Evilginx 2 - Next Generation of Phishing 2FA Tokens

By Kuba Gretzky

Published: 2018-07-26 · Archived: 2026-04-29 02:11:06 UTC

It's been over a year since the first release of Evilginx and looking back, it has been an amazing year. I've received tons of feedback, got invited to WarCon by @antisnatchor (thanks man!) and met amazing people from the industry. A year ago, I wouldn't have even expected that one day **Kevin Mitnick** would showcase Evilginx in his live demos around the world and Techcrunch would write about it!

At WarCon I met the legendary @evilsocket (he is a really nice guy), who inspired me with his ideas to learn GO and rewrite Evilginx as a standalone application. It is amazing how GO seems to be ideal for offensive tools development and bettercap is its best proof!

This is where Evilginx is now. No more nginx, just pure evil. My main goal with this tool's release was to focus on minimizing the installation difficulty and maximizing the ease of use. Usability was not necessarily the strongest point of the initial release.

Updated instructions on usage and installation can always be found up-to-date on the tool's official GitHub project page. In this blog post I only want to explain some general concepts of how it works and its major features.

Update: Check also version 2.1 release post

TL;DR What am I looking at?

Evilginx is an attack framework for setting up phishing pages. Instead of serving templates of sign-in pages lookalikes, **Evilginx becomes a relay between the real website and the phished user**. Phished user interacts with the real website, while Evilginx captures all the data being transmitted between the two parties.

Evilginx, being the man-in-the-middle, captures not only usernames and passwords, but also **captures authentication tokens sent as cookies**. Captured authentication tokens allow the attacker to **bypass any form of 2FA enabled on user's account** (except for U2F - more about it further below).

Even if phished user has 2FA enabled, **the attacker, outfitted with just a domain and a VPS server, is able to remotely take over his/her account**. It doesn't matter if 2FA is using SMS codes, mobile authenticator app or recovery keys.

Take a look at the video demonstration, showing how attacker's can **remotely hack an Outlook account with enabled 2FA**.

> **Disclaimer:** Evilginx project is released for educational purposes and should be used only in demonstrations or legitimate penetration testing assignments with written permission from to-be-phished parties. Goal is to show that 2FA is not a silver bullet against phishing attempts and people should be aware that their accounts can be compromised, nonetheless, if they are not careful.

[>> Download Evilginx 2 from GitHub <<](https://github.com/kgretzky/evilginx2)

Remember - 2FA is not a silver bullet against phishing!

2FA is very important, though. This is what head of Google Threat Intelligence had to say on the subject:

Old phishing tactics

Common phishing attacks, we see every day, are HTML templates, prepared as lookalikes of popular websites' sign-in pages, luring victims into disclosing their usernames and passwords. When the victim enters his/her username and password, the credentials are logged and attack is considered a success.

> I love digging through certificate transparency logs. Today, I saw a fake Google Drive landing page freshly registered with Let's Encrypt. It had a hardcoded picture/email of presumably the target. These can be a wealth of info that I recommend folks checking out. pic.twitter.com/PRweQsgHKD
>
> — Justin Warner (@sixdub) July 22, 2018

This is where 2FA steps in. If phished user has 2FA enabled on their account, the attacker would require an additional form of authentication, to supplement the username and password they intercepted through phishing. That additional form of authentication may be SMS code coming to your mobile device, TOTP token, PIN number or answer to a question that only the account owner would know. Attacker not having access to any of these **will never be able to successfully authenticate** and login into victim's account.

Old phishing methods which focus solely on capturing usernames and passwords are **completely defeated by 2FA**.

