RASpray: Understanding and Using a Password Spraying Tool

Comprehensive Exercise Fall 2024, Carleton College
Case, Roo & Touré, Elhadji Amadou | Advisor: Jeff Ondich

Introduction

RASpray is a password spraying tool developed to demonstrate the challenges and vulnerabilities inherent in authentication systems. This document will guide you through the underlying concepts, and how the tool works. Whether you're a cybersecurity enthusiast or simply curious about web security, this guide will provide a solid foundation for understanding RASpray and its role in ethical hacking.

What is Password Spraying?

Password spraying is a targeted attack technique where a single password (or a set of passwords) is tested against many usernames over time. Unlike brute attacks, which test many passwords for one user, spraying avoids triggering account lockouts by spreading attempts across multiple accounts. Simply put, it's trying many usernames and passwords in every combination, hoping that one will be the magic combination to get you into a system.

Why does it matter?

- Common passwords (e.g., 123456, password, or welcome) are still widely used, leaving accounts vulnerable to this technique, often used by individuals with malicious goals. Hence, companies and organizations might deem it necessary to hire security professionals and penetration testers[1] to emulate password spraying on their own services to ensure the technique fails.
- Lots of people use the same passwords for multiple services/websites/etc. When a password is found, a malicious attacker might try the same username/password combinations on other websites (a cracked social media password might be tried on several financial institution logins, for example).

[1] A **penetration tester** (often referred to as a "pen tester") is a cybersecurity professional who simulates cyberattacks on a company's computer systems, networks, or applications. The goal of a penetration test is to identify and exploit vulnerabilities in security defenses before malicious hackers can do so. Pen testers use a variety of tools, techniques, and ethical hacking methods to assess the security posture of an organization and recommend improvements.

- Organizations often enforce lockouts after multiple failed attempts for a single account but fail to detect multiple failed attempts across multiple accounts. This requires the spraying tool to have timing and coordination so as not to trip any alarms.

What We Did

Server

To test a password spraying tool properly, we needed a server to attack. We chose to create a server in the cloud (specifically Amazon Web Services' Lightsail platform) instead of locally for several reasons, namely:

- Most password spraying attacks occur from a non-local entity. We wanted to work in a slightly more realistic environment and not have the server locally hosted (where latency, packet loss, etc, are more negligible).
- AWS is a standard service many companies use. We wanted to gain some more experience with it.
- AWS has built-in tools for load balancing and elegantly automatically recovering from system crashes.
- They allow white-hat penetration testing.
- AWS has an easy-to-use automatic snapshot system. If anything we did would have resulted in system corruption and required a system rebuild. (This turned out to be very useful) We were less confident that such a crash would be quick and easy to recover from if it occurred on our hardware.

Within the AWS container, we have a system running the latest version of Debian. For the server functionality, we're running both nginx and Apache. Both tools are free to use and open source.

nginx

Nginx (pronounced engine-ex, stylized as **nginx** or **NGINX**) acts as our buffer and load balancing tool, which is one of the things the software is commonly used for. It reserves a few megabytes to ask as a queue. Apache can only take so many requests at once, and too many requests result in Apache either rejecting connection attempts or crashing. Nginx takes all requests through port 80 (the standard port for HTTP connections) and puts them into its queue. If there are more requests in the queue than Apache can handle, it waits and only passes a certain number of requests per second. Otherwise, it empties its queue and passes everything to Apache.

Apache2

Apache is our primary server tool. It handles all the internal logic, serves up webpages, and acts as the authentication server for basicauth. In addition, it keeps a list of all the files that can or cannot be served to a requester, blocking requests for configuration, password hashes, or other sensitive files.

Login Techniques

There are numerous ways that people can log into a website. Before going into detail about how our tool works and if it was successful, here are the four types of logins we worked on, successfully implementing three.

BasicAuth

BasicAuth is one of the primary ways that basic servers allow specific web pages or folders to be accessible only after a username or password has been given. BasicAuth is very easy to implement. Apache natively uses a form of password encryption called MD5. A tool that comes with Apache's utilities package is **.htpasswd**, which allows you to automatically create salted password hashes (a salted password hash is one where an extra few random characters have been added to the password to reduce the ability for an attacker to pre-compute hashed passwords and simply try to find matches). There are two files within Apache's configuration settings where you can specify server-wide password requirements or file/folder-specific permissions.

