Till REcollapse

Fuzzing the Web for Mysterious Bugs

@0xacb

$ whoami

- André Baptista / 0xacb
- Co-founder @ Ethiack
- Invited professor @ MSc in Infosec - University of Porto
- Bug bounty hunter
- Former captain @ xSTF team

STRANGE & UNBELIEVABLE STORIES OF THE
H1-ELITE
HACKERONE COMICS GROUP 12¢
APPROVED BY THE COMICS CODE AUTHORITY
0XACB
MASTER OF MYSTERY

Agenda

- 1. Input & Regex quirks
- 2. The REcollapse technique
- 3. Mysterious bugs
- 4. Real-world examples

Intro

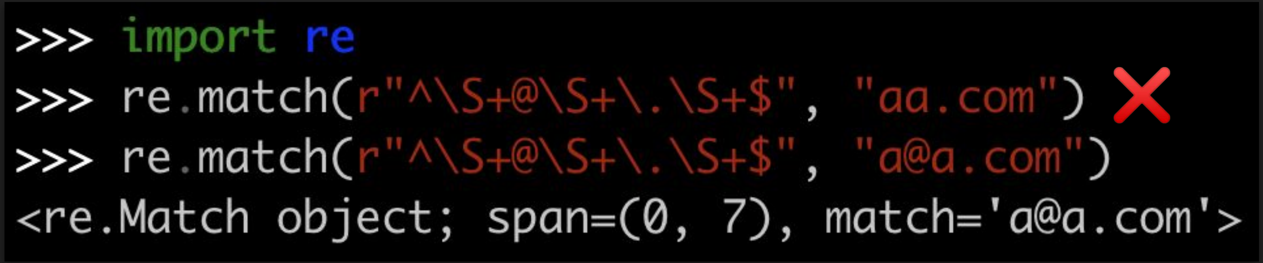
https://example.com/redirect?url=https://legit.example.com ✅

https://example.com/redirect?url=https://evil.com ❌

1. User Input

Dealing with User Input

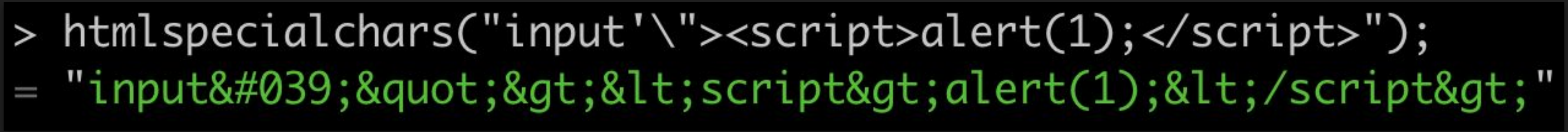
- Modern webapps / APIs rely on:
  - Validation



```
>>> import re
>>> re.match(r"^\S+@\S+\.\S+$", "aa.com") ❌
>>> re.match(r"^\S+@\S+\.\S+$", "a@a.com")
<re.Match object; span=(0, 7), match='a@a.com'>
```

Dealing with User Input

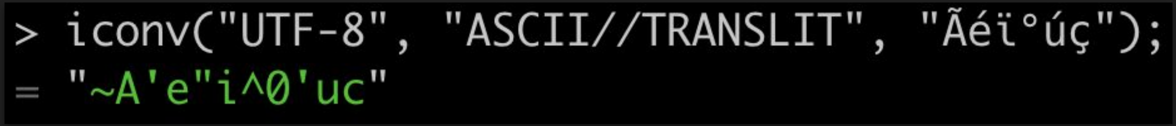
- Modern webapps / APIs rely on:
  - Validation
  - Sanitization



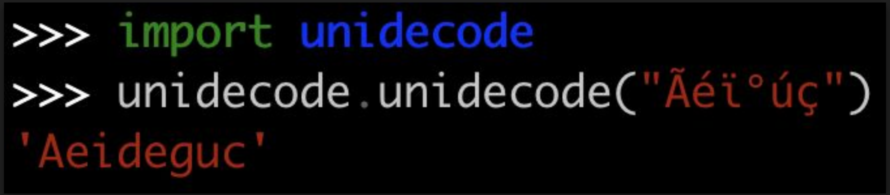
```
> htmlspecialchars("input'\"><script>alert(1);</script>");
= "input&#039;&quot;&gt;&lt;script&gt;alert(1);&lt;/script&gt;"
```

Dealing with User Input

- Modern webapps / APIs rely on:
  - Validation
  - Sanitization
  - Normalization



