

Sprinkler: A Multi-Service Password Sprayer

Prompt Eua-anant, Sam Ederington

Introduction

Would you like to find the password for your mom's X account? A password sprayer is your friend.

In a nutshell, a password sprayer...

- Sends a bunch of usernames and passwords to a remote server **rapidly**
- Then tells you which ones grant you access

Why should you care?

- Hackers use password sprayers to **gain unauthorized access** to accounts with weak passwords
- If you own a server that allows users to authenticate themselves, you want to know how a password sprayer works, and how to protect against password-spraying attacks.
- A password sprayer can help you learn about password-spraying tactics and see how easy it is to gain unauthorized access to someone's account on your server.

Our goal

- We made Sprinkler, a password sprayer written in C, that supports the following services: ssh, http-get (basic auth), and http-post
- We set up a target server compatible with all those services above.

Introduction - Services

Before going in-depth, here is a brief introduction on each of the services Sprinkler offers, and how they differ from each other.

SSH (Secure shell)

A protocol that allows you to gain access to the OS of a remote server (i.e. another device far away). Once you've successfully entered the login credentials, you get a shell where you can type in OS commands, and they'll be executed on the remote server! Moreover, the

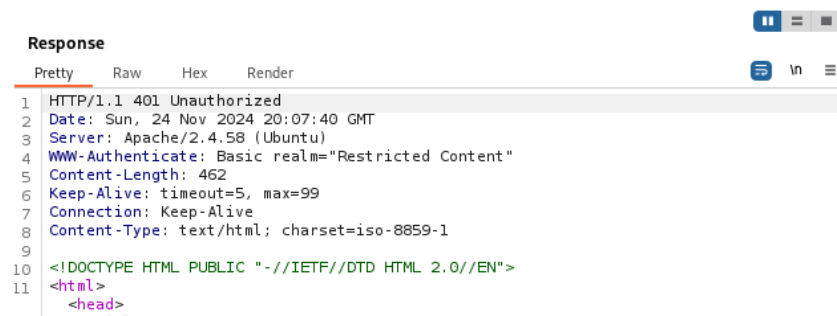
communication between you and the remote server is secure, meaning that it's very unlikely to be intercepted and read by third parties

HTTP-GET

In the password spraying context, this refers to HTTP/HTTPS server that accepts login credentials via a GET request. However, while there are different authentication schemes by which credentials are submitted through a GET request (eg. Basic, Digest, Negotiate)¹, Sprinkler only supports one authentication scheme: Basic, also known as basic auth.

In normal contexts (not password spraying), here's how it works:

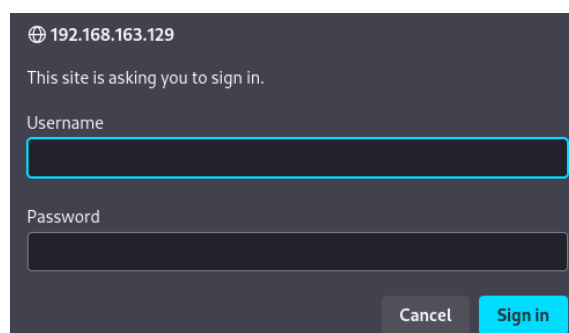
1. User's server send a GET request to the target server
2. The target server sends the user a 401 Unauthorized response code, and one of the response headers is www-authentication, with a value of 'Basic realm = ...'
 - a. 'Basic' indicates that the server uses basic auth
 - b. The 'realm' variable is optional and indicates the name of the group of resources protected by the same login credentials



```
Response
Pretty Raw Hex Render
1 HTTP/1.1 401 Unauthorized
2 Date: Sun, 24 Nov 2024 20:07:40 GMT
3 Server: Apache/2.4.58 (Ubuntu)
4 WWW-Authenticate: Basic realm="Restricted Content"
5 Content-Length: 462
6 Keep-Alive: timeout=5, max=99
7 Connection: Keep-Alive
8 Content-Type: text/html; charset=iso-8859-1
9
10 <!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
11 <html>
  <head>
```

Server's response

3. The browser shows a pop-up where the user can input **username** and **password**



192.168.163.129

This site is asking you to sign in.

Username

Password

Cancel Sign in

¹ <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/WWW-Authenticate#directives>

4. Once the user typed in and hit enter, the browser concatenates **username** and **password** into the string “**username:password.**”
5. The browser encodes the string into base64 format, which ensures that all characters are printable ASCII characters.
6. The browser prepares a new GET request, and this time it adds the encoded text to the request header “Authentication”

[IMAGE]


7. Send this new GET request to server.
8. The server’s response to the user depends on whether the credentials are correct or not.

For password sprayers, the process starts at step 4, but instead of typing the credentials to the browser, the user inputs password and username files to a password sprayer, which will encode and format it properly in a GET request and send it to the server.

