RPA for security software testing automation

Bejan Şerban, Euro-Testing Software Solutions

Abstract

RPA is changing the way IT works and software testing automation is the next area that will be shaken by robots. RPA's technology provided and continues to provide significant advantages over more elementary automation tools as being code-free and non-disruptive. This paper provides detailed examples on using Ui-Path for automatization of security testing web applications.

1 Introduction

RPA tool suite is not specifically related to test automation, these tools are used for automating business processes which are often performed by end user by using a diverse variety of systems. Testing scenario are based on end user interface and entering different use cases. RPA does the same thing which again is configured to interact or act as an end user to convey repetitive jobs.

RPA Benefits [Bhu19]:

- Code-less: No need to memorize any syntax.
- Simplicity: Easy to create a process through simple drag and drop.
- Scalability: It can be achieved by assigning work to multiple workstations.
- Cost saving: Huge reduction in cost as very minimal workforce is required.
- Accuracy: As the tasks are performed by the bots.
- Productivity: As it is robotic, productivity will be very high.
- Flexibility: Test process is not depend on type of software under test, whether it is web based, desktop application or mobile application.

1.1 Differences between Test Automation and RPA

There is multiple overlaps between a Test Automation Tool and RPA tool. For instance, they both drive screen, keyboard, mouse, etc. and have similar tech architecture. But following are the key differences between the two

Tab. 1: Tab description: This is the Style for the table caption [Gur19]

Parameter	Test Automation	RPA
Goal	Reduce Test execution time through automation	Reduce headcount through automation
Task	Automate repetitive Test Cases	Automate repetitive Business processes

Coding	Coding knowledge required to create Test Scripts	Wizard-driven, and coding knowledge not required
Tech Approach	Supports limited software environment. Example: Selenium can support only web applications.	Supports a wide array of software environments
Example	Test cases are automated	Data entry, forms, loan processing, is automated
Application	Test Automation can be run on QA, Production, Performance, UAT environments	RPA is usually run only on production environments
Implementation	It can automate a product.	It can automate a product as well as a service.
Users	Limited to technical users.	Can be used across the board by all stakeholders.
Role	Acts as a virtual assistant.	Acts as a virtual workforce.
AI	Can execute only what is coded.	Many RPA tools come with an AI engine can process information like a human

1.2 Reversing The Testing Pyramid

RPA tools are challenging the model of the testing pyramid by enabling business focussed GUI (graphical user interface) automation both on-demand and scheduled in nightly builds. RPA solves challenges of GUI automation such as testing if an icon was highlighted or not, or CSS elements having locations other than previously specified. [Ott19]

In one project described in [Ott19] was found that RPA scripts were highly effective for a regression test suite, for test scenarios with many variations and for managing test data. They could build RPA building blocks that reduced 40 clicks to one “shared function” that could then be parameterized based on the specific test case. Compared to previous testing with no RPA tool, the tests were faster to run and prepare already for the second run. Usually, a rule-of-thumb has been that the tests should run 4 times before break even. Tests with many variations saved time within the first execution using RPA

as compared to having a business subject matter experts run the execution of the test again.

In this paper, we first explain the experimental environment used to validate that RPA tools can be used in automating security testing[Dou10] following a blackbox approach, similar to other Dynamic Application Security Testing (DAST) tools. We then described the process design for each test we implemented and presented the results we obtained by using the implemented testing processes on two web applications.

Section 2 explain the experimental setup, while Section 3 presents the process design. We present our results in Section 4 and conclude in Section 5 .

2 Experimental Evironment

Our objective was to implement an automated security testing framework based on RPA and assess the required knowledge, ease of use and effectivnes. In order to validate this objective we set-up an experimental environment based on DVWA, XVWA and UiPath REFramework.

Damn Vulnerable Web Application (DVWA) is a PHP/MySQL web application that has already baked common security vulnerabilities, with various level of difficulty. One of its goals is to be an aid for security professionals to test their tools. [Dew12]

DVWA embeds several vulnerabilities, notably vulnerabilities of the kind SQL Injection and Blind SQL Injection, and Reflected and Stored XSS. These vulnerabilities are commonly used to attack current Web applications [Tak08] . Each vulnerability has a dedicated menu item leading to a dedicated page. DVWA also embeds three security levels: low, medium, and high. Each level carries different security protections: the lowest level has no protection at all, the medium level is a refined version but is still quite vulnerable, and the highest level is a properly secured version. Users can choose which level they want to work with by specifying it through the application. It is also possible to view and compare the source code of each security level.