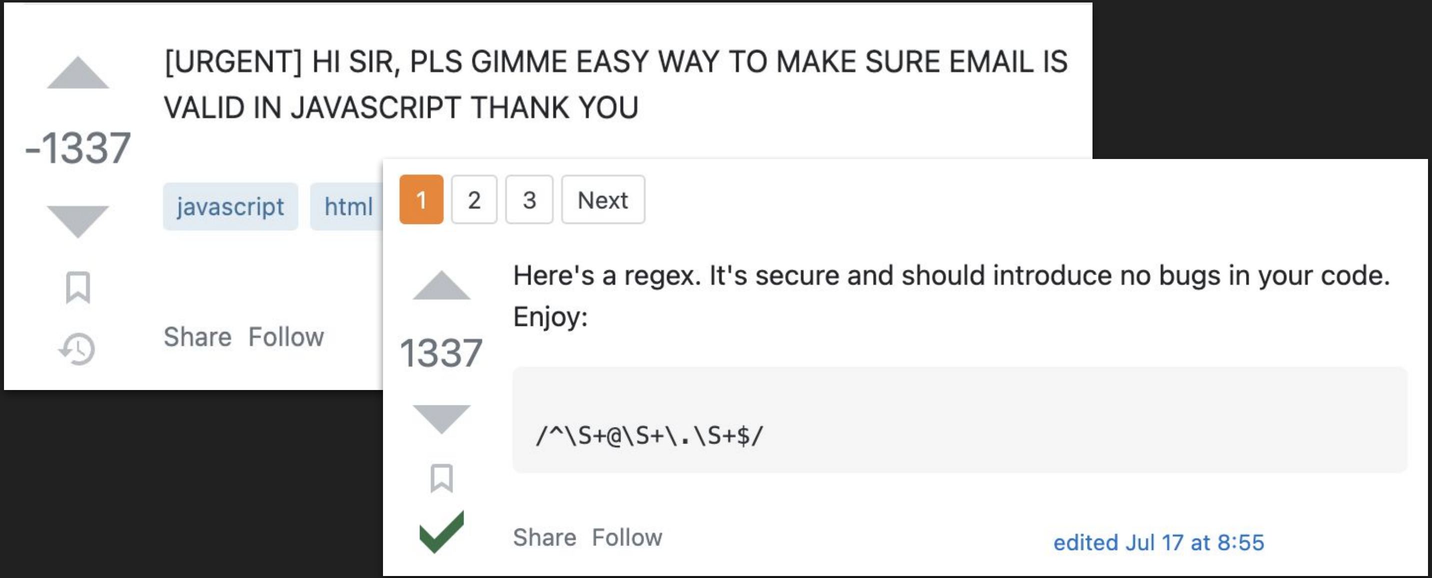
```
> iconv("UTF-8", "ASCII//TRANSLIT", "Ãéï°úç");
= "~A'e"i^0'uc"
```



```
>>> import unidecode
>>> unidecode.unidecode("Ãéï°úç")
'Aeideguc'
```

Problems with Validation

- Regex is widely used to validate parameters from the user
  - Copied from StackOverflow, etc



-1337

[URGENT] HI SIR, PLS GIMME EASY WAY TO MAKE SURE EMAIL IS VALID IN JAVASCRIPT THANK YOU

1337

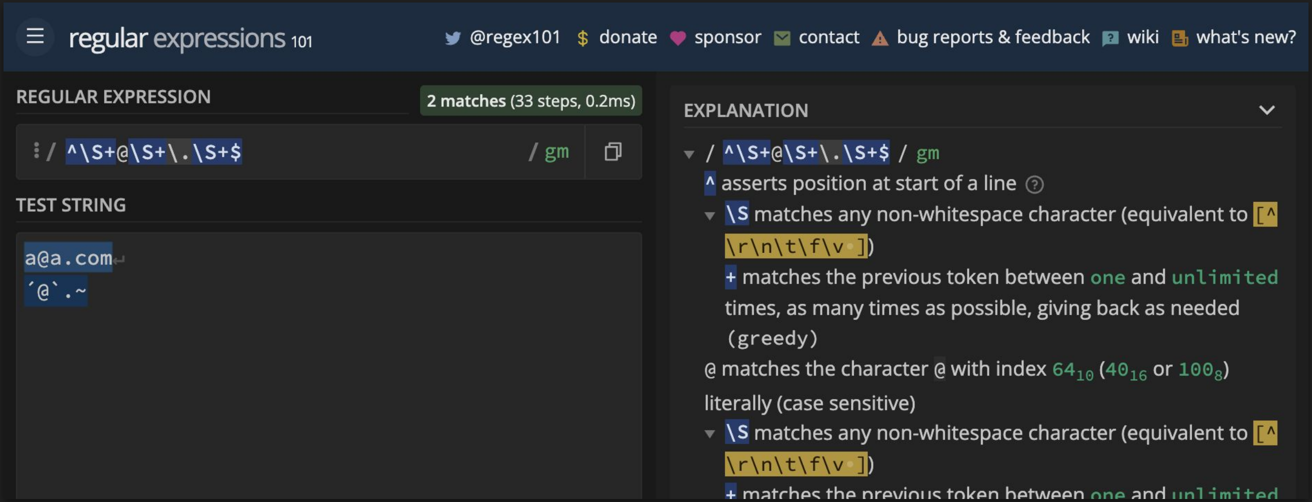
Here's a regex. It's secure and should introduce no bugs in your code. Enjoy:

```
/^\S+@\S+\.\S+$/
```

edited Jul 17 at 8:55

Problems with Validation

- Regex is widely used to validate parameters from the user
  - Copied from StackOverflow, etc
  - Mostly not tested by devs (copy paste)



regular expressions 101 — @regex101 · donate · sponsor · contact · bug reports & feedback · wiki · what's new?