HTTP-POST

This refers to HTTP/HTTPS that uses form-based authentication. Here’s how it works:

1. The user sends a GET request to the server
2. The server sends a login form, containing the variable names it expects the user to use.



```
Response
Pretty Raw Hex Render
1 HTTP/1.1 200 OK
2 Date: Sun, 24 Nov 2024 20:07:44 GMT
3 Server: Apache/2.4.58 (Ubuntu)
4 Vary: Accept-Encoding
5 Content-Length: 184
6 Keep-Alive: timeout=5, max=98
7 Connection: Keep-Alive
8 Content-Type: text/html; charset=UTF-8
9
10 <html>
11   <body>
12     <form method="POST">
13       <input type="text" name="name">
14       <input type="password" name="psw">
15       <input type="submit">
16     </form>
17   </body>
18 </html>
```

In the image above, the login form has the variables “name” and “psw” that correspond to the username and password, and a submit button.

3. The user types in the username and password in the login form and click submit.
4. The user’s browser prepares a POST request and appends the username and password in the body of the request:

[IMAGE]

Unlike basic auth, the credentials are below the headers, and separated by an ‘&’ (NOTE, this syntax where variable values are separated by ‘&’, and non-alphanumeric characters are percent-encoded, is called `application/x-www-form-urlencoded` and is not the only encoding type out there.)²

5. User’s browser sends the POST request to the server.
6. The server processes and stores the variables sent from the user. Unlike GET requests, a POST request means that the variables will be stored on the server, and that sending the same POST request again and again does not guarantee the same effects (the standard term is *non-idempotent*).
7. The server’s response depends on whether the credentials are correct.

Here’s a brief summary of all services

Service	Description
ssh (secure sh ell)	For interacting/executing OS commands on the remote server, from your local machine.
http-get	In the password spraying context, this refers to HTTP/HTTPS server that accepts login credentials via a GET request. However, while there are different authentication schemes by which credentials are submitted through a GET request, Sprinkler only supports one authentication scheme: basic auth.

² <https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/POST>

http-post	This concerns HTTP/HTTPS servers that send the user a login form, and expect the user to send a POST request, containing login credentials as form variables, which will be stored on the server.
-----------	---

What is Sprinkler?

So much for how each service works. Here is

- A password sprayer written in C that supports spraying across the following services: ssh, http-get, and http-post.
- Extra global options include
 -

How Sprinkler works?

TL;DR

1. User inputs options/arguments through command line
2. Sprinkler processes the command line and checks for invalidities
3. According to the service specified, Sprinkler...
 - a. Prepares login request so it has the right format
 - b. Sends login request to server
 - c. Receives server's response
 - d. Determines whether it's a login success or failure
4. If a login success is detected, stop spraying that username and move on to the next.

Here's a more detailed explanation of each step!

- 1. User input options/arguments through command line**

The user input the following mandatory options/argument through the command line

- a. Username or username file (**-u username** or **-U FILE**)
- b. Password or password file (**-p password** or **-P FILE**)
- c. Target server's port (**-s PORT**)
- d. Target server's IP address or domain name
- e. Service

The user can also input extra options such as

- Set a delay time between logins (**-d DELAY**)
- Prints out status report and/or all login attempts (**-v** or **-V**)
- Connect via TLS (**-S**)
- Input a regex that determines a login success/failure based on whether the regex is found in target's response (**-r REGEX**)
- Specify form parameter names and values (if applicable) for servers that use form-based authentication (**-i PARAMETERS**)

2. Sprinkler processes the command line

In sprinkler.c file, Sprinkler parses the command line using C's `<unistd.h>` library function **getopt**. Essentially, a single call to **getopt** returns the nearest option character in the command line, starting with the first one. **getopt** also sets the variable *optarg* if there is an argument next to the option. Accordingly, Sprinkler has a while loop where each iteration gets the option, checks what option it is, whether an argument is supplied, and initialises appropriate variables.

While processing each option/argument, Sprinkler also checks its validity by using **getopt**'s feature: if the user types in an argument to an option that does not expect one, or there's no argument for an option that requires an argument, then **getopt** returns a recognisable marker. If multiple invalidities exist in the command line, Sprinkler detects as many as it can, prints the error messages for each invalidity, and exits.

The following invalidities are detected during this process:

- The five mandatory inputs (see above) are not provided
- Mismatch between actual vs. expected number of arguments
- Port is not a number or ≤ 0
- Username or password files do not exist
- Delay time is not a number or ≤ 0
- Input regex does not start with S= or F=

- Unknown option provided
- Service unsupported

3. Sprinkler prepares and send login request for the service specified, and processes the target server's response

Once the service is identified, Sprinkler either calls the function from sprinkler-ssh.c or sprinkler-http.c, depending on the service.

This is the step where each service is wildly different: each expects a different login request format and returns different things, so Sprinkler needs to take that into account. Here is a breakdown of what's going on for each service.

SSH

Sprinkler uses Libssh library functions to perform the following tasks:

- a) Create a **ssh_session** struct by calling `ssh_new()`. This struct is for storing necessary information used for connecting to an ssh server.
- b) Call `ssh_options_set()` to add the necessary information to the **ssh_session** object, including target's IP, target's port, and what username to log in as.
- c) Connect to the server by calling `ssh_connect`, which accepts the **ssh_session** from the previous step.
- d) Sends the password to the target. This is done by calling the function `user_auth_password`
- e) The return value indicates whether it's a login success or failure.