XVWA is a badly coded web application written in PHP/MySQL that helps security enthusiasts to learn application security. XVWA is designed to understand following security issues: SQL Injection (Error Based and Blind), OS Command Injection, XPATH Injection, Formula Injection, PHP Object Injection, Unrestricted File Upload, Reflected Cross Site Scripting, Stored Cross Site Scripting, DOM Based Cross Site Scripting, Server Side Request Forgery (Cross Site Port Attacks), File Inclusion, Session Issues, Insecure Direct Object Reference, Missing Functional Level Access Control, Cross Site Request Forgery (CSRF), Cryptography, Unvalidated Redirect & Forwards, Server Side Template Injection. [San17]

Based on [Cla18] and mainly adevantage in “Bot development and core functions”, we choosed UiPath as RPA tool, specifficaly using already developed UiPath REFramwework.

UiPath

UiPath, founded in 2005, is a software company originating from Romania that specializes in RPA. Their flagship product is called UiPath platform. It consists of three components, those being UiPath Studio, UiPath Robot, and UiPath Orchestrator. UiPath Studio is a software that provides configured tools and automation library to develop a robot without coding. Developers can apply VB.NET or C#.NET as a programming language to develop new functions and generate reusable contents for library for their own benefits.

UiPath REFramework

The framework is meant to be a template that helps the user design processes that offer, at a barebones minimum, a way to store, read, and easily modify project configuration data, a robust exception handling scheme and event logging for all exceptions and relevant transaction information. The main purpose of the framework is to solve a collection of business transactions.

3 Process Design

We designed the following processes that aim to identify some type of security vulnerability:

3.1 Command Injection Attack

The purpose of the command injection attack is to inject and execute commands specified by the attacker in the vulnerable application. In situation like this, the application, which executes unwanted system commands, is like a pseudo system shell, and the attacker may use it as any authorized system user. However, commands are executed with the same privileges and environment as the web service has.

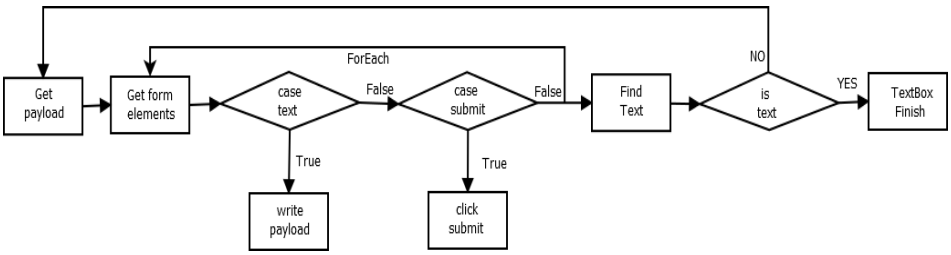

Fig. 1: Logic diagram of Command Injection testing process

The robot loads all the payload it has to test from a prepopulated CSV file using special crafter payloads using Commix Project (https://github.com/commixproject/commix), an automated tool used by security researchers in order to test web-based applications with the view to find bugs, errors or vulnerabilities related to command injection attacks. The

CSV has two columns, the first one is the payload and the second one is the output the command will generate in case the attack is succsesfull.
Secondly, the robot identify all the elements of the form in the page and for each payload will do the following stepts:

- in case it finds an UI element with type text it will type into it the payload, else if it finds one with type submit it will click it;
- clicking the submit element will make a request to server, after the new page loads the robot will check if the expected result (second column of the CSV file) is present on the page;
- if it finds the elemnent it will log the result, will pop a message box with the vulnerability it found and it will break the process.

3.2 Server Side Request Forgery (SSRF) Attack

The target application may have functionality for importing data from a URI, publishing data to a URI or otherwise reading data from a URI that can be tampered with. In SSRF, the attacker modifies the calls to this functionality by supplying a completely different URI or by manipulating how URIs are built (path traversal etc.). When the manipulated request goes to the server, the server-side code picks up the manipulated URL and tries to read data to the manipulated URI. By selecting target URIs the attacker may be able to read data from services that are not directly exposed on the internet.