REGULAR EXPRESSION — 2 matches (33 steps, 0.2ms)

```
/ ^\S+@\S+\.\S+$ / gm
```

TEST STRING

```
a@a.com
´@`.~
```

EXPLANATION

/ ^\S+@\S+\.\S+$ / gm

^ asserts position at start of a line

\S matches any non-whitespace character (equivalent to [^\r\n\t\f\v])

+ matches the previous token between one and unlimited times, as many times as possible, giving back as needed (greedy)

@ matches the character @ with index 64_{10} (40_{16} or 100_8) literally (case sensitive)

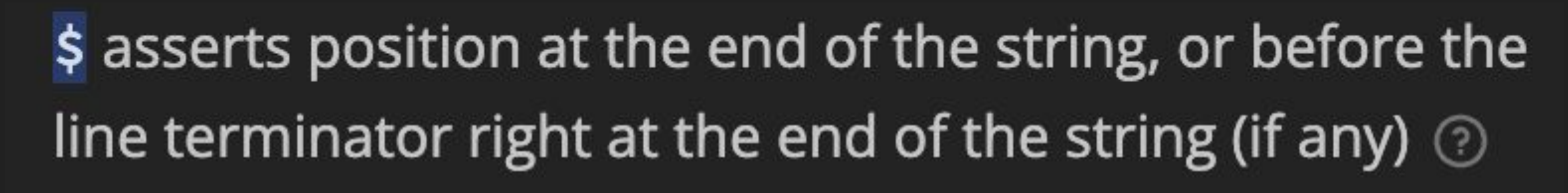
\S matches any non-whitespace character (equivalent to [^\r\n\t\f\v])

+ matches the previous token between one and unlimited

Problems with Validation

- Regex is widely used to validate parameters from the user
  - Copied from StackOverflow, etc
  - Mostly not tested by devs (copy paste)
  - Sometimes testing code exists but it’s specific to a subset of the cases

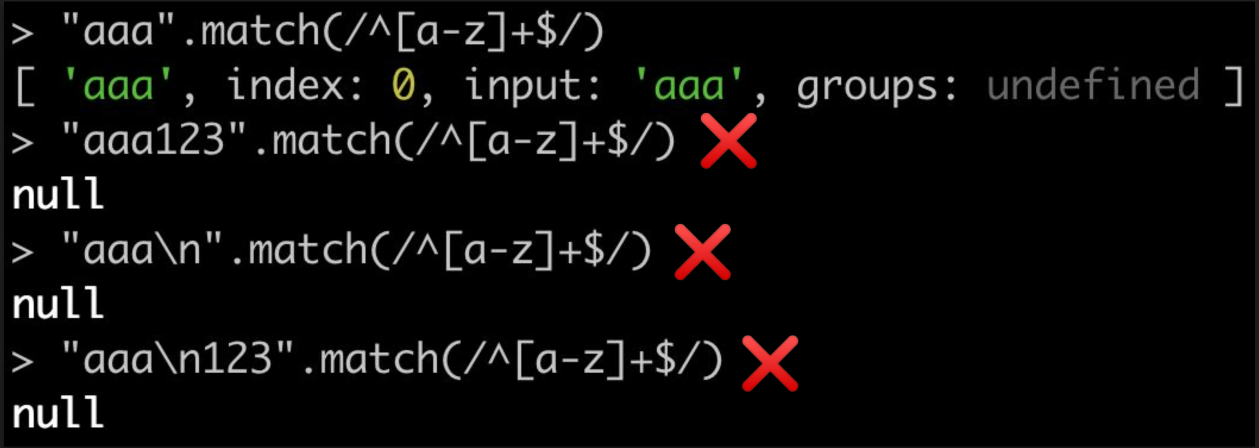
```python
import re
msg = 'Entity "test" is not available'
assert re.match(r'^Entity ".+" is not available$', msg)
```



$ asserts position at the end of the string, or before the line terminator right at the end of the string (if any) ?

