SOFTWARE SECURITY

ASSIGNMENT 6

ASSIGNMENT 6

SANTAS-COOKIE

- Clearly, we need to turn a cookie in the cookie jar into Santa’s special cookie
- Construct an object with the correct *vtable* pointer to win

SANTA

- `checksec`: everything enabled.
- TCP server that spawns a new thread for each connection.
- `ynetd` is only there for isolation.

SANTA

- You can do “nice” (eat your veggies) and “not nice” (steal candy) things.
- Getting presents (the flag) requires you to have been nice, but isn’t nice itself, so getting the flag should be impossible.

SANTA

- You can do “nice” (eat your veggies) and “not nice” (steal candy) things.
- Getting presents (the flag) requires you to have been nice, but isn’t nice itself, so getting the flag should be impossible.
- Approach 1: Fill up the history with “nice” actions, then try to reset it during the check.
    - Timing this correctly is tricky-to-impossible, but this works in theory.

SANTA

- You can do “nice” (eat your veggies) and “not nice” (steal candy) things.
- Getting presents (the flag) requires you to have been nice, but isn’t nice itself, so getting the flag should be impossible.
- Approach 1: Fill up the history with “nice” actions, then try to reset it during the check.
  - Timing this correctly is tricky-to-impossible, but this works in theory.
- Approach 2: Continually reset the history with new connections that try to get the flag immediately, and get lucky eventually.
  - With 16 threads, this works near-instantaneously!

CALC

- `checksec`: everything enabled (except `_FORTIFY_SOURCE`)
- Trivial pointer leaks with `dump( ... )`

CALC

- `checksec`: everything enabled (except `_FORTIFY_SOURCE`)
- Trivial pointer leaks with `dump( ... )`
- `Comment` gives us arbitrary-size allocations

CALC

- `checksec`: everything enabled (except `_FORTIFY_SOURCE`)
- Trivial pointer leaks with `dump( ... )`
- `Comment` gives us arbitrary-size allocations
- Renaming gives us almost-arbitrary heap writes

```cpp
void set_name(std::string_view name) {
    std::memset(this->raw_name, 0, sizeof(this->raw_name));
    std::memcpy(this->raw_name, name.data(), name.length());
}
```

CALC

- `checksec`: everything enabled (except `_FORTIFY_SOURCE`)
- Trivial pointer leaks with `dump( ... )`
- `Comment` gives us arbitrary-size allocations
- Renaming gives us almost-arbitrary heap writes

```
void set_name(std::string_view name) {
    std::memset(this->raw_name, 0, sizeof(this->raw_name));
    std::memcpy(this->raw_name, name.data(), name.length());
}
```

- We can use this to replace the vtable of an object with a custom vtable
- Just set →`dump` to `win()`
- *Bonus question: how would you solve this without* `win()`*?*

HTTPD

- `checksec`: everything enabled.
- HTTP server that spawns a new thread for each connection.

HTTPD

```
strtok(3)                Library Functions Manual                strtok(3)

NAME
       strtok, strtok_r - extract tokens from strings

LIBRARY
       Standard C library (libc, -lc)

SYNOPSIS
       #include <string.h>

       char *strtok(char *restrict str, const char *restrict delim);
       char *strtok_r(char *restrict str, const char *restrict delim,
                      char **restrict saveptr);
```

HTTPD

```
strtok(3)                Library Functions Manual                strtok(3)

BUGS
       Be cautious when using these functions. If you do use them,
       note that:

       *  These functions modify their first argument.

       *  These functions cannot be used on constant strings.

       *  The identity of the delimiting byte is lost.

       *  The strtok() function uses a static buffer while parsing,
          so it's not thread safe.  Use strtok_r() if this matters
          to you.
```

HTTPD

“The `strtok()` function uses a static buffer while parsing, so it’s not thread safe.”

HTTPD

“The `strtok()` function uses a static buffer while parsing, so it’s not thread safe.”

- In practice, `strtok()` stores *a pointer to* the last (replaced) delimiter

HTTPD

“The `strtok()` function uses a static buffer while parsing, so it’s not thread safe.”

- In practice, `strtok()` stores *a pointer to* the last (replaced) delimiter
- This means we have a race condition while parsing two simultaneous requests

HTTPD

Thread 1

Request line `r1` allocated: `GET /r1 HTTP/1.0`

`http_method = strtok(r1)` (“`GET`”)
Delay from `getpeername/getnameinfo`

`request_path = strtok(NULL)` (“`/r2`”)
`http_version = strtok(NULL)` (“`HTTP/1.0`”)
`request_path` is sanity-checked here

Thread blocks on reading HTTP headers
`request_path` is now a dangling pointer

`request_path` is now “`/../../../flag`”
`handle_get` will give us the flag

Thread 2

Request line `r2` allocated: `GET /r2 HTTP/1.0`

`http_method = strtok(r2)` (“`GET`”)
Delay from `getpeername/getnameinfo`

`request_path = strtok(NULL)` (⇒ `NULL`)
Request rejected as invalid (HTTP 400 Bad Request)
`r2` is freed

Thread 3

Request line `r3` allocated where `r2` used to be:
`GET /../../../flag\0 HTTP/1.0`

HTTPD

Thread 1

Request line `r1` allocated: `GET /r1 HTTP/1.0`

`http_method = strtok(r1)` (“`GET`”)
Delay from `getpeername/getnameinfo`

`request_path = strtok(NULL)` (“`/r2`”)
`http_version = strtok(NULL)` (“`HTTP/1.0`”)
`request_path` is sanity-checked here

Thread blocks on reading HTTP headers
`request_path` is now a dangling pointer

`request_path` is now “`/../../../flag`”
`handle_get` will give us the flag

Thread 2

Request line `r2` allocated: `GET /r2 HTTP/1.0`

`http_method = strtok(r2)` (“`GET`”)
Delay from `getpeername/getnameinfo`

`request_path = strtok(NULL)` (⇒ `NULL`)
Request rejected as invalid (HTTP 400 Bad Request)
`r2` is freed

Thread 3

Request line `r3` allocated where `r2` used to be:
`GET /../../../flag\0 HTTP/1.0`

In practice, we send many `r3`s to ensure that one of them ends up in the same location.

HTTPD

Thread 1

Request line `r1` allocated: `GET /r1 HTTP/1.0`

`http_method = strtok(r1)` (“`GET`”)
Delay from `getpeername/getnameinfo`

`request_path = strtok(NULL)` (“`/r2`”)
`http_version = strtok(NULL)` (“`HTTP/1.0`”)
`request_path` is sanity-checked here

Thread blocks on reading HTTP headers
`request_path` is now a dangling pointer

`request_path` is now “`/../../../flag`”
`handle_get` will give us the flag

Thread 2

Request line `r2` allocated: `GET /r2 HTTP/1.0`

`http_method = strtok(r2)` (“`GET`”)
Delay from `getpeername/getnameinfo`

`request_path = strtok(NULL)` (⇒ `NULL`)
Request rejected as invalid (HTTP 400 Bad Request)
`r2` is freed

Thread 3

Request line `r3` allocated where `r2` used to be:
`GET /../../../flag\0 HTTP/1.0`

```
r1/r2   ________________________GET_/existing-file_HTTP/1.0
r3      GET_/existing-file_HTTP/1.0_/../../../flag\0
```