

Security Tool Demo Report

Kaiyu Dong, Long Chen, Shaolun Liu

Simon Fraser University
{kaiyu.dong,long_chen.3,shaolun.liu}@sfu.ca

Keywords: Arachni · Vulnerability Scan · DAST

1 Introduction

Arachni is an open-source scanner that focuses on the recon phase of our penetration testing in a different manner than any other tool out there. The creator of the tool, Tasos Laskos developed the tool to address a couple of opposed goals. First, scans can take an excessive amount of time (many hours to even weeks) and as such makes these scans less-than-helpful. That time is lost and makes testing a more drawn-out process. Second, more data and coverage is a good thing, as it enhances accuracy, but it also adds additional time to the test process to complete the necessary transactions.

Laksos developed Arachni to both reduce the amount of time for a scan while allowing the tool to scale such that it is able to process more test vectors efficiently. The timing was improved by employing asynchronous HTTP requests to do the tool's bidding, and by allowing Arachni to scale across a cluster of systems so that they are all processing in parallel. The accuracy goal was enhanced by open-sourcing the tool and using a Ruby framework that can allow anyone to add new or better-optimized tests to the tool. Together, these enhancements make Arachni a formidable scan tool that we should all dive deeper into for improved testing efficacy and timing.

Arachni differs from other scanners in that it makes heavy use of asynchronous HTTP to conduct its scans. This allows each Instance to send HTTP requests to targets in parallel. Because you can have multiple scans in progress at the same time, it greatly improves the speed with which we can go from initiation to completion. Most other scanners spend considerable time waiting on scan threads to finish. This is made more egregious given that these scanners often run a set array of scans, and do not tailor their scan list on the fly like Arachni.

2 Using Arachni

The Arachni system is implemented as a series of batch files. Initializing a total scan of a Web asset launches all of the batch files in sequence, each executing a different test vector. This is a series of pen testing attempts covering a full range of exploits, including HTTP GET attacks, SQL injection, code injection, cross-site scripting, and so on. There are a total of 40 modules that provide a

set of passive checks and active attempts. Each module will run through many permutations of possible inputs to test its particular topic of inquiry. All of this work can take a long time to run.

Running the system from the command line results in a text-based report displayed on the screen.

```
[~] http://example.com/

[~] Total: 1
[+] Without issues: 0
[-] With issues: 1 ( 100% )

[~] Arachni is heading towards obsolescence, try out its next-gen successor Ecsypno SCNR:
[~] https://ecsypno.com/

[~] Report saved at: /home/kda78/arachni-1.6.1.3-0.6.1.1/bin/example.com 2022-10-30 23_18_30 +0000.afr [0.01MB]

[~] Audited 1 page snapshots.

[~] Duration: 00:00:04
[~] Processed 618/618 HTTP requests.
[~] -- 309.279 requests/second.
[~] Processed 0/0 browser jobs.
[~] -- 0 second/job.

[~] Currently auditing      http://example.com/
[~] Burst response time sum  0.34 seconds
[~] Burst response count     15
[~] Burst average response time 0.023 seconds
[~] Burst average           18.702 requests/second
[~] Timed-out requests       0
[~] Original max concurrency 10
[~] Throttled max concurrency 10
```

Fig. 1. Running Arachni from the command line

This report is also saved to an AFR file. This is a native binary format – AFR stands for Arachni Framework Report. The file can be converted into various formats, such as XML, plain text, and HTML.

The other method of access to the scanner is through its Web-based GUI. To get the GUI running, you need to open a Command Prompt window, go to the home directory of Arachni on your computer, and enter `bin arachni_web`. You will see a message telling you that the listener is running on `localhost:<port>` - you will see a number instead of `<port>`. Go to your browser and enter that address in the address bar (e.g., `localhost:9292`). This opens the Web interface. You need to log in, giving the username `admin@admin.admin` and password `administrator`.

This leads to the launch page for the scanner, which gives you a range of options. I left all of the defaults for a test and specified a Direct mode scan of the Arachni website at `https://demo.testfire.net/`.