1. **Initial Request:** The client sends a request to the server for a protected resource without authentication headers.

```
Unset
GET /protected-resource HTTP/1.1
Host: example.com
```

2. **Server Response:** The server responds with a **401 Unauthorized** status code and includes a **WWW-Authenticate** header, which shows that Basic Authentication is required.

```
Unset
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Basic real="Access to the protected site"
Host: example.com
```

3. **Client Resubmission:** The client resubmits the request, this time including the Authorization header with credentials encoded in Base64[2]

```
Unset
GET /protected-resource HTTP/1.1
Host: example.com
Authorization: Basic <...>
```

4. **Server Verification:** The server proceeds to decode and verify the credentials. If valid, it responds with a **200 OK** status code and provides the requested resource.

```
Unset
HTTP/1.1 200 OK
```

5. **Failed Authentication:** If the credentials are invalid, the server may respond with a **403 Forbidden** or another **401** Unauthorized status code.

PHP/HTML Form

Apache takes a hands-off approach, assigning session tokens to each user. These sessions are then granted access to specific sections of the server's file structure if they pass an HTML login form. PHP handles the hashing and comparison, using a library designed to correctly implement the MD5 hashing algorithm (PHP doesn't have native support). If the inputted username/password combo matches a known username, the session is given a variable that is marked as "true."

Each target page has a PHP element that checks if the session has the assigned variable set to "true." If not, the user is automatically redirected to the login page; otherwise, the page is served as normal.

Login Logging

This is very similar to the HTML/PHP form. The server keeps logs of all login attempts. The same login interaction occurs, but if the login attempt fails, it's added to another log file

[2] Base64 is a binary-to-text encoding scheme that represents binary data in an ASCII string format by translating it into a radix-64 representation. Commonly used for encoding data in applications like email and data transmission in JSON or XML, Base64 makes sure that binary data can be safely transferred or stored without being corrupted by systems that may not support non-text binary formats. The encoding uses a set of 64 characters, including uppercase and lowercase letters, numbers, plus (`+`), and slash (`/`). Padding with `=` is used to make sure the output length is always a multiple of four.

containing all attempts logged in the past 10 minutes. Every time an account attempts to log in before the password is checked, it first checks to ensure there have not been ten or more incorrect password attempts in the past 10 minutes. If there are, then the user is blocked from attempting to login. This protects against more targeted spraying: a tool looking to get into a small number of accounts will fail as it blocks more than ten attempts.

Google reCAPTCHA

This still needs to be fixed. We attempted to add Google's reCAPTCHA API to detect if the user trying to log in is human. However, the documentation, especially Enterprise APIs (such as the one Carleton provides) for adding it to more homebrew versions of PHP websites, is lacking, and we didn't have enough time to troubleshoot and test properly. However, once implemented, it would stop spraying as it would detect that the spraying tool is not "human" enough - typing too fast, submitting too many requests in a short period, and so on.

Tool

RASpray is a Python-based tool designed to perform password-spraying attacks. It is designed explicitly for ethical penetration testing and educational purposes. It's named after its authors, **R**oo Case and **A**madou Touré, and also serves as a nod to the Egyptian God **Ra**, whose existence and lighting up the world can be linked to RASpray's ability to shine a light on vulnerable usernames and passwords, and to peer into the darkness past the defenses of servers. That said, the tool interacts with a list of usernames and passwords provided by the user to attempt login on web pages that request credentials, such as Basic Authentication or login forms with password fields.

Overall, the tool automates the password spraying process by:

- Traversing the target website / IP address to identify pages that require authentication.
- Differentiating between pages that use HTTP Basic Authentication and those that have password fields in forms.
- Attempting to authenticate using provided lists of usernames and passwords.
- Optionally performing brute-force attacks if initial attempts fail.
- Logging resource usage and providing a summary of the attempts.

RASpray displays some key features in terms of its workflow:

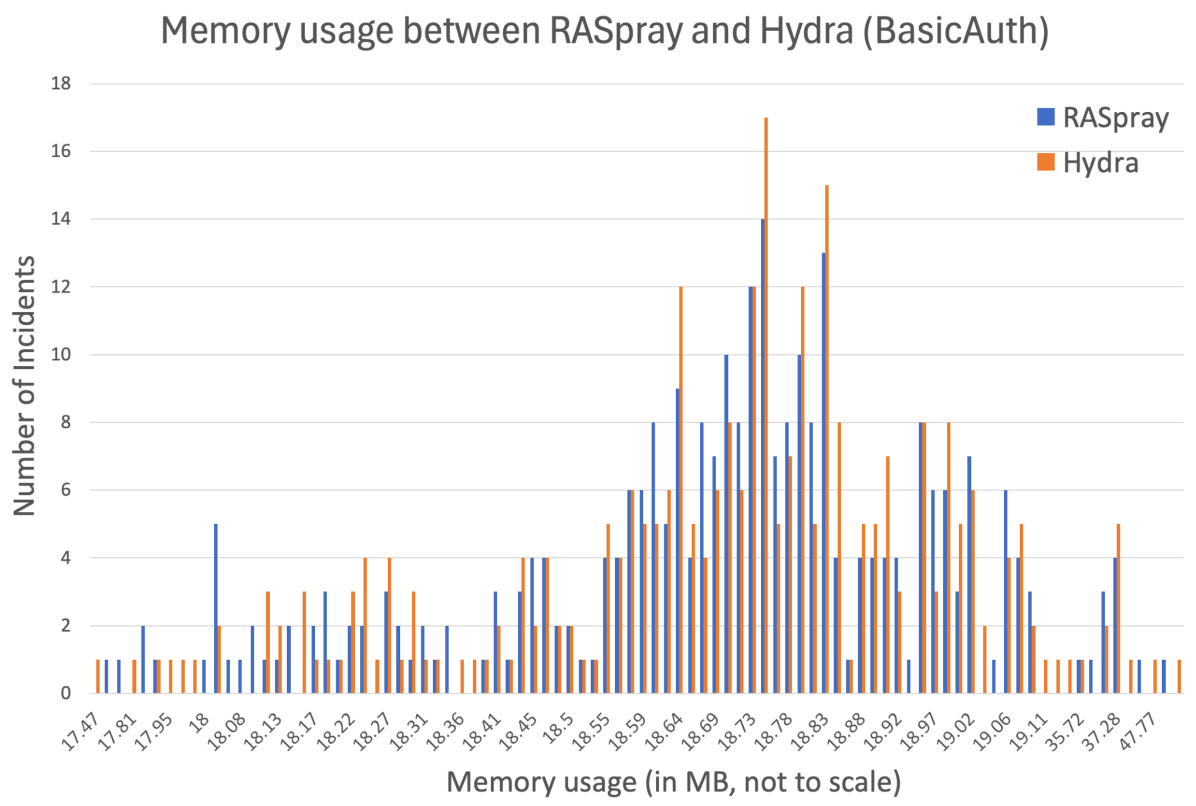
1. **Website Traversal:** The tool can traverse a website to discover login forms and authentication systems. It starts from a given base URL and follows internal links up to a specified depth (by default, 3). For each page, RASpray checks whether it contains a password field or a Basic Authentication request through the function

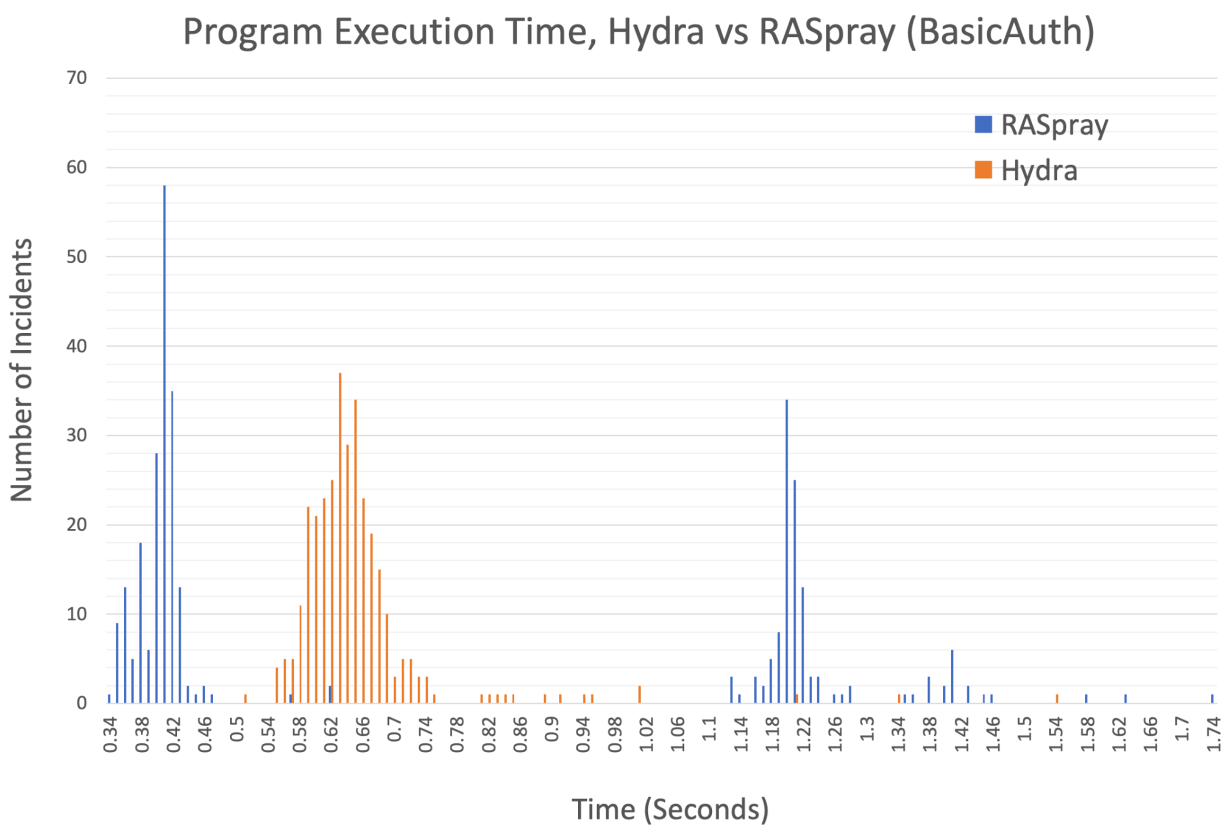
traverse_website(base_url, session, maxdepth=3). For each discovered URL, it checks whether it requires authentication and identifies the type:

   a. HTTP Basic Authentication:
      i. Sends a GET request without credentials.
      ii. Checks if the response status code is **401 Unauthorized**.
      iii. Uses **is_password_requested_basic_auth(url, session)**.
   b. Password Fields in Forms:
      i. Loads the page content.
      ii. Parses the HTML to find **<input type=”password”>** fields.
      iii. Checks for forms that resemble login forms (“username”).
      iv. Uses **is_password_requested_basic_password_field(url, session)**

2. **Password Spraying:** For each page requiring authentication, RASpray proceeds to attempt to authenticate using the provided usernames and passwords.
   a. **Filtering:** Users can specify criteria for usernames and passwords, such as minimum length, presence of uppercase letters, numbers, etc. This filtering is done through the function **filter_by_criteria(items, criteria).**
   b. **Authentication Attempts:**
      i. HTTP Basic Authentication uses the **requests** library with **HTTPBasicAuth,** using the function **basic_auth_spray(full_url, password, username, session)**.
      ii. The tool submits the login form with the provided credentials for forms with password fields and parses the response to determine if the login was successful. It particularly uses the function **attempt_login_with_form(full_url, session, username, password)**
   c. **Concurrency:**
      i. To fasten the process and perform multiple authentication attempts simultaneously, RASpray finally utilizes threading through the **threading** library.
3. **Brute Force Option:** If password spraying doesn’t work, RASpray also offers a fallback to perform a full brute-force attack:
   a. It generates all possible combinations of characters within specified length constraints.
   b. It also tries each combination as a password against the list of usernames.
   c. All done through the function **perform_bruteforce(username_file, password_file, ip_address, criteria, session, pages_to_check).**

Results


Memory usage between RASpray and Hydra (BasicAuth)
RASpray
Hydra
Number of Incidents
Memory usage (in MB, not to scale)
0
2
4
6
8
10
12
14
16
18
17.47
17.81
17.95
18
18.08
18.13
18.17
18.22
18.27
18.31
18.36
18.41
18.45
18.5
18.55
18.59
18.64
18.69
18.73
18.78
18.83
18.88
18.92
18.97
19.02
19.06
19.11
35.72
37.28
47.77


Program Execution Time, Hydra vs RASpray (BasicAuth)
RASpray
Hydra
Number of Incidents
Time (Seconds)
0
10
20
30
40
50
60
70
0.34
0.38
0.42
0.46
0.5
0.54
0.58
0.62
0.66
0.7
0.74
0.78
0.82
0.86
0.9
0.94
0.98
1.02
1.06
1.1
1.14
1.18
1.22
1.26
1.3
1.34
1.38
1.42
1.46
1.5
1.54
1.58
1.62
1.66
1.7
1.74

Analysis

We primarily tested on BasicAuth. This is because Hydra is not easy to set on any other login type. BasicAuth performs fairly consistently on the server side, whereas PHP can sometimes be inconsistent. These two tools are also doing slightly different things, which may skew the data. Hydra takes in the direct URL to the page that BasicAuth protects, while RASpray traverses the website, finding and spraying the BasicAuth-protected page.

We can see in the diagrams that our tool is consistent in memory usage between Hydra and RASpray. However, the data looks weirder when we look at the overall runtime. There are two peaks in runtime with our tool and only one for Hydra. This is because our tool is running too fast. Hydra can better pace itself and fine-tune its delays when it detects that it is sending enough requests to overwhelm the target server. At the same time, RASpray cannot be as fine-grained, waiting a base amount of 1 second before resuming its spray.