We are Not the Same

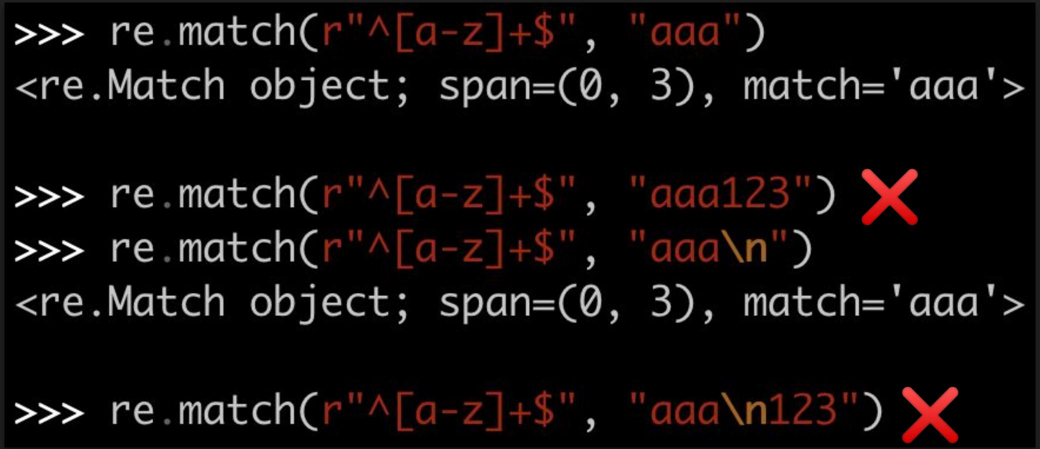
JavaScript



```
> "aaa".match(/^[a-z]+$/)
[ 'aaa', index: 0, input: 'aaa', groups: undefined ]
> "aaa123".match(/^[a-z]+$/) ❌
null
> "aaa\n".match(/^[a-z]+$/) ❌
null
> "aaa\n123".match(/^[a-z]+$/) ❌
null
```

We are Not the Same

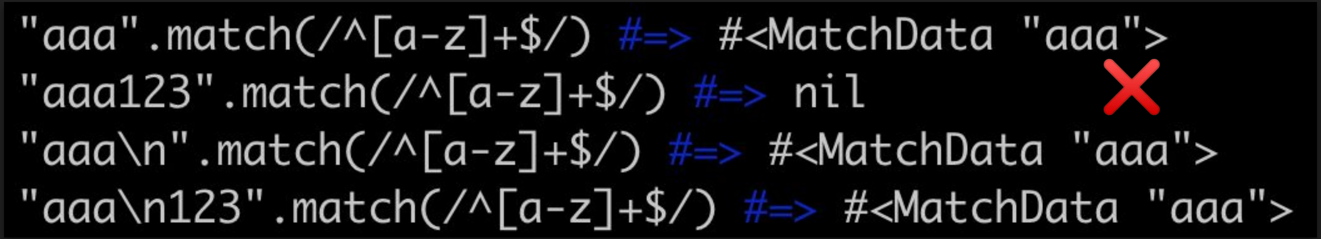
Python



```
>>> re.match(r"^[a-z]+$", "aaa")
<re.Match object; span=(0, 3), match='aaa'>

>>> re.match(r"^[a-z]+$", "aaa123") ❌
>>> re.match(r"^[a-z]+$", "aaa\n")
<re.Match object; span=(0, 3), match='aaa'>

>>> re.match(r"^[a-z]+$", "aaa\n123") ❌
```

We are Not the Same

Ruby



```
"aaa".match(/^[a-z]+$/) #=> #<MatchData "aaa">
"aaa123".match(/^[a-z]+$/) #=> nil          ❌
"aaa\n".match(/^[a-z]+$/) #=> #<MatchData "aaa">
"aaa\n123".match(/^[a-z]+$/) #=> #<MatchData "aaa">
```

We are Not the Same

/^[a-z]+$/

	JavaScript	Python	Ruby
"aaa"	✅	✅	✅
"aaa123"	❌	❌	❌
"aaa\n"	❌	✅	✅
"aaa\n123"	❌	❌	✅

2. REcollapse

Redefining the Impossible

- How to bypass most user input validations?
- How to leverage user input transformations?

Fuzz the parameters. In a smart way.

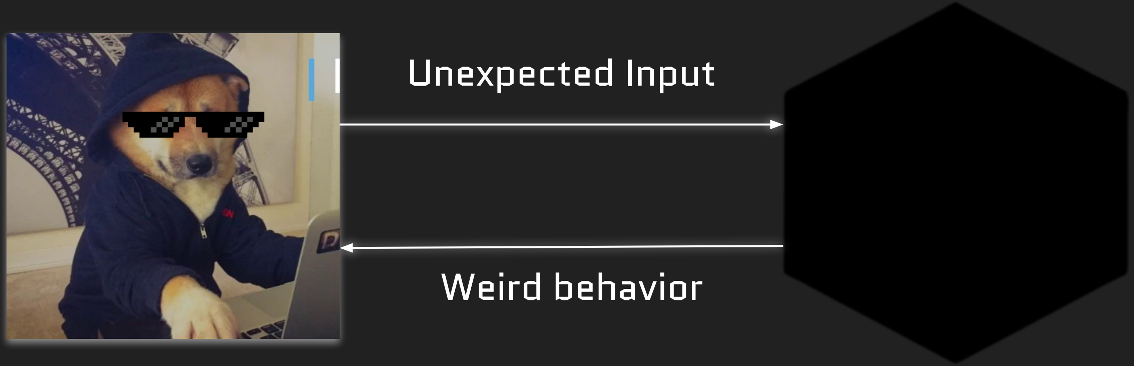
Redefining the Impossible

Let’s start with the initial scenario.