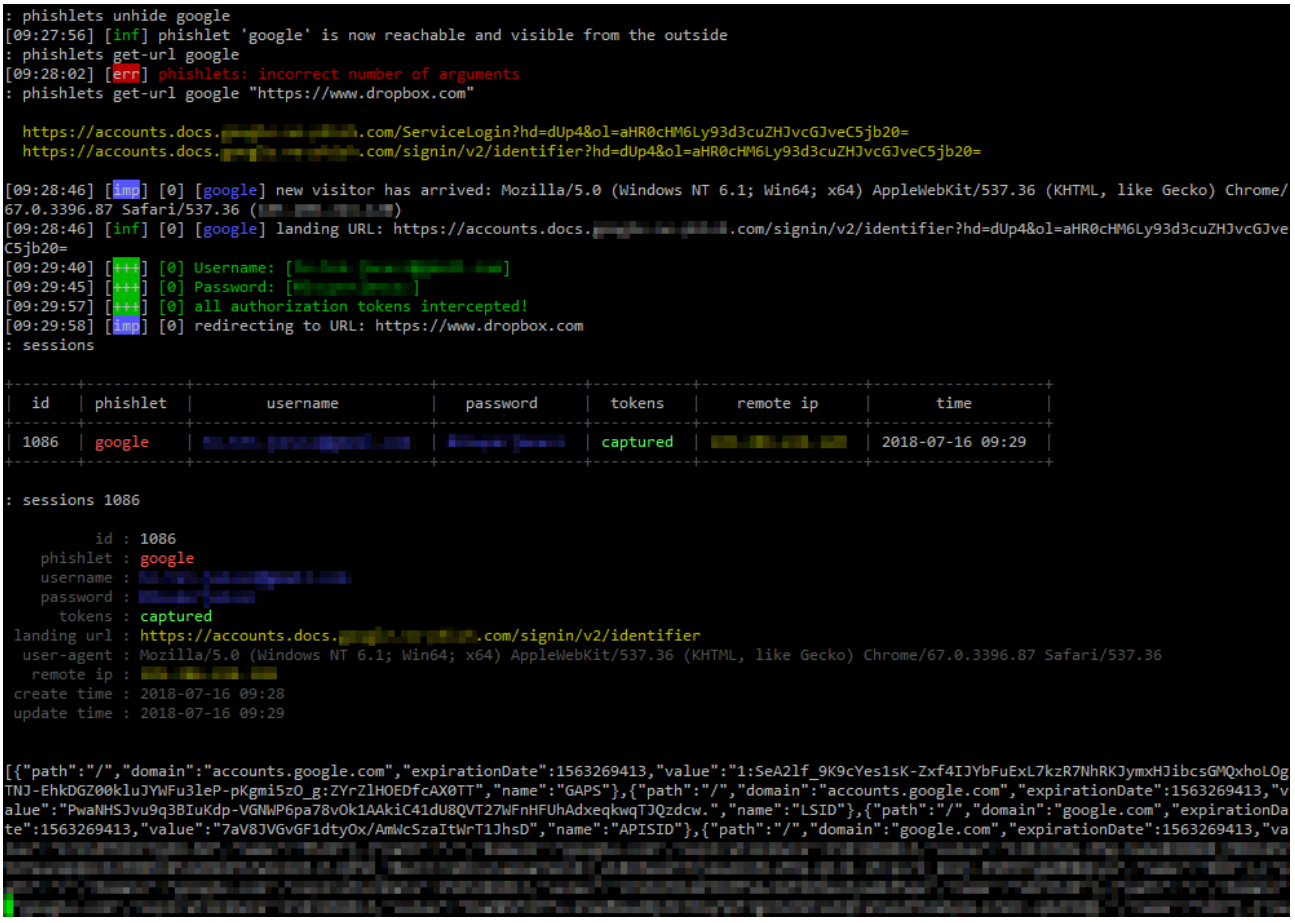
Phishing 2.0

What if it was possible to lure the victim not only to disclose his/her username and password, but also to provide the answer to any 2FA challenge that may come after the credentials are verified? Intercepting a single 2FA answer would not do the attacker any good. Challenge will change with every login attempt, making this approach useless.

After each successful login, website generates an **authentication token** for the user's session. This token (or multiple tokens) is sent to the web browser as a cookie and is saved for future use. From that point, every request sent from the browser to the website will contain that session token, sent as a cookie. This is how websites recognize authenticated users after successful authentication. They do not ask users to log in, every time when page is reloaded.

