Penetration Testing

This topic explains how to perform penetration testing with Parasoft SOAtest.

- Overview
- Creating Penetration Testing Scenarios
- Executing Penetration Testing Scenarios
- Reviewing Results
- Configuring Scan Policies
- Penetration Testing Rules Supported
- Integration with Burp Suite

Overview

Penetration testing is critical to uncover security holes in your application. With Parasoft SOAtest, you can efficiently take your existing API functional testing scenarios and create security tests, adding penetration testing into your automated CI process.

Creating Penetration Testing Scenarios

SOAtest supports penetration testing of REST and SOAP APIs that are accessible over HTTP or HTTPS.

Penetration testing is supported by starting with a functional test scenario that contains the APIs that need penetration testing and then configuring those scenarios for penetration testing. Existing functional test scenarios can be leveraged and turned into penetration tests, or you can create custom simplified test scenarios meant only for penetration testing.

Repurposing functional tests for use as security tests gives the following benefits:

1. Since functional tests have already been written, you can reuse work that has already been done, saving time and effort.
2. To execute certain APIs, you might have to do some setup, like prepping the database or calling other APIs. If you start with functional tests that already work, this setup is already done.

The general workflow for configuring penetration test scenarios is:

1. Identify the test scenarios that you want to use for penetration testing and copy them. You can continue executing the original test scenarios for functional testing as normal.
2. Add the Penetration Testing Tools to the Traffic Object output of the test clients (e.g., SOAP Client, REST Client, EDI Client, or Messaging Client) that make the API calls that need penetration testing.
3. As the application changes, update only the functional test scenarios. Whenever you are ready to run the corresponding penetration test scenarios, repeat the above process of copying from the latest set of functional tests and then configuring the copy for penetration testing.

Penetration testing requires a user license with the API Security Testing license feature enabled.

Best Practices for Creating Security Test Scenarios

Inherent risks of penetration testing

By its nature, penetration testing simulates malicious user activity to find security vulnerabilities in the application under test (AUT). Because of this, penetration testing may cause the following effects on the AUT or the system on which the AUT or SOAtest is running:

- The AUT can be brought down.
- The AUT data can be modified or corrupted.
- If an antivirus application is installed on the machines where the AUT or SOAtest are running, it can interpret SOAtest penetration testing activity as suspicious or malicious and generate warnings or take other actions.

Following the best practices will help you mitigate these and other potential issues that may be caused by the execution of penetration tests.

- You should maintain your functional test scenarios separately from your security test scenarios, and run them from separate test jobs. The main reason for this is that adding penetration testing to existing functional tests will likely serve to destabilize the functional tests. You need to select which functional test scenarios should be turned into automated security tests, and then make copies of the functional tests that will be maintained as separate security tests.
- You need to be selective in which tests you choose since penetration testing is expensive; you need to maximize the attack surface of the API that is covered while minimizing the number of tests. You should consider the following:
  - The Penetration Testing Tool analyzes request/response traffic to understand which parameters in the request are available to be tested. You need to select functional tests that exercise all the parameters in each API, to ensure that every input to the API gets analyzed.
  - Within each scenario, you need to decide which API calls should be penetration tested. The same API may be referenced from multiple scenarios, and you don't want to duplicate penetration testing on an API that is tested in a different scenario. The Penetration Testing Tool should only get added to the appropriate tests for the API(s) to be penetration tested.
  - The number of scenarios needs to be manageable so that the security test run is short enough to run at least once a day.
- Your functional test scenarios may have set up or teardown sections for initialization or cleanup. These typically don't need to be penetration tested.

- If the functional test has any parameterization, you should remove it. The Penetration Testing Tool doesn't need to see multiple sets of values for the same parameters to know what to test, and sending different sets of values could just lead to making the test runs go longer due to duplicated testing.
- You should remove all assertions from any functional test scenarios that were converted into penetration test scenarios. API functional tests will usually have assertions that validate the response from the service. When used as security tests, these assertions can fail but will be noisy when reviewing the results, since in this context you only care about the security vulnerabilities that were found.
- Some API calls add data to the database. When using the Penetration Testing Tool against such APIs, the database can get bloated with information due to the number of attacks that the penetration testing tool directs at the API. In some cases, this can cause unexpected side effects, such as the performance of the application becoming so bad that the automated security test run cannot finish in a reasonable amount of time. If you see behavior like this, exclude the security tests for that API from your automated run until the development team can fix the problem.
- You need to consider whether to run your functional and security tests within the same test environment or a different one. Resetting the environment between the functional and security test runs, or using a separate environment, promotes better test stability but is usually not necessary. You can often reuse the same environment, but when you do, you should run the functional tests first and the security tests last, since the security tests can destabilize the environment for the functional tests. When you use different environments, you need to make sure that the original functional test scenarios are configured with variables so that it is easy to point the tests at different endpoints for different environments. SOAtest supports this using environment variables. (See Configuring Testing in Different Environments.)

Executing Penetration Testing Scenarios

To run your penetration testing scenarios, execute them as described in Executing Functional Tests.

- When a REST Client or SOAP Client with an attached Penetration Testing Tool is executed, the corresponding request and response data is captured and used as a starting point by the Penetration Testing Tool to execute the penetration test.

Because penetration testing can take a long time to run, even for a single API, the Penetration Testing Tool has a built-in timeout that governs how long the tool will run before moving on to the next test. The timeout complies with the Connection Settings default timeout configured in the Misc Preferences. If the timeout is reached before the penetration testing for the current API is complete, the penetration testing for that API will be interrupted and an error will be reported. The timeout can be extended in the Misc Preferences by updating the default timeout. See Misc Settings.

Reviewing Results

When running via UI, errors are reported to the Quality Tasks view, and details about the error and how to fix it can be seen by double-clicking on each error or right-clicking and choosing View Details.

More detailed results are available by generating an HTML version of the report. The HTML report will organize the errors by CWE, Risk, and Confidence.

For more information see Reviewing Results.

Suppressing False Positives

A number of rules used by the Penetration Testing Tool apply heuristics to analyze the AUT HTTP responses and raise alerts. For this reason, some of the reported errors may be false positive. Review each new error to decide whether it is an actionable item or a false positive (you may want to use the *Confidence* level of the alert to help your assessment).

To prevent the Penetration Testing Tool from reporting a false positive during subsequent runs, you can suppress the error in the in the Quality Tasks view or in the suppression file. See Suppressing Tasks for details.

To suppress an active scan rule for all tests, modify the default active scan policy. See Configuring Scan Policies.

Configuring Scan Policies

SOAtest's penetration testing uses active and passive scan rules to do its analysis. Active scan rules make additional (manipulated) requests to the API to attempt to discover security vulnerabilities. In contrast, passive scan rules make no new requests to the application but instead analyze request/response data captured by the corresponding REST Client or SOAP Client to discover security vulnerabilities. SOAtest leverages OWASP ZAP for penetration testing.

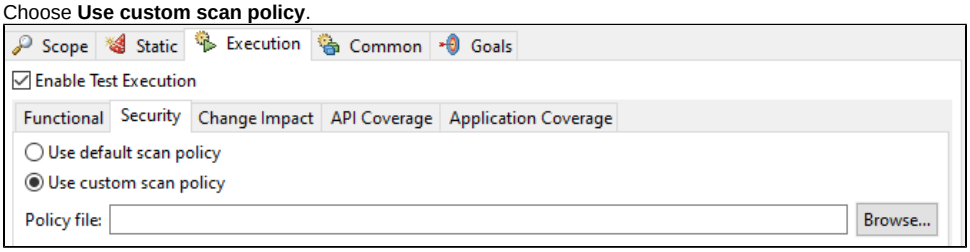
SOAtest contains two built-in penetration test profiles: one for REST APIs, and another for SOAP APIs. Each profile consists of an active scan policy and a set of passive scan rules. By default, a profile is chosen based on whether the configured test makes a request to a REST API or SOAP API.

If you want to use a custom active scan policy, you can do so by exporting a scan policy from OWASP ZAP and configuring SOAtest to use it. The exported scan policy will configure the set of active scan rules that will be used by the Penetration Testing Tool. When using a custom active scan policy, an appropriate set of passive scan rules will be automatically used based on whether the request is made to a REST API or SOAP API.

To configure SOAtest use a custom ZAP scan policy:

1. Select **Parasoft > Test Configuration** to open the Test Configuration Manager.
2. Click **New** to create a new Test Configuration, or select an existing one.
3. Open the Test Configuration's Execution > Security tab.

4. Choose **Use custom scan policy**.



5. Using the **Browse** button select your ZAP .policy file.

Creating Custom Active Scan Policies

You can create or modify a custom active scan policy using the SOAtest embedded ZAP located in the SOAtest ZAP installation directory:

plugins\com.parasoft.ptest.libs.web_<version>\root\zap

(where *version* is the actual version of the com.parasoft.ptest.libs.web plugin. For example, 10.5.2.202109012000)

If you have a ZAP installation on your machine that is independent of SOAtest, you will likely have a ZAP home directory in one of the following locations:

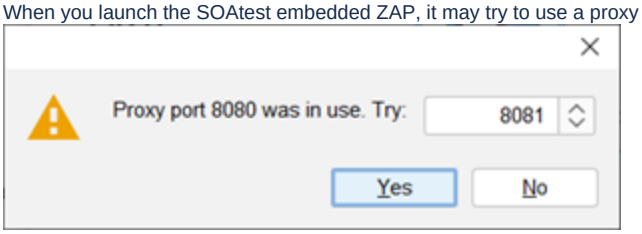
- Windows: C:\Users\<*username*>\OWASP ZAP
- Linux: ~/.ZAP
- Mac: ~/Library/Application Support/ZAP

Your independent ZAP installation user settings stored in these directories may come in conflict with the SOAtest embedded ZAP when you launch it. For this reason, you should launch the SOAtest embedded ZAP with a custom ZAP home directory using the *-dir* command-line argument. Using a command prompt navigate to the SOAtest ZAP installation directory, and launch the SOAtest embedded ZAP with a custom ZAP home directory for example:

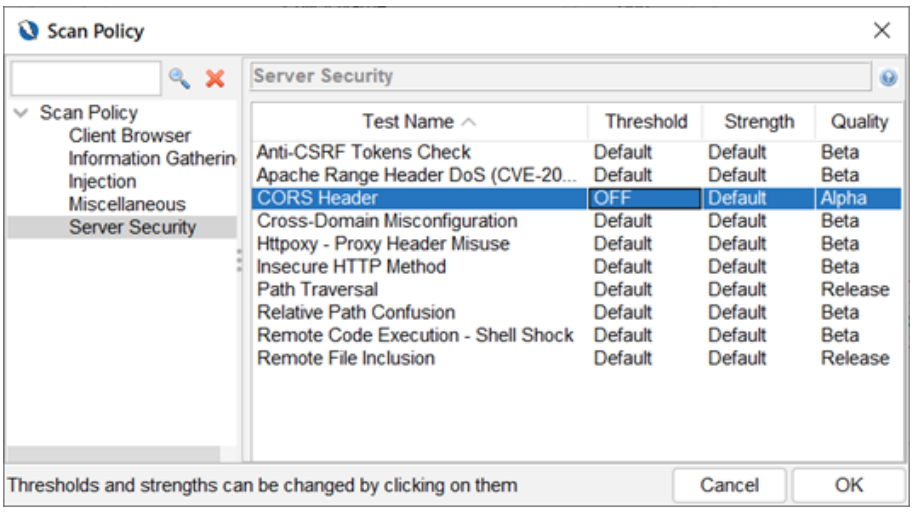
- Windows: zap -dir C:\Users\your_name\ZAP_SOATEST
- Linux: ./zap.sh -dir ~/.ZAP_SOATEST
- Mac: ./zap.sh -dir ~/Library/Application Support/ZAP_SOATEST

If you don't have a ZAP installation on your machine that is independent of SOAtest, you can run the SOAtest embedded ZAP with either zap.bat (Windows) or zap.sh (Linux, Mac) scripts located in the SOAtest ZAP installation directory.

When you launch the SOAtest embedded ZAP, it may try to use a proxy port that is already taken. If so, ZAP will prompt you to select a different port:



To create a new active scan policy or edit an existing one select **Analyze -> Scan Policy Manager** from the ZAP top menu. In the **Scan Policy Manager** di alog select an existing policy or create a new one. In the **Scan Policy** dialog select the active scan rules you would like to apply to your custom policy:



Your custom .policy file will be saved in the policies folder of the ZAP home directory.

Note

If you want to add active scan rules that are not included in the SOAtest embedded ZAP installation, you can add ZAP add-ons that contain these rules to the plugin directory of the SOAtest ZAP installation directory and select those rules in a custom policy as described in this section. However, if you add or modify any ZAP add-ons in the SOAtest ZAP installation directory we can no longer guarantee proper execution of the SOAtest embedded ZAP, so proceed with caution and make necessary backups before proceeding.

Inspecting/Modifying Default Active Scan Policies

You can inspect and/or modify the default active scan policies used by SOAtest. They are located in the following folder of the SOAtest installation:

plugins\com.parasoft.ptest.libs.web_*version*\root\zap\policies

(where *version* is the actual version of the com.parasoft.ptest.libs.web plugin, for example, 10.5.2.202109012000)

This folder contains the following active policy files:

- Parasoft REST.policy – used by Penetration Testing Tools attached to REST Clients.
- Parasoft SOAP.policy – used by Penetration Testing Tools attached to SOAP Clients.

Once you've modified either of these policies, it will be used in the next Penetration Testing Tool invocation.

Penetration Testing Rules Supported

ID	Rule	CWE ID	Risk	Type	Profile
0	Directory Browsing	548	medium	Active	REST/SOAP
2	Private IP Disclosure	200	low	Passive	REST/SOAP
3	Session ID in URL Rewrite	200	medium	Passive	REST/SOAP
6	Path Traversal	22	high	Active	REST/SOAP
7	Remote File Inclusion	98	high	Active	REST
41	Source Code Disclosure - Git	541	high	Active	REST/SOAP
42	Source Code Disclosure - SVN	541	medium	Active	REST/SOAP
43	Source Code Disclosure - File Inclusion	541	high	Active	REST/SOAP
10003	Vulnerable JS Library	829	medium	Passive	REST/SOAP
10009	In Page Banner Information Leak	200	low	Passive	REST/SOAP
10010	Cookie No HttpOnly Flag	1004	low	Passive	REST/SOAP
10011	Cookie Without Secure Flag	614	low	Passive	REST/SOAP
10015	Incomplete or No Cache-control Header Set	525	low	Passive	REST
10017	Cross-Domain JavaScript Source File Inclusion	829	low	Passive	REST/SOAP
10019	Content-Type Header Missing	345	informational	Passive	REST/SOAP
10020	X-Frame-Options Header	1021	medium	Passive	REST/SOAP
10021	X-Content-Type-Options Header Missing	693	low	Passive	REST
10023	Information Disclosure - Debug Error Messages	200	low	Passive	REST/SOAP
10024	Information Disclosure - Sensitive Information in URL	200	informational	Passive	REST/SOAP
10025	Information Disclosure - Sensitive Information in HTTP Referrer Header	200	informational	Passive	REST/SOAP
10026	HTTP Parameter Override	20	medium	Passive	REST/SOAP
10027	Information Disclosure - Suspicious Comments	200	informational	Passive	REST/SOAP
10028	Open Redirect	601	high	Passive	REST/SOAP
10029	Cookie Poisoning	20	informational	Passive	REST/SOAP
10030	User Controllable Charset	20	informational	Passive	REST/SOAP
10031	User Controllable HTML Element Attribute (Potential XSS)	20	informational	Passive	REST/SOAP
10032	Viewstate	642	high, medium, low, informational	Passive	REST/SOAP
10033	Directory Browsing	548	medium	Passive	REST/SOAP
10034	Heartbleed OpenSSL Vulnerability (Indicative)	119	high	Passive	REST/SOAP
10035	Strict-Transport-Security Header	319	low, informational	Passive	REST/SOAP

10036	HTTP Server Response Header	200	low, informational	Passive	REST/SOAP
10037	Server Leaks Information via 'X-Powered-By' HTTP Response Header Field(s)	200	low	Passive	REST/SOAP
10038	Content Security Policy (CSP) Header Not Set	693	medium, informational	Passive	REST/SOAP
10039	X-Backend-Server Header Information Leak	200	low	Passive	REST/SOAP
10040	Secure Pages Include Mixed Content	311	medium, low	Passive	REST/SOAP
10041	HTTP to HTTPS Insecure Transition in Form Post	319	medium	Passive	REST/SOAP
10042	HTTPS to HTTP Insecure Transition in Form Post	319	medium	Passive	REST/SOAP
10043	User Controllable JavaScript Event (XSS)	20	info	Passive	REST/SOAP
10044	Big Redirect Detected (Potential Sensitive Information Leak)	201	low	Passive	REST/SOAP
10045	Source Code Disclosure - /WEB-INF folder	541	high	Active	REST/SOAP
10047	HTTPS Content Available via HTTP	311	low	Active	REST/SOAP
10048	Remote Code Execution - Shell Shock	78	high	Active	REST/SOAP
10049	Content Cacheability	524	informational	Passive	REST
10050	Retrieved from Cache	Unspecified	informational	Passive	REST/SOAP
10052	X-ChromeLogger-Data (XCOLD) Header Information Leak	200	medium	Passive	REST/SOAP
10054	Cookie without SameSite Attribute	1275	low	Passive	REST/SOAP
10055	CSP	693	medium, low, informational	Passive	REST/SOAP
10056	X-Debug-Token Information Leak	200	low	Passive	REST/SOAP
10057	Username Hash Found	284	informational	Passive	REST/SOAP
10061	X-AspNet-Version Response Header	933	low	Passive	REST/SOAP
10062	PII Disclosure	359	high	Passive	REST/SOAP
10063	Permissions Policy Header Not Set	16	low	Passive	REST/SOAP
10070	Use of SAML	Unspecified	informational	Passive	REST/SOAP
10094	Base64 Disclosure	200	high, informational	Passive	REST/SOAP
10095	Backup File Disclosure	530	medium	Active	REST/SOAP
10096	Timestamp Disclosure	200	informational	Passive	REST/SOAP
10097	Hash Disclosure	200	high, low	Passive	REST/SOAP
10098	Cross-Domain Misconfiguration	264	medium	Passive	REST/SOAP
10099	Source Code Disclosure	540	medium	Passive	REST/SOAP
10103	Image Location and Privacy Scanner	200	informational	Passive	REST/SOAP
10105	Weak Authentication Method	287	high, medium	Passive	REST/SOAP
10106	HTTP Only Site	311	medium	Active	REST/SOAP
10107	Httpoxy - Proxy Header Misuse	20	high	Active	REST/SOAP
10108	Reverse Tabnabbing	Unspecified	medium	Passive	REST/SOAP
10109	Modern Web Application	Unspecified	informational	Passive	REST/SOAP
10110	Dangerous JS Functions	749	low	Passive	REST/SOAP
10202	Absence of Anti-CSRF Tokens	352	low, informational	Passive	REST/SOAP
20015	Heartbleed OpenSSL Vulnerability	119	high	Active	REST/SOAP
20016	Cross-Domain Misconfiguration	264	high	Active	REST/SOAP
20017	Source Code Disclosure - CVE-2012-1823	20	high	Active	REST/SOAP
20018	Remote Code Execution - CVE-2012-1823	20	high	Active	REST/SOAP
20019	External Redirect	601	high	Active	REST
30001	Buffer Overflow	120	medium	Active	REST/SOAP
30002	Format String Error	134	medium	Active	REST/SOAP
30003	Integer Overflow Error	190	medium	Active	REST
40003	CRLF Injection	113	medium	Active	REST
40008	Parameter Tampering	472	medium	Active	REST/SOAP