HTTP-GET

First, in order to send or receive messages from the target server, Sprinkler sets up a network socket.

What is that?

In UNIX systems, in order to send messages to a remote server, there must be a file to write messages to. Once the message is written to the file, the OS will send the file contents to the network card, which passes it on to the router, then from the router to the modem, and so on until it reaches the remote server. But given that there are lots of files on the system, how does

the OS know which file is meant for communicating with the remote server? Answer: the OS assigns each file a file descriptor, which is just a number that's associated with the file location. When Sprinkler sends a message to the server, it also passes in a file descriptor to the function, so that the OS knows which file to write the messages to.

A socket, then, is just an endpoint (or for UNIX systems, a file) where messages intended for the remote server are written to.

How does Sprinkler set up a socket and connect to server?

First, sprinkler use C's <netdb> library function *getaddrinfo* to get the necessary information for setting up a socket. This includes what internet protocol the target server accepts, whether stream or datagram protocol is used, what specific protocol it uses, and the IP address. Sprinkler requires that the target server use IPv4 and one of the streaming protocols (TCP is a popular one).

Next, Sprinkler uses C's <sys/socket> library functions *socket* and *connect* to set up a stream socket and connect to the target server respectively.

Formatting login request

Sprinkler assumes that the server uses basic auth authentication scheme and prepares the request buffer:

- a) Set up all the required headers in the GET request
- b) Concatenates the username and password in to "username:password"
- c) Encode the text into base64.
- d) Add the encoded text to the "Authentication" header. So far, the request looks like this:
- e) Use C's <sys/socket> library function to send the request to server.
- f) Receive the server's response (see the section sprinkler_recv for details).
- g) Process the server's response. By default, if the server sends a 200 OK, then Sprinkler deems that the login was successful.

HTTP-POST

Like HTTP-GET, Sprinkler also sets up a socket and connects to the server. But the next steps are wildly different.

By default:

- a) Sprinkler sends a GET request to the server:
- b) The server sends a login form
- c) Sprinkler parses the login form.
 - i) Get the name and value fields for each <input> tag
 - ii) For the name field, if the following regex is found in the field...
"usr|user|email|.name"
...then the field is a parameter name that corresponds to the username.
 - iii) In contrast, if the following regex is found in the name field...
"pass|pw|psw|ps"
Then the field is a parameter name that corresponds to the password.

(NOTE: for steps i-iii, the regex uses POSIX extended regex and ignores case)
 - iv) If there is a value field present, this is neither the username nor password, but Sprinkler appends the name and value to the body of the POST request
- d) The actual username and password are added in the body of the POST request next to their form parameter names.
- e) Sprinkler calculates the Content-Length, which is how many bytes does the body of the request contain, and add that to the Content-Length header.

At this step, this is what the POST request looks like

- f) Sprinkler sends the request to the server
- g) Receives the server's response (see the [sprinkler_recv](#) section for details)
- h) Check if the response indicates a redirect to a different page, then Sprinkler deems a login success. A redirect is indicated by the Location header value that's different from the login page. For instance:

```
Response
Pretty Raw Hex Render
1 HTTP/1.1 302 Found
2 Date: Sun, 24 Nov 2024 20:11:02 GMT
3 Server: Apache/2.4.58 (Ubuntu)
4 Location: https://zapatopi.net/treeoctopus/
5 Content-Length: 0
6 Keep-Alive: timeout=5, max=100
7 Connection: Keep-Alive
8 Content-Type: text/html; charset=UTF-8
```

sprinkler_recv - how it works

The `sprinkler_recv` function is used for receiving the target server's response for `http-get` and `http-post`. It utilizes C's `<sys/socket>` library function `recv`. Since the server can send the response in multiple packets, one call to `recv` does **not** guarantee that it receives all of the server's response. Consequently, `sprinkler_recv` proceeds by the following steps

1. Call `recv` once to determine whether there is a Content-Length header:
 - a.

NAME

`sprinkler` - a very fast password sprayer that works for `ssh`, `http-get` (basic auth), and `http-post`

SYNOPSIS

```
sprinkler [-u USERNAMES | -U FILE] [-p PASSWORD | -P FILE] [-s PORT] SERVICE TARGET
sprinkler [-h SERVICE | -h]
```

DESCRIPTION

`sprinkler` is a fast password sprayer that supports the following services:
`ssh`, `http-get`, `http-post`

This tool allows researchers and pen-testers to see how easy it would be to gain unauthorized access to a remote server.

MANDATORY OPTIONS

TARGET The server (and directory, if applicable) to attack, can be an IPv4 address or domain name

SERVICE The target's login service. Type 'sprinkler -h services' to list the protocols available

-s PORT

the target server's port

-u USERNAME / -U FILE

supply a login username, or -U FILE to supply a username list, one line per username

-p PASSWORD / -P FILE

supply a password, or -P FILE to supply a password list, one line per password

EXTRA GLOBAL OPTIONS

-d DELAY

set a delay time in seconds between each login attempt

-v / -V

Verbose mode. -V prints out all login attempts.