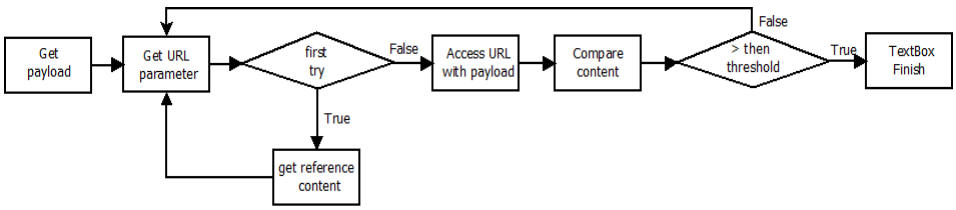

Fig. 2: Logic diagram of SSRF testing process

The robot loads all the payload it has to test from a prepopulated CSV file using special crafter payloads using PayloadsAllTheThings GitHub repository (https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/Server%20Side%20Request%20Forgery).
Secordly, the robot gets the page URI and parse it, removing the value of the parameter and store in a variable the number of words from the page for comparison purposes. For each payload will do the following:

- request a new URI where the value for the parameter is the payload;
- compare the number of words for the newly requested page with the number of words stored in the variable;

- if the number of words from the new page is significantly higher means the access is successful and we could access other content. The robot will log the result, will pop a message box with the vulnerability it found and it will break the process.

3.3 Cross-Site Request Forgery (CSRF) Attack

CSRF is an attack that forces an end user to execute unwanted actions on a web application in which they are currently authenticated. Using social engineering (such as sending a link via email/chat), an attacker may force the users of a web application to execute actions of the attacker's choosing.

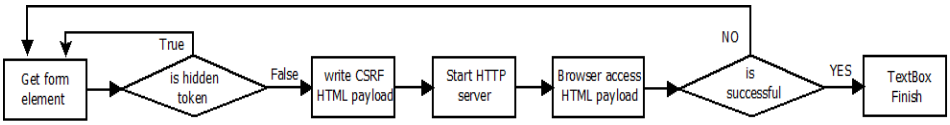

Fig. 3: Logic diagram of CSRF testing process

Tthe robot identify all the elements of the form in the page and check if any of the item has the type equal to hidden. If no element has this type, the robot will do the following:

- it will generate a new HTML file as a self submitting form with the same elements as the form in the original page as a CSRF proof-of-concept.
- it will start a nginx server that can serve the HTML file generated before;
- it will start a new browser and access the HTML file;
- check if the form action has been performed successfully;
- if the action was been realised it will log the result, will pop a message box with the vulnerability it found and it will break the process.

3.4 File Inclusion Attack

Some web applications allow the user to specify input that is used directly into file streams or allows the user to upload files to the server. At a later time the web application accesses the user supplied input in the web applications context. If the file chosen to be included is local on the target machine, it is called Local File Inclusion (LFI)

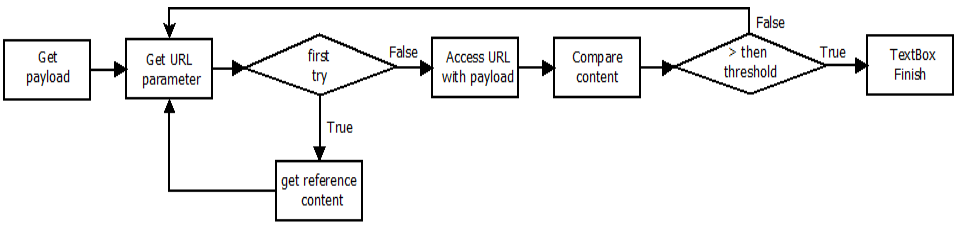

Fig. 4: Logic diagram of File Inclusion testing process

Process is very similar with the one used for SSRF testing, big part of the process was reused, the difference is hte CSV file used, in this tests were used payloads specially crafter for LFI attacks from PayloadAllTheThings Github repository (https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/File%20Inclusion).

3.5 PHP Object Injection Attack

PHP Object Injection is an application level vulnerability that could allow an attacker to perform different kinds of malicious attacks, such as Code Injection, SQL Injection, Path Traversal and Application Denial of Service, depending on the context. The vulnerability occurs when user-supplied input is not properly sanitized before being passed to the unserialize() PHP function. Since PHP allows object serialization, attackers could pass ad-hoc serialized strings to a vulnerable unserialize() call, resulting in an arbitrary PHP object(s) injection into the application scope.

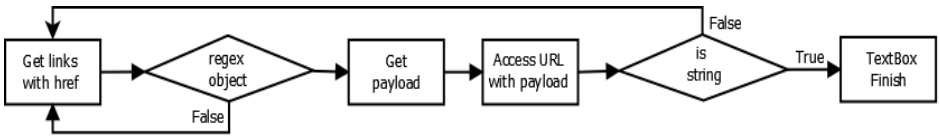

Fig. 5: Logic diagram of Object Injection testing process

This process was tested only against XVWA, DVWA doesn't have this type of vulnerability. First the robot loads all the payload it has to test from a prepopulated CSV file using special crafter payloads using PayloadsAllTheThings GitHub repository (https://github.com/swisskyrepo/PayloadsAllTheThings/blob/master/Insecure%20Deserialization/PHP.md).

Secordly, the robot gets all the elements in the page with a "href" attribute. For each elemenet will do the following:

- parse the value of the href and get the value of the parameter;
- apply a special crafter regex (Listing 1.) to identify PHP objects:

Listing 1. PHP Object RegEx

```
"^o:\d+:""[a-z0-9_]+"":\d+:{.*?}$"
```

- if the value pass the regex, we make a request changing the object with each payload from CSV file;
- if it finds the text corresponding to injected object it will log the result, will pop a message box with the vulnerability it found and it will break the process.

3.6 Sql Injection Attack

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, modify database data (insert/update/delete), execute administration operations on the database (such as shutdown the DBMS), recover the content of a given file present on the DBMS file system (load_file) and in some cases issue commands to the operating system.

When an attacker executes SQL injection attacks, sometimes the server responds with error messages from the database server complaining that the SQL query's syntax is incorrect. Blind SQL injection is identical to normal SQL Injection except that when an attacker attempts to exploit an application, rather then getting a useful error message, they get a generic page specified by the developer instead. This makes exploiting a potential SQL Injection attack more difficult but not impossible. An attacker can still steal data by asking a series of True and False questions through SQL statements, and monitoring how the web application response (valid entry retunred or 404 header set).

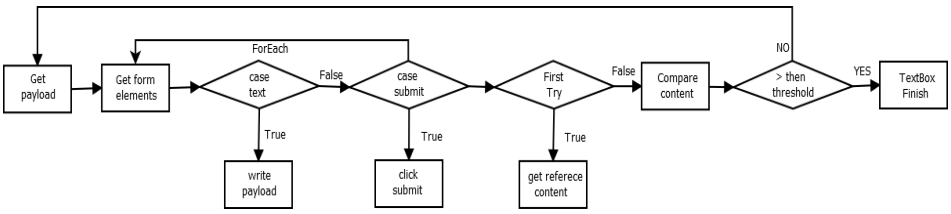

Fig. 6: Logic diagram of SQL Injection testing process

For both types of SQL injection, the robot loads all the payload it has to test from a prepopulated CSV file using special crafter payloads from SecLists GitHub repository (https://github.com/danielmiessler/SecLists/tree/master/Fuzzing/SQLi).

Secordly, the robot identify all the elements of the form in the page and for each payload will do the following stepts:

- in case it finds a dropdown element will inject the JS script from Listing 2. that adds another entry in the list equals with the payload from the CSV file and it will selected the newly added value;
- in case it finds an UI element with type text it will type into it the payload, else if it finds one with type submit it will click it;
- clicking the submit element will make a request to server, after the new page loads the robot will check if exploit was successful:

  SQL injection will compare the word count from the initial request with the one after the test and if the new value is significantly higher it mark the vulnerability;

Blind SQL injection will compare the time needed to load the new page, in case the page loads in more than 2500ms it mark the vulnerability;

- if the vulnerability is marked it will log the result, will pop a message box with the vulnerability it found and it will break the process.

Listing 2. JS script used to inject a new element in a drop down list

```
function (element, input) {
   var option = document.createElement("option");
   option.text = input;
   element.add(option, element[0]);
   element.value = input;
}
```

3.7 Cross-site Scripting (XSS) Attack

"Cross-Site Scripting (XSS)" attacks are a type of injection problem, in which malicious scripts are injected into the otherwise benign and trusted web sites. We implemented three similar processes that target the following types of XSS:

1. **Reflected XSS.** the malicious code is not stored in the remote web application, in order to trigger this vulnerability it requires some social engineering (such as a link via email/chat).
2. **Stored XSS**. The XSS is stored in the database. The XSS is permanent, until the database is reset or the payload is manually deleted..
3. **DOM XSS.** DOM Based XSS is a special case of reflected XSS where the JavaScript is hidden in the URL and pulled out by JavaScript in the page while it is rendering rather than being embedded in the page when it is served.