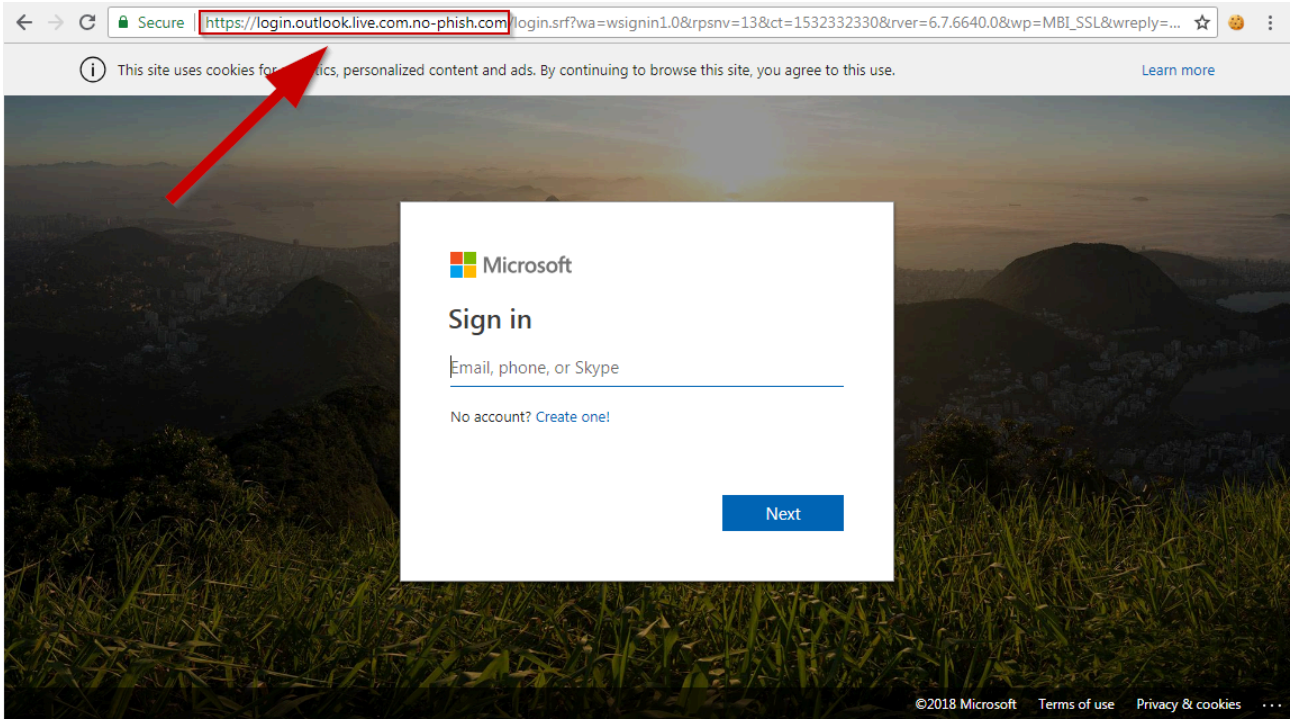
This session token cookie is pure gold for the attacker. If you export cookies from your browser and import them into a different browser, on a different computer, in a different country, you will be authorized and get full access to the account, without being asked for usernames, passwords or 2FA tokens.

This is what it looks like, in Evilginx 2, when **session token cookie is successfully captured**:



Now that we know how valuable the session cookie is, how can the attacker intercept it remotely, without having physical access to the victim's computer?

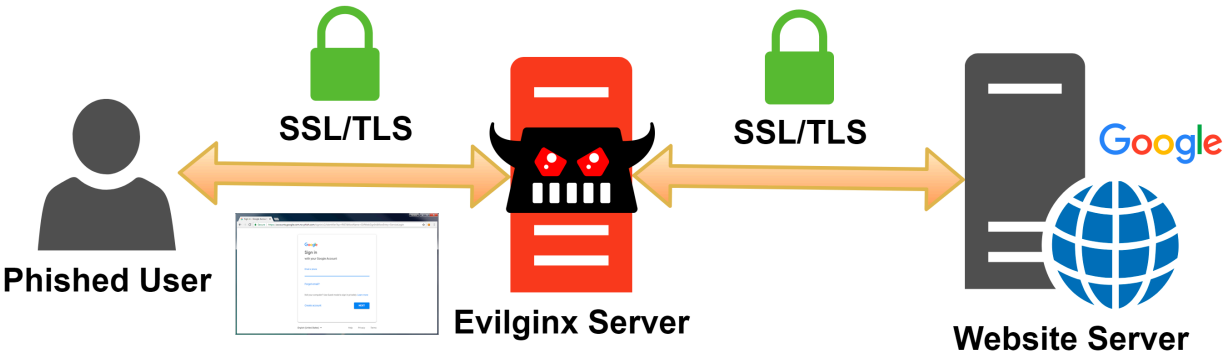
Common phishing attacks rely on creating **HTML templates** which **take time to make**. Most work is spent on making them look good, being responsive on mobile devices or properly obfuscated to evade phishing detection scanners.