As the test run progresses, a stream of feedback messages will pass by in the Command Prompt window.

The Web interface gives a live progress report.

The lower part of the scan report shows the number of problems encountered by type. Clicking on a category gets an explanation of the kind.

```

I, [2022-10-30T21:21:15.528214 #45522] INFO -- Call: framework.progress [127.0.0.1]
I, [2022-10-30T21:21:15.528562 #45525] INFO -- Call: framework.progress [127.0.0.1]
I, [2022-10-30T21:21:20.529683 #45519] INFO -- Call: service.native_progress [127.0.0.1]
I, [2022-10-30T21:21:20.538872 #45523] INFO -- Call: framework.progress [127.0.0.1]
I, [2022-10-30T21:21:20.539642 #45522] INFO -- Call: framework.progress [127.0.0.1]
I, [2022-10-30T21:21:20.540612 #45525] INFO -- Call: framework.progress [127.0.0.1]
I, [2022-10-30T21:21:25.546388 #45519] INFO -- Call: service.native_progress [127.0.0.1]
I, [2022-10-30T21:21:25.555187 #45523] INFO -- Call: framework.progress [127.0.0.1]
I, [2022-10-30T21:21:25.555372 #45522] INFO -- Call: framework.progress [127.0.0.1]
I, [2022-10-30T21:21:25.555805 #45525] INFO -- Call: framework.progress [127.0.0.1]
I, [2022-10-30T21:21:30.563580 #45519] INFO -- Call: service.native_progress [127.0.0.1]
I, [2022-10-30T21:21:30.570930 #45523] INFO -- Call: framework.progress [127.0.0.1]
I, [2022-10-30T21:21:30.571276 #45522] INFO -- Call: framework.progress [127.0.0.1]

```

Fig. 2. Feedback messages in the Command Prompt window

TOGGLE VISIBILITY OF

- Comments
- Statistics

ACTIONS

- Share
- Edit schedule
- Full edit

https://petstore.octoperf.com/

petstore

Edit description

Scanning

Currently auditing:

- https://petstore.octoperf.com/actions/Catalog.action
- https://petstore.octoperf.com/actions/Catalog.action
- https://petstore.octoperf.com/actions/Catalog.action
- https://petstore.octoperf.com/actions/Catalog.action

Issues [37]

Issues may be missing some context while the scan is running. You better wait until the scan is over to review them as the meta-analysis phase will flag probable false-positives and other untrusted issues accordingly.

All [37] Fixed [0] Verified [0] Pending verification [0] False positives [0] Awaiting review [0]

Listing all logged issues.

TOGGLE BY SEVERITY

Reset Show all Hide all

Low 3

Informational 34

NAVIGATE TO

URL	Input	Element
Common sensitive file	1	
Missing 'X-Frame-Options' header	2	
Allowed HTTP methods	2	
Interesting response	31	
Insecure cookie	1	
Common sensitive file	1	
Missing 'X-Frame-Options' header	2	