https://example.com/redirect?url=https://legit.example.com ✅

https://example.com/redirect?url=https://evil.com ❌

Probing the Unknown



The REcollapse Technique

1. Identify the regex pivot positions
   a. Starting & termination positions
   b. Separator positions
   c. Normalization positions

2. Fuzz positions with all possible bytes

3. Analyze the responses

The REcollapse Technique

```
https://example.com/redirect?url=$https://legit.example.com$
```

Starting position

Termination position

The REcollapse Technique

```
https://example.com/redirect?url=https$:$/$/$legit$.$example$.$com
```

Separator positions

The REcollapse Technique

```
https://example.com/redirect?url=https://l$git.ex$mple.c$m
```

Normalization positions

Typically vowels

A á ^{a} ã ⓐ $\longrightarrow$ a

The REcollapse Technique

```
https://example.com/redirect?url=$https$:$/$/$l$git$.$ex$mple$.$c$m$
```

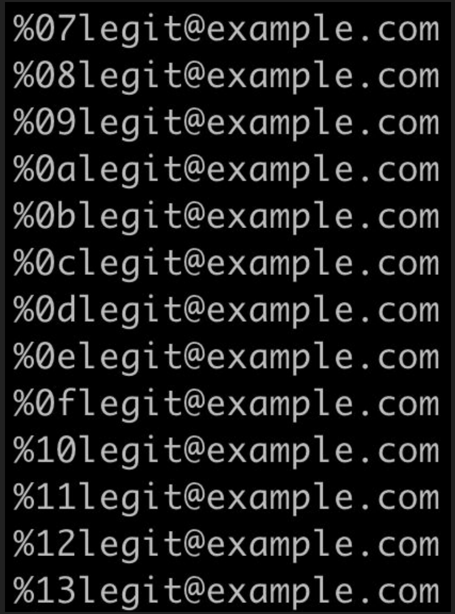
Fuzz all positions from **%00** to **%ff** ⚡

More Examples

```
https://legit.example.com → $https$:$/$/$l$git$.$ex$mple$.$c$m$

legit@example.com → $l$git$@$ex$mple$.$c$m$

user_name→   $us$r$_$n$me$

<a href=x>y</a> → $<$$$ $hr$f$=$$$>$$$<$/$$$>$
```

REcollapse Tool

- Helper tool capable of generating inputs according to these rules
- Supports multiple fuzzing sizes and encodings
- Easy to paste on Burp or other tools
- Available at https://github.com/0xacb/recollapse



```
%07legit@example.com
%08legit@example.com
%09legit@example.com
%0alegit@example.com
%0blegit@example.com
%0clegit@example.com
%0dlegit@example.com
%0elegit@example.com
%0flegit@example.com
%10legit@example.com
%11legit@example.com
%12legit@example.com
%13legit@example.com
```

Demo

3. Mysterious Bugs

What to Look for?

Literally anything that gets validated,
sanitized, normalized, used in queries, etc.

**This will open the door
to mysterious bugs.**

Uncovering Mysterious Bugs

1. Set your goal (e.g. ATO)
2. Pick your target field (e.g. email)
3. Identify all flows that consume it
4. For every endpoint: `REcollapse`
5. Analyze all response codes. Any successful response?
   a. Is the regex always the same in all endpoints? Usually not
   b. Pick a weird byte that went through

Uncovering Mysterious Bugs

6. Go through all the flows from step 3

   Recovery, login, signup, OAuth, SSO, email change & confirmation (depends on target field)

7. Hopefully, you just found a mysterious bug
   a. Look for errors and weird behaviors
   b. Try to realize the impact or an attack scenario
   c. If not, go back to step 5b or 1 / 2

4. Real-world Examples

1. Interaction-based ATO via Redirect

`https://login.redacted.com/auth?url=https://mail.redacted.com 302`

`Location: https://mail.redacted.com/?token=13371337...`

- After/If the user is logged in, it redirects to `url` with an auth `token` parameter
- As an attacker, we want to steal the auth `token` parameter to perform ATO
- There's some sort of validation (regex) that only allows `redacted.com` and `subdomains` of it

1. Interaction-based ATO via Redirect

```
url=https://evil.com 403

url=https://redacted.com.evil.com 403

url=https://redacted.com@evil.com 403
```

Now what? 🤔

1. Interaction-based ATO via Redirect

- Fuzzing `url=https://redacted.com$evil.com` from `%00` to `%ff` (1 byte) returns no useful `302` > only `# / ?`
- Fuzzing `%00%00` to `%ff%ff` (2 bytes) returns a nice `302` with `%3b%40`
- We can send a link to the victim and exfil a legitimate token to perform ATO

```
Location: https://redacted.com;@evil.com
```

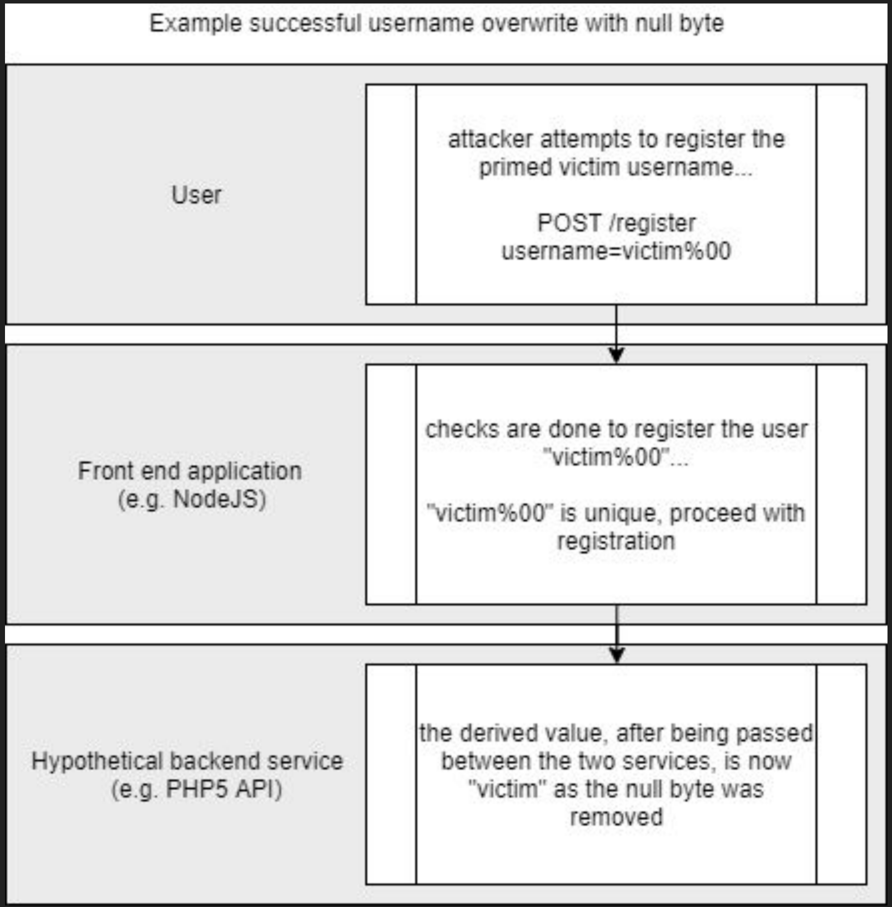
2. Null Boy

- We were fuzzing a target with this technique
- @samwcyo / zlz noticed that a %00 on a signup request would reveal a weird behavior

Original blog post

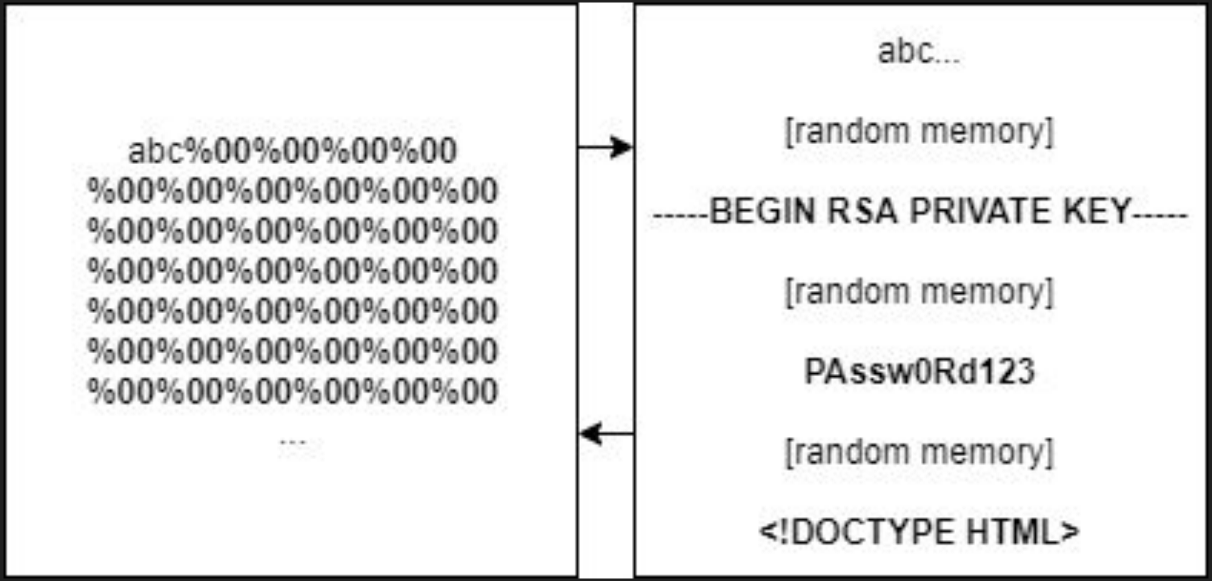
Filling in the Blanks: Exploiting Null Byte Buffer Overflow for a $40,000 Bounty (samcurry.net)

2. Null Boy


Example successful username overwrite with null byte
User
attacker attempts to register the primed victim username...
POST /register
username=victim%00
Front end application
(e.g. NodeJS)
checks are done to register the user "victim%00"...
"victim%00" is unique, proceed with registration
Hypothetical backend service
(e.g. PHP5 API)
the derived value, after being passed between the two services, is now "victim" as the null byte was removed

2. Null Boy

- Sign up as victim%00@domain.com would return victimL@domain.com



3. REcache Deception

- https://redacted.com/wp-json/v1/user 200

```
{
   "username": "xxxxxxxxx",
   "api_token": "xxxxxxxxx"
}
```

- https://redacted.com/wp-json/v1/user.css 404

```
              [...]                        .pdf  404
              [...]                        .js   404
```

3. REcache Deception

- Caching rules are usually regex-based
- A static extension is not enough these days to perform web cache deception
- We need to enforce the correct `Content-Type` in the response
- Let’s fuzz it!

3. REcache Deception

- Fuzzing https://redacted.com/wp-json/v1/user$.[extension] from %00 to %ff and well-known extensions returned 200 with %23 [#] and %3f [?]

  Age: 35, X-Cache: Hit

https://redacted.com/wp-json/v1/user%23.pdf

We can send a link to a logged-in victim that will request this URL, and then we just need to access the cached content from our end and steal the api_token.

4. Username Confusion

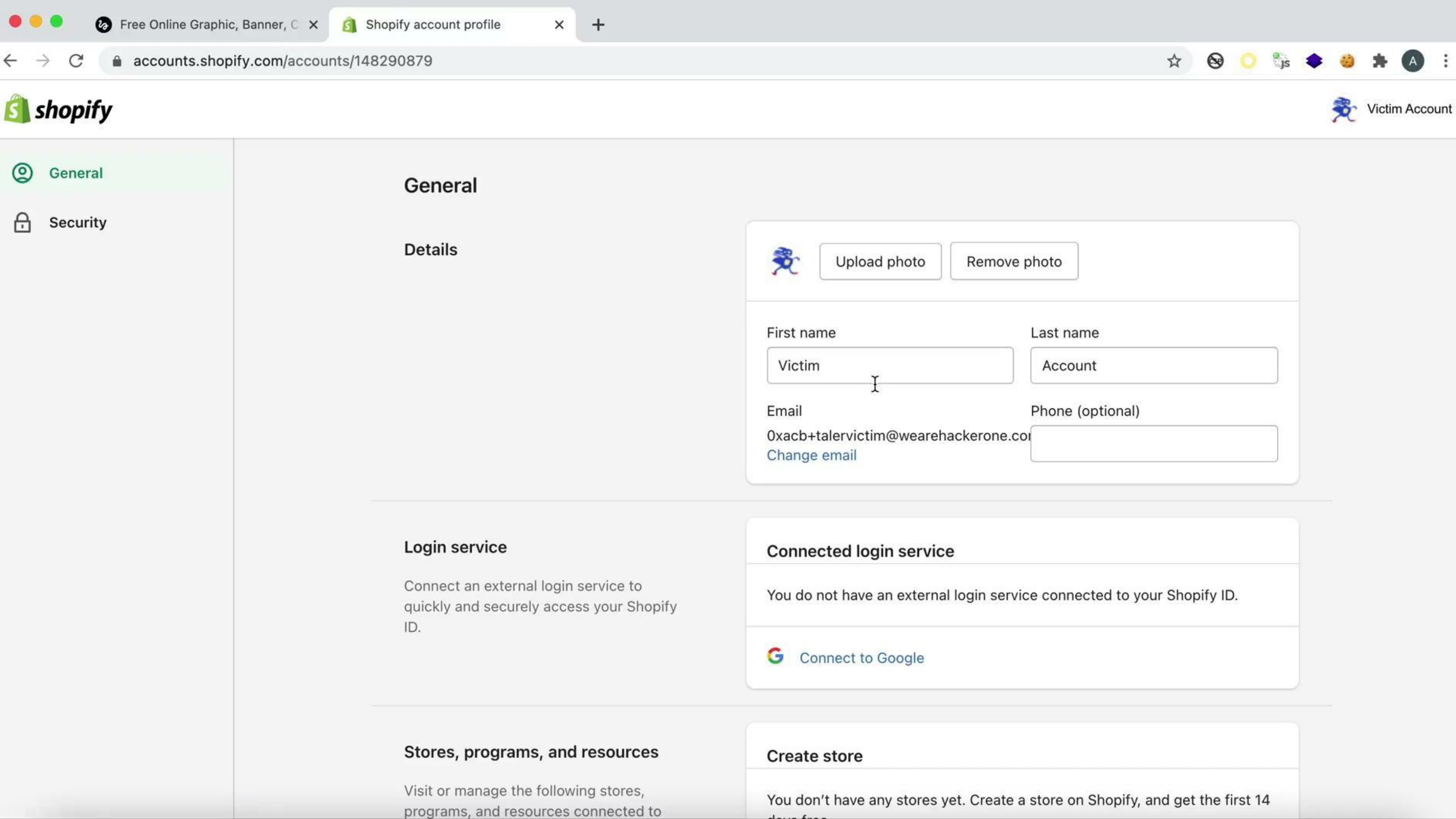
Waiting for permission to make this one public. Will update later.

