and does not test the configuration of the web server.*

**Fig 19: Internet Explorer with a Seeker recording toolbar.**

# 5 Integration and Automation – Bringing it all Together

Seeker integrates into existing development and testing tools in the organization, and becomes a part of the SDLC processes. The Command Line Interface (CLI), proxy recording, ticketing functionalities and a variety of other interfaces make it a perfect solution for integration into Agile development environments. Seeker can be easily configured to launch tests automatically as part of regression or on new builds, and open tickets directly in the bug tracking software when it finishes testing. Seeker can also be easily used in non-Agile development procedures.

Seeker integrates easily out-of-the-box with all common QA and development tools such as Selenium, HP QTP, Jenkins, Hudson, JIRA, Microsoft TFS and more

## 5.1 Integration with Ticketing Systems

To allow security testing to be part of regular Quality Assurance cycles, Seeker provides integration into all common ticketing systems. By supporting HP HQ, IBM Clear Quest, Microsoft Team Foundation Server, Bugzilla, Jira, Seeker allows the user to open tickets in a ticketing system of his choice.

Seeker-generated tickets are clearly marked in the ticketing system to allow the user to easily generate reports of security status from within the bug tracking software used by the organization.

Seeker offers flexibility in opening tickets. Tickets can be opened for a specific vulnerability, for a specific page, for the entire project, or by severity levels of vulnerabilities.

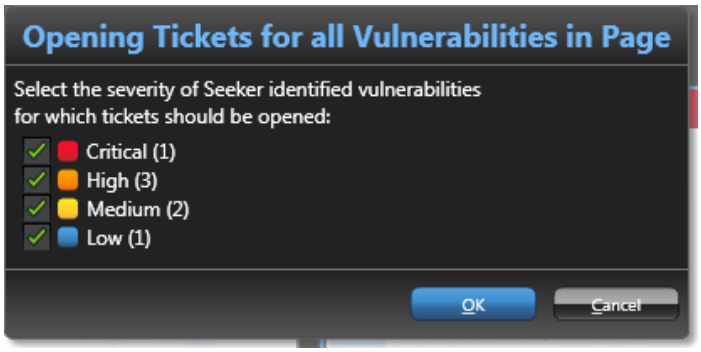



**Fig 20: Opening tickets from Seeker's GUI for all vulnerabilities in a specific page**

Ticket severity is configurable, allowing the user to map vulnerability severity in Seeker to its equivalent vulnerability severity in the ticketing system.

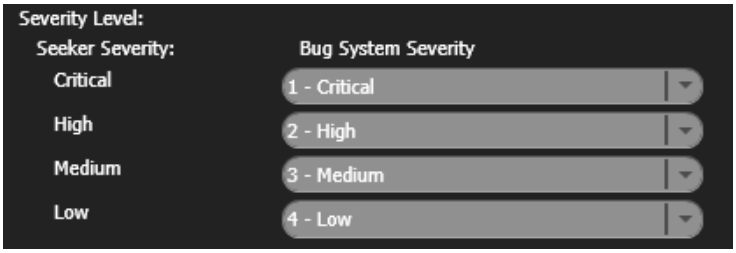



**Fig 21: configuring severity mapping between Seeker and the bug tracking system**

Tickets can also be opened from the command line interface (CLI), allowing for full automation of the entire ticket-opening process.

## 5.2 Extensive, Scalable Command Line Functionality

Key functionality provided in the Seeker graphical interface is available also via the command line.

The user can start the Seeker recording proxy from the command line. This allows the user to use any external tool to drive requests that record scenarios for Seeker to test. The user can provide templates for recording to specify any additional parameters for the test. Recording tests can be launched directly via the command line, allowing the user to schedule tests at convenient times, or within the automation process. In addition, reports can be generated and bugs can be opened directly from the command line.

# 6 Summary

In this paper we have explored the key features of Seeker, a leading application security testing solution. We have followed the steps of the ***Observe & Analyze → Test & Verify → Document*** process carried out by Seeker.

Seeker mimics the behavior of a human tester. It observes application use cases as carried out by automatic tools or by manual testers. A variety of utilities built into Seeker simplify this process. Seeker also observes data and business logic of the application to better understand the impact of identified vulnerabilities.

During testing, Seeker utilizes its unique technology to accurately identify vulnerabilities which pose a real threat. Vulnerabilities are pinpointed, false positives eliminated, and threats are ranked and classified by their business impact on application data. The process is fully automatic, requiring no manual intervention. At the same time all testing can be fully customized to cater for specific organization needs.

Test results delivered by Seeker can be tailored for individual needs. Built-in report formats provide reports from high level overviews to full technical details reports. All results are provided with information tuned for remediation, allowing developers to easily understand and fix vulnerabilities in the application. Application security status can be compared over time to understand risks and track progress.

All the functionality in Seeker is designed to be used as part of development and testing processes. Seeker provides out-of-the box integration with common development and testing tools, and Seeker's key features can be invoked via the command line, allowing for seamless integration into existing processes and tools in the organization. Many of the Seeker functions are aimed specifically towards integration into the Agile development process.

Seeker is the best quality application security testing solution for any organization.

# 7 About Seeker

Seeker is the leader of the new generation of application security testing software. Easily integrating with existing software testing processes, Seeker allows developers to efficiently develop secure applications.

# 8 About Quotium

Quotium Technologies is a specialist in the development of innovative software solutions to guarantee the security and performance of business critical applications throughout their lifecycle.