Evilginx takes the attack one step further and **instead of serving its own HTML lookalike pages, it becomes a web proxy**. Every packet, coming from victim's browser, is intercepted, modified and forwarded to the real website. The same happens with response packets, coming from the website; they are intercepted, modified and sent back to the victim. With Evilginx **there is no need to create your own HTML templates**. On the victim side everything looks as if he/she was communicating with the legitimate website. User has no idea idea that Evilginx **sits as a man-in-the-middle**, analyzing every packet and **logging usernames, passwords and, of course, session cookies**.

You may ask now, what about encrypted HTTPS connection using SSL/TLS that prevents eavesdropping on the communication data? Good question. Problem is that the victim is only talking, over HTTPS, to Evilginx server and not the true website itself. Evilginx initiates its own HTTPS connection with the victim (using its own SSL/TLS certificates), receives and decrypts the packets, only to **act as a client itself** and establish its own HTTPS connection with the destination website, where it sends the re-encrypted packets, as if it was the victim's browser itself. This is how the trust chain is broken and the **victim still sees that green lock icon next to the address bar**, in the browser, thinking that everyone is safe.

When the victim enters the credentials and is asked to provide a 2FA challenge answer, they are still talking to the real website, with Evilginx relaying the packets back and forth, sitting in the middle. Even while being phished, the **victim will still receive the 2FA SMS code** to his/her mobile phone, because he/she is talking to the real website (just through a relay).



After the 2FA challenge is completed by the victim and the website confirms its validity, website generates the session token, which it returns in form of a cookie. This **cookie is intercepted by Evilginx** and saved. Evilginx determines that authentication was a success and redirects the victim to any URL it was set up with (online document, video etc.).

At this point the **attacker holds all the keys to the castle** and is able to use the victim's account, **fully bypassing 2FA protection**, after importing the session token cookies into his web browser.

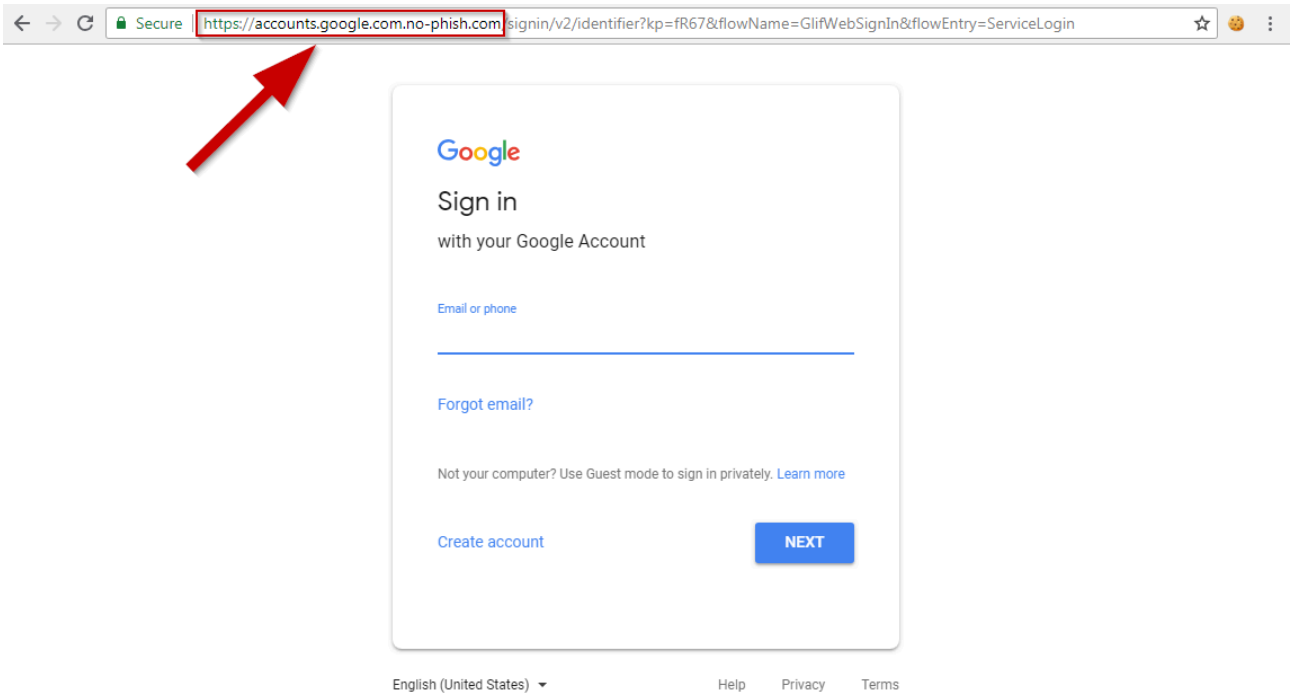
Be aware that: **Every sign-in page, requiring the user to provide their password, with any form of 2FA implemented, can be phished using this technique!**