Fig. 3. The live progress report in the Web interface

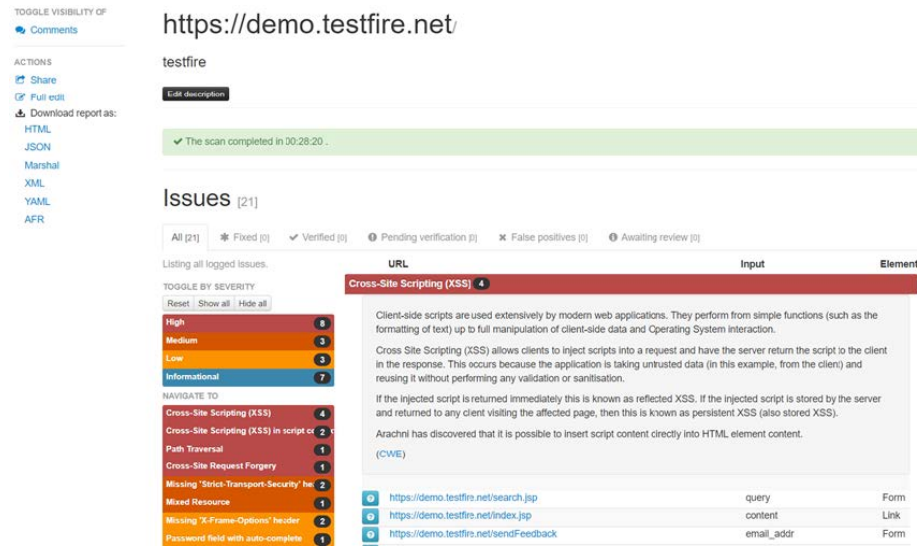


Fig. 4. The number of problems encountered by type

Although creating the scan results in the AFR format is an obscure choice, the utility `arachni-reporter` can convert that report into HTML format. The results are impressive. Here are the results produced by a scan of the `arachni-scanner.com` site.

The front tab of the report shows interactive charts based on the scanner's findings.

An Issues tab shows each of the vulnerabilities that were identified.

Further details on each problem can be found by clicking on the line in the Issues report. The system also explains the error and a description of the exploit as given by OWASP.

3 Summary

Arachni is a series of penetration testing exercises implemented as batch files and strung into a single chain that can be launched with one command. However, a vulnerability scanner is precisely that – an automated pen-testing session.

Effective penetration testing is contingent on many things. As current events have revealed, the sheer number of vectors available for attackers to leverage demands comprehensive test suites that automate scans and help us quickly ascertain the risk exposure of a target. Almost as important is the ability to process the copious amounts of raw data and turn it around as actionable intelligence.

Arachni is a fantastic tool to provide these scans, and as an added bonus can form the basis of our detailed reports or deliverables. Because Arachni is an open source and extensible product, it is well supported by the community and offers

Summary

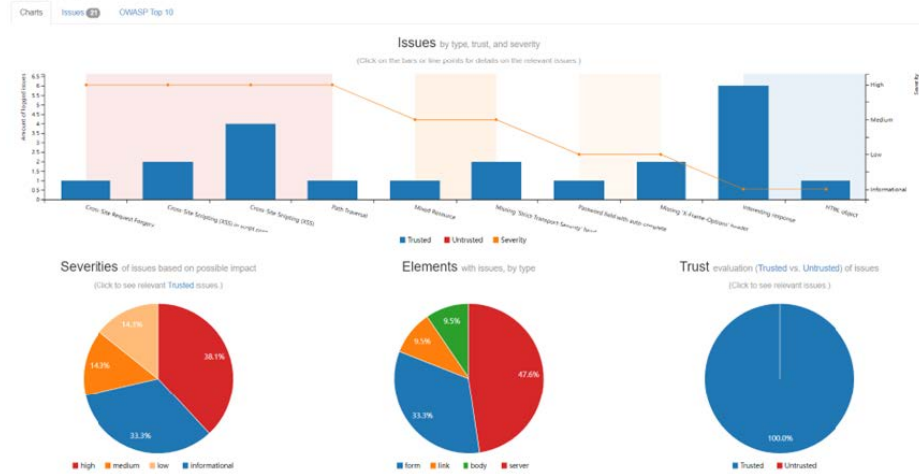


Fig. 5. Arachni's report formats are production-ready and a great start on building your deliverables to the customer

Summary

Charts Issues 49 OWASP Top 10

Trusted 49
High severity 4

Cross-Site Request Forgery (CSRF)

In the majority of today's web applications, clients are required to submit forms which can perform sensitive operations.

An example of such a form being used would be when an administrator wishes to create a new user for the application.

In the simplest version of the form, the administrator would fill-in:

- Name
- Password
- Role (level of access)

Continuing with this example, Cross Site Request Forgery (CSRF) would occur when the administrator is tricked into clicking on a link, which if logged into the application, would automatically submit the form without any further interaction.

Cyber-criminals will look for sites where sensitive functions are performed in this manner and then craft malicious requests that will be used against clients via a social engineering attack.