5. Zero-interaction ATO [OAuth]

- Shopify offers a “Signup/Login with Shopify” OAuth mechanism
- OAuth scope includes email address to login in multiple applications
- In taler.app, the email address doesn’t need to be verified to create an account
- If the email already exists, you can’t login or sign up on Shopify

5. Zero-interaction ATO [OAuth]

- Let's fuzz the email change request on accounts.shopify.com
    - Proper regex in place, no weird characters allowed ❌
- Fuzzing the signup request on accounts.shopify.com:
    - vict(i)m@domain.com goes through ✅
- Login with Shopify in this state on taler.app
- Successful ATO



Free Online Graphic, Banner, C

Shopify account profile

accounts.shopify.com/accounts/148290879

shopify

Victim Account

General

Security

General

Details

Upload photo

Remove photo

First name	Last name
Victim	Account

Email

0xacb+talervictim@wearehackerone.co

Change email

Phone (optional)

Login service

Connect an external login service to quickly and securely access your Shopify ID.

Connected login service

You do not have an external login service connected to your Shopify ID.

Connect to Google

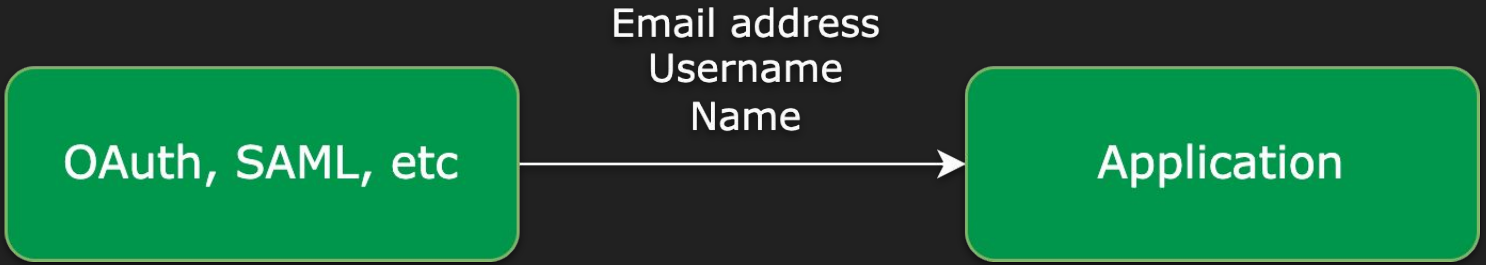
Stores, programs, and resources

Visit or manage the following stores, programs, and resources connected to

Create store

You don't have any stores yet. Create a store on Shopify, and get the first 14

5. Zero-interaction ATO [OAuth]



Normalization is often used in these flows.

6. Zero-interaction ATO takeover [Recovery]

- Target is an email provider
- Our goal is to ATO a victim@target.com inbox without any interaction
- People can sign up as username@target.com or use the current email address
- Let’ explore all the flows

Recovering victim@target.com will send a code to a redacted email address:

```
***********@redacted.com
```

6. Zero-interaction ATO takeover [Recovery]

Adding victim@target.com as attacker@target.com recovery email:

- Will require email verification but...
- It results in a change in the flow of https://redacted.target.com/recovery if we submit victim@target.com

Recovering victim@target.com returns now multiple emails:

1. victim@target.com itself!
2. ***********@redacted.com

6. Zero-interaction ATO takeover [Recovery]

- Some sort of regex was matching `@target.com` in order to distinguish both account types
- After fuzzing the email parameter, some special characters were displaying the same recovery email addresses: `victim@target.c./.o./m`

6. Zero-interaction ATO takeover [Recovery]

Adding a recovery email address as victim@target.com.domain.com will:

- Show up as a recovery email of the attacker's account
- But as option 2 we still have ***********@redacted.com available

After recovering the code via email to victim@target.com.domain.com:

Select an account:

attacker@target.com

victim@target.com ✅

Takeaways

- Developers: always test/fuzz your regex, or rely on well-known libraries
- Simple input modifications can result in great damage
    - Fuzz by flipping or adding bytes ⚡
- Black-box regex testing is still not very touched
    - Creative and manual work. Go for it 💰
- Regex behavior can reveal information about libraries, languages, etc
- If something is being validated and you can bypass it...
    - Think about the impact and you'll see the big picture! 🖼️

Special thanks

@regala_ / fisher
@0xz3z4d45
@jllis
@samwcyo / zlz
@yassineaboukir
@0xteknogeek

@ethiack team
@0xdisturbance team
@hacker0x01 team