What's Next?

We'd like to keep working on this project. Some areas we're interested in improving this tool:

- **Moving to a formally open source model:** We're interested in making this project more accessible for others to contribute to.
- **Making a formal Request For Comments (RFC):** If we are going to continue on this project, we want a better outline of our goals, what we are including in scope, and what is not included.
- **Implementation of ML/AI:** Sometimes a login page will give more detailed information, such as if a provided login username/email even exists, more detailed information on password requirements. Machine learning models to detect this and modify the attack could prove helpful.
- **Better rate limiting on attacks:** Our tool is fairly clumsy in terms of our attacks: Sometimes it performs faster than Hydra, other times not. We think this is partially due to a set limit in case of an error response. We'd like to investigate more and make our rate limiting more eloquent.

Bibliography

NOTE: This bibliography was created between September and November 2024. Many websites are listed as being accessed on November 23, 2024. They are listed as such, as this was when they were imported into Zotero.

1. Brower M. How to Create and Use .htpasswd | Hostwinds. Hostwinds Tutorials. 2017 Jun 22 [accessed 2024 Nov 23]. https://www.hostwinds.com/tutorials/create-use-htpasswd

2. Jeremy. whitehat101/apr1-md5. 2024 [accessed 2024 Nov 23]. https://github.com/whitehat101/apr1-md5

3. Jevtic G. mod_evasive on Apache: Install & Configure to Defend DDoS Attacks. Knowledge Base by phoenixNAP. 2019 Mar 5 [accessed 2024 Nov 23]. https://phoenixnap.com/kb/apache-mod-evasive

4. Pollard B. Answer to “How to Disable ModSecurity without any Errors.” Server Fault. 2015 [accessed 2024 Nov 23]. https://serverfault.com/a/676089

5. user9517. Answer to “How to Disable ModSecurity without any Errors.” Server Fault. 2015 [accessed 2024 Nov 23]. https://serverfault.com/a/676095

6. User829193. How to Disable ModSecurity without any Errors. Server Fault. 2015 [accessed 2024 Nov 23]. https://serverfault.com/q/675135

7. user1016850. Answer to “How to Disable ModSecurity without any Errors.” Server Fault. 2023 [accessed 2024 Nov 23]. https://serverfault.com/a/1129112

8. Web L. NGINX vs Apache: Picking Best Web Server for Your Business. Liquid Web. 2024 [accessed 2024 Nov 23]. https://www.liquidweb.com/blog/nginx-vs-apache/

9. Authentication and Authorization - Apache HTTP Server Version 2.4. [accessed 2024 Nov 23]. https://httpd.apache.org/docs/2.4/howto/auth.html

10. How to block IP addresses at the Apache HTTP Server level? | Adobe Experience Manager. [accessed 2024 Nov 23]. https://experienceleague.adobe.com/en/docs/experience-cloud-kcs/kbarticles/ka-17455

11. How To Set Up Password Authentication with Apache on Ubuntu 14.04 | DigitalOcean. [accessed 2024 Nov 23]. https://www.digitalocean.com/community/tutorials/how-to-set-up-password-authentication-with-apache-on-ubuntu-14-04

12. htpasswd - Manage user files for basic authentication - Apache HTTP Server Version 2.4. [accessed 2024 Nov 23]. https://httpd.apache.org/docs/2.4/programs/htpasswd.html

13. htpasswd - Manage user files for basic authentication - Apache HTTP Server Version 2.4. [accessed 2024 Nov 23]. https://httpd.apache.org/docs/2.4/programs/htpasswd.html

14. PHP: apache_setenv - Manual. [accessed 2024 Nov 23]. https://www.php.net/manual/en/function.apache-setenv.php

15. PHP: HTTP authentication with PHP - Manual. [accessed 2024 Nov 23]. https://www.php.net/manual/en/features.http-auth.php

16. Top 200 Most Common Passwords. NordPass. [accessed 2024 Nov 23]. https://nordpass.com/most-common-passwords-list/

17. NGINX rate-limiting in a nutshell. freeCodeCamp.org. 2017 Mar 24 [accessed 2024 Nov 9]. https://www.freecodecamp.org/news/nginx-rate-limiting-in-a-nutshell-128fe9e0126c/

18. How to Configure ModEvasive with Apache on Ubuntu Linux | Rapid7 Blog. Rapid7. 2017 Apr 9 [accessed 2024 Nov 23]. https://www.rapid7.com/blog/post/2017/04/09/how-to-configure-modevasive-with-apache-on-ubuntu-linux/