40009	Server Side Include	97	high	Active	REST
40012	Cross Site Scripting (Reflected)	79	high	Active	REST
40013	Session Fixation	384	high	Active	REST/SOAP
40014	Cross Site Scripting (Persistent)	79	high	Active	REST
40015	LDAP Injection	90	high	Active	REST/SOAP
40016	Cross Site Scripting (Persistent) - Prime	79	informational	Active	REST
40017	Cross Site Scripting (Persistent) - Spider	79	informational	Active	REST
40018	SQL Injection	89	high	Active	REST/SOAP
40025	Proxy Disclosure	200	medium	Active	REST/SOAP
40028	ELMAH Information Leak	215	medium	Active	REST/SOAP
40029	Trace.axd Information Leak	215	medium	Active	REST/SOAP
40032	.htaccess Information Leak	215	medium	Active	REST/SOAP
40034	.env Information Leak	215	medium	Active	REST/SOAP
40035	Hidden File Finder	538	medium	Active	REST/SOAP
40038	Bypassing 403	Unspecified	medium	Active	REST/SOAP
40039	Web Cache Deception	Unspecified	medium	Active	REST/SOAP
40040	CORS Header	942	high, medium, informational	Active	REST
90001	Insecure JSF ViewState	642	medium	Passive	REST/SOAP
90002	Java Serialization Object	502	medium	Passive	REST/SOAP
90003	Sub Resource Integrity Attribute Missing	345	medium	Passive	REST/SOAP
90004	Insufficient Site Isolation Against Spectre Vulnerability	693	low	Passive	REST/SOAP
90011	Charset Mismatch	436	informational	Passive	REST/SOAP
90017	XSLT Injection	91	medium	Active	REST/SOAP
90019	Server Side Code Injection	94	high	Active	REST/SOAP
90020	Remote OS Command Injection	78	high	Active	REST/SOAP
90021	XPath Injection	643	high	Active	REST/SOAP
90022	Application Error Disclosure	200	medium	Passive	REST/SOAP
90023	XML External Entity Attack	611	high	Active	REST/SOAP
90024	Generic Padding Oracle	209	high	Active	REST/SOAP
90028	Insecure HTTP Method	200	medium	Active	REST/SOAP
90030	WSDL File Detection	Unspecified	informational	Passive	REST/SOAP
90033	Loosely Scoped Cookie	565	informational	Passive	REST/SOAP
90034	Cloud Metadata Potentially Exposed	Unspecified	high	Active	REST/SOAP
110001	Application Error Disclosure via WebSockets	209	medium	Passive	REST/SOAP
110002	Base64 Disclosure in WebSocket message	Unspecified	informational	Passive	REST/SOAP
110003	Information Disclosure - Debug Error Messages via WebSocket	200	low	Passive	REST/SOAP
110004	Email address found in WebSocket message	200	informational	Passive	REST/SOAP
110005	Personally Identifiable Information via WebSocket	359	high	Passive	REST/SOAP
110006	Private IP Disclosure via WebSocket	Unspecified	low	Passive	REST/SOAP
110007	Username Hash Found in WebSocket message	284	informational	Passive	REST/SOAP
110008	Information Disclosure - Suspicious Comments in XML via WebSocket	200	informational	Passive	REST/SOAP
111001	HTTP Verb Tampering (Parasoft proprietary rule)	287	medium	Active	REST

Integration with Burp Suite

SOAtest uses a preconfigured instance of OWASP ZAP under the hood to perform penetration testing. You also have the option to use the commercial tool Burp Suite for penetration testing by leveraging the extension https://docs.parasoft.com/display/SOA20211/Burp+Suite+Extensions+1.0.