How "../sms" could bypass Authy 2 Factor Authentication

The first part defines Format Injection and explains interesting but low severity bug in Duo Web SDK. (https://sakurity.com/blog/2015/03/03/duo_format_injection.html)

Update 20 March Authy contacted me to clarify that not everybody was vulnerable, and vulnerable API libraries **were limited to Node.JS by Daniel Barnes, Authy.NET by Devin Martin and Authy OpenVPN**.

Meanwhile we audited another popular 2FA provider and found a High-severity format injection in Authy API. In fact the root of the problem was default Sinatra dependency “rack-protection”! I responsibly disclosed this vulnerability to Authy on February 8 and worked with them to fix the issue that same day.

There are two API calls:

1. The client requests new token: `https://api.authy.com/protected/json/sms/AUTHY_ID?api_key=KEY` where AUTHY_ID is publicly available identifier associated with current user account. Expected response: `{"success":true,"message":"SMS token was sent","cellphone":"+1-XXX-XXX-XX85"}` with 200 status.
2. The user sends the token back and the client verifies if the token is valid with `https://api.authy.com/protected/json/verify/SUPPLIED_TOKEN/AUTHY_ID?api_key=KEY` and authenticates with second factor if API responds with 200 status (body is ignored): `{"success":true,"message":"Token is valid.","token":"is valid"}`

Authy-node does not encode token from user params

There was a blatant bug in authy-node (**not an official library**, btw another popular **node library (https://www.npmjs.com/package/co-authy)** wasn’t vulnerable) - “token” supplied by the user was not URL encoded at all: `this._request("get",`

`"/protected/json/verify/" + token + "/" + id, {}, callback, qs);`

Which means by typing `VALID_TOKEN_FOR_OTHER_AUTHY_ID/OTHER_AUTH_ID#` we would overwrite the path and make the client send `/protected/json/verify/VALID_TOKEN_FOR_OTHER_AUTHY_ID/OTHER_AUTH_ID#/AUTH_ID`. Anything after hash `#` is ignored and Authy's response with 200 status for `/protected/json/verify/VALID_TOKEN_FOR_OTHER_AUTHY_ID/OTHER_AUTH_ID?api_key=KEY` let's the attacker in.

It's impossible to distinguish forged request from a valid one on the server side because `#/AUTHY_ID` is not sent.

Authy-python is vulnerable too

Then I noticed Python's urllib.quote doesn't escape slashes. Indeed, for some reason it escapes everything but slashes and it's **a documented feature (https://docs.python.org/2/library/urllib.html#urllib.quote)** - `urllib.quote("#?&=/")` returns `%23%3F%26%3D/`. Which means our "../sms" will not be encoded (/../ means "go one directory up").

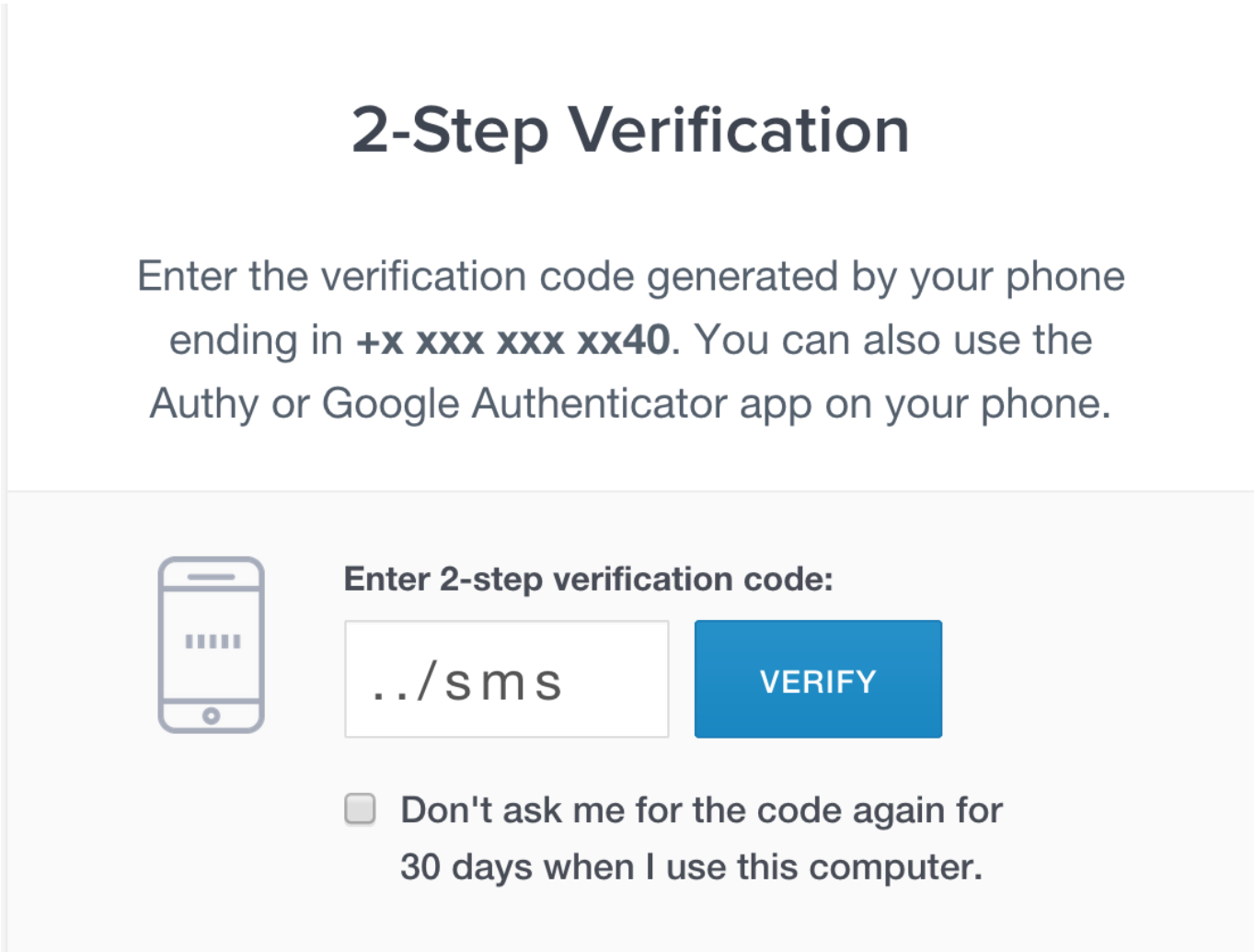
Web browsers parse /../, /%2e%2e/ and even /%252e%252e/ and go "one directory up", but web servers don't have to do it. Anyway, I tried and it worked - Authy API was removing directories before `/../`.

It introduces path traversal making attacker's job much easier - you only need to type `../sms` to turn /verify API call into /sms (`/verify/../sms/authy_id`) which will always return 200 status and will bypass 2FA.

No, wait. Everyone is vulnerable!

Few hours later I realized what made path traversal work: I recently read **Daniel's interview on Authy (https://stackshare.io/posts/how-authy-built-a-fault-tolerant-two-factor-authentication-service/)** and recalled it runs Sinatra, which uses rack-protection by default.

It turns out even URL encoding was futile - **path_traversal module in rack-protection (https://github.com/rkh/rack-protection/blob/master/lib/rack/protection/path_traversal.rb#L34)** was decoding %2f back to slashes! This literally affects every API running Sinatra and reading parameters from the path. This is also a great example how libraries or features that aim to add security actually introduce security vulnerabilities (see also **CSP for evil (https://homakov.blogspot.com/2014/01/using-content-security-policy-for-evil.html)** and **XSS auditor for evil (https://homakov.blogspot.com/2013/02/hacking-with-xss-auditor.html)**)



1. The attacker types `../sms` in the SMS token field
2. The client app encodes it as `..%2fsms` and makes an API call to Authy - `https://api.authy.com/protected/json/verify/..%2fsms/authy_id`
3. Path_traversal middleware decodes path to `https://api.authy.com/protected/json/verify/../sms/authy_id`, splits by slashes and removes the directory in front of `/..`.
4. Actual Authy API sees modified path `https://api.authy.com/protected/json/sms/authy_id`, simply sends another SMS to authy_id (the victim) and responds with 200 status and `{"success":true,"message":"SMS token was sent","cellphone":"+1-XXX-XXX-XX85"}`
5. All Authy SDK libraries consider 200 status as a successful response and let the attacker in. Even a custom integration most likely will look for `"success":true` in the JSON body, and our `/sms` response body has it. So the only secure way to verify the response is to search for `"token":"is valid"` substring (which is what Authy libraries do now).

Yes, the attacker was able to bypass 2 factor authentication on any website using Authy with something as simple as ../sms in the token field!

Timeline: reported on Feb 8, the path_traversal module was patched right away and we waited for a month to let authy-node users to update.

Update 20 March Authy contacted me to clarify that not everybody was vulnerable, and vulnerable API libraries **were limited to Node.JS by Daniel Barnes, Authy.NET by Devin Martin and Authy OpenVPN**.

This is another example of format injection and why you need to treat URLs as a format like JSON or XML. **Read our first post on format injection in Duo Security Web SDK. (https://sakurity.com/blog/2015/03/03/duo_format_injection.html)**

Mar 15, 2015 • Egor Homakov (**@homakov (https://twitter.com/homakov)**)