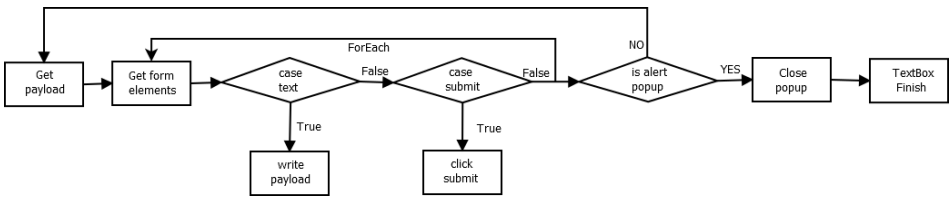

Fig. 7: Logic diagram of XSS testing process

The structure for all tree processes is similar with small changes. First the robot loads all the payload it has to test from a prepopulated CSV file using special crafter payloads from SecLists GitHub repository (https://github.com/danielmiessler/SecLists/tree/master/Fuzzing/XSS).

For reflected and stored XSS the robot gets the UI elements from the page and in case it finds an UI element with type text it will type into it the payload, else if it finds one with type submit it will click it. For DOM XSS the robot gets the page URL and parse it and replace the value of the parameter with the payload.

To check if the payload was successful, the robot check if a new alert textbox exists. For stored XSS first it will request again the page where the payload is stored.If a new alert box is present, the robot closes it and will log the result, will pop a message box with the vulnerability it found and it will break the process.

4 Results

Overall we managed to develop the framework in five days, without any prior knowledge of RPA and we successfully tested using both DVWA and XVWA. Our results are presented in Tab. 2:

Tab. 2: Tab description: This is the Style for the table caption

Vulnerability	DVWA - low	DVWA - medium	DVWA - high	XVWA
Command Injection	yes	yes	yes	yes
SSRF	yes	no	no	yes
CSRF	yes	yes	no	yes
File Inclusion	yes	yes	no	yes
PHP Object Injection	N/A	N/A	N/A	yes
Sql Injection	yes	yes	no	yes
Blind sql Injection	yes	yes	no	yes
Reflected XSS	yes	yes	yes	yes
Stored XSS	yes	yes	yes	yes
DOM XSS	yes	yes	no	yes

5 Conclusion

RPA can power business testing, and that can affect the amount of testing done by the professional testing team including security testing. Some testers might have to rearrange their work when RPA powers the business testers to do more.

Our proof-of-concept framework for security testing performed well in the tests on DVWA, XVWA, proving RPA's potential in QA testing and even in security testing for some niche cases.

RPA will play an important role in the QA development and provide businesses with an opportunity to add more value to their underlying stability and scalability models with less investment and sweat.

References

[Bhu19] S.Bhukan: “Robotic Process Automation and The Testing future” - http://mihir-bhatt.strikingly.com/blog/robotic-process-automation-and-the-testing-future. 2019

[Gur19] “Robotic Process Automation (RPA) Tutorial: What is, Tools & Example” - https://www.guru99.com/robotic-process-automation-tutorial.html. 2019

[Ott19] J.Ottosen: “Robot Process Automation As A Power Tool For Testing” - https://www.ministryoftesting.com/dojo/lessons/rpa-as-a-power-tool-for-testing, 2019

[Dou10] Doupé, A., Cova, M., Vigna, G.: Why Johnny can’t pentest: An analysis of black-box web vulnerability scanners. In: Kreibich, C., Jahnke, M. (eds.) DIMVA 2010. LNCS, vol. 6201, pp. 111–131. Springer, Heidelberg 2010

[Dew12] Dewhurst R.Damn Vulnerable Web Application (DVWA). 2012

[Tak08] Takanen, J. DeMott, and C. Miller, Fuzzing for Software Security Testing and Quality Assurance. Norwood, MA, USA: Artech House, Inc., 2008

[San17] T. Sanoop, Xtreme Vulnerable Web Application (XVWA), 2017

[Cla18] Clair, C.L., Cullen A. & King, M. 2017. The Forrester Wave: Robotic Process Automation, Q1 2017. https://www.edgeverve.com/wp-content/uploads/2017/02/forrester-wave-robotic-processautomation.pdf 2018