There are 3 things that are required for a CSRF attack to occur:

1. The form must perform some sort of sensitive action.
2. The victim (the administrator the example above) must have an active session.
3. Most importantly, all parameter values must be known or guessable.

Arachni discovered that all parameters within the form were known or predictable and therefore the form could be vulnerable to CSRF.

Manual verification may be required to check whether the submission will then perform a sensitive action, such as *reset a password*, *modify user profiles*, *post content on a forum*, etc.

Vector type	HTTP method	Action
form	GET	https://demo.testfire.net/doSubscribe

Cross-Site Scripting (XSS) in script context

Client-side scripts are used extensively by modern web applications. They perform from simple functions (such as the formatting of text) up to full manipulation of client-side data and Operating System interaction.

Cross Site Scripting (XSS) allows clients to inject scripts into a request and have the server return the script to the client in the response. This occurs because the application is taking untrusted data (in this example, from the client) and reusing it without performing any validation or sanitisation.

If the injected script is returned immediately this is known as reflected XSS. If the injected script is stored by the server and returned to any client visiting the affected page, then this is known as persistent XSS (also stored XSS).

Arachni has discovered that it is possible to force the page to execute custom JavaScript code.

Vector type	Input name	HTTP method	Action
form	name	GET	https://demo.testfire.net/sendFeedback
form	query	GET	https://demo.testfire.net/search.jsp

Cross-Site Scripting (XSS)

Client-side scripts are used extensively by modern web applications. They perform from simple functions (such as the formatting of text) up to full manipulation of client-side data and Operating System interaction.

Cross Site Scripting (XSS) allows clients to inject scripts into a request and have the server return the script to the client in the response. This occurs because the application is taking untrusted data (in this example, from the client) and reusing it without performing any validation or sanitisation.

If the injected script is returned immediately this is known as reflected XSS. If the injected script is stored by the server and returned to any client visiting the affected page, then this is known as persistent XSS (also stored XSS).

Arachni has discovered that it is possible to insert script content directly into HTML element content.

Fig. 6. Detailed Vulnerability information can help quickly educate the customer or set up the next phases

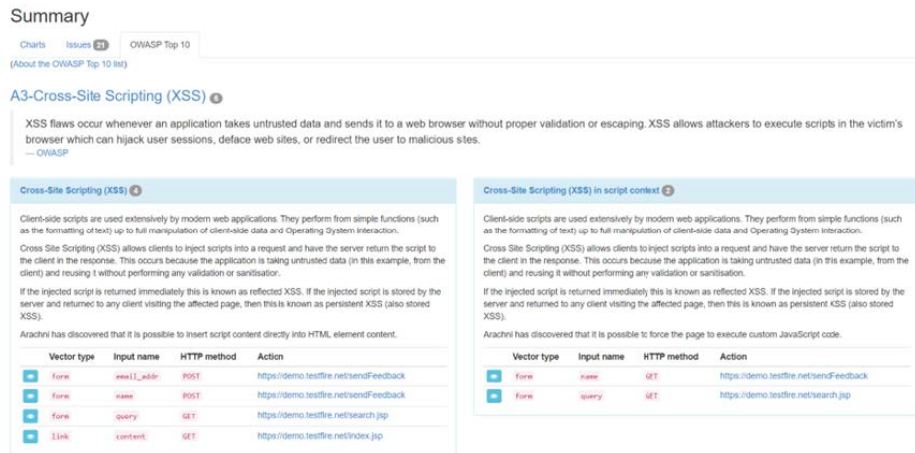


Fig. 7. The description of the exploit given by OWASP

an Ruby framework on which anyone can graft their own extensions or plugins. This system was developed as a student project by one person and promised to be a commercially viable tool. The developer put in a lot of work on this package, and, unlike many enthusiast-created systems, it works and doesn't have any bugs. Many heritage pen-testing tools and vulnerability scanning tools of a similar age to Arachni no longer work or have been made obsolete by advancements in technology. Arachni does not fit into that category – it still works, and its results are insightful.