How to protect yourself?

There is one major flaw in this phishing technique that anyone can and should exploit to protect themselves - **the attacker must register their own domain**.

By registering a domain, attacker will try to make it look as similar to real, legitimate domain as possible. For example if the attacker is targeting Facebook (real domain is `facebook.com`), they can, for example, register a domain `faceboook.com` or `faceb00k.com`, maximizing their chances that phished victims won't spot the difference in the browser's address bar.

That said - **always check the legitimacy of website's base domain, visible in the address bar, if it asks you to provide any private information**. By base domain I mean the one that precedes the top-level domain.



As an example, imagine this is the URL and the website, you arrived at, asks you to log into Facebook:

```
https://en-gb.facebook.cdn.global.faceboook.com/login.php
```

The top-level domain is `.com` and the base domain would be the preceeding word, with next `.` as a separator. Combined with TLD, that would be `faceboook.com`. When you verify that `faceboook.com` is not the real `facebook.com`, you will know that someone is trying to phish you.

As a side note - **Green lock icon seen next to the URL, in the browser's address bar, does not mean that you are safe!**

Green lock icon only means that the website you've arrived at, **encrypts the transmission between you and the server**, so that no-one can eavesdrop on your communication. **Attackers can easily obtain SSL/TLS certificates** for their phishing sites and give you a false sense of security with the ability to display the green lock icon as well.

Figuring out if the base domain you see is valid, sometimes may not be easy and leaves room for error. It became even harder with the support of Unicode characters in domain names. This made it possible for attackers to register domains with special characters (e.g. in Cyrillic) that would be lookalikes of their Latin counterparts. This technique recieved a name of a homograph attack.

As a quick example, an attacker could register a domain `facebooĸ.com`, which would look pretty convincing even though it was a completely different domain name (`ĸ` is not really `k`). It got even worse with other Cyrillic characters, allowing for `ebay.com` vs `ebay.com`. The first one has an Cyrillic counterpart for `a` character, which looks exactly the same.

Major browsers were fast to address the problem and added special filters to prevent domain names from being displayed in Unicode, when suspicious characters were detected.

If you are interested in how it works, check out the IDN spoofing filter source code of the Chrome browser.

Now you see that verifying domains visually is not always the best solution, especially for big companies, where it often **takes just one employee to get phished** and allow attackers to **steal vast amounts of data**.

This is why FIDO Alliance introduced U2F (*Universal 2nd Factor Authentication*) to allow for unphishable 2nd factor authentication.

In short, you have a **physical hardware key** on which you just press a button when the website asks you to. Additionally it may ask you for account password or a complementary 4 digit PIN. The website talks directly with the hardware key plugged into your USB port, with the web browser as the channel provider for the communication.


YubiKey slide

What is different with this form of authentication, is that **U2F protocol is designed to take the website's domain as one of the key components in negotiating the handshake**. This means that if the domain in the browser's address bar, does not match the domain used in the data transmission between the website and the U2F device, the **communication will simply fail**. This solution leaves no room for error and is **totally unphishable** using Evilginx method.

Citing the vendor of U2F devices - Yubico (who co-developed U2F with Google):

> With the YubiKey, user login is bound to the origin, meaning that only the real site can authenticate with the key. The authentication will fail on the fake site even if the user was fooled into thinking it was real. This greatly mitigates against the increasing volume and sophistication of phishing attacks and stops account takeovers.

It is important to note here that Markus Vervier (@marver) and Michele Orrù (@antisnatchor) did demonstrate a technique on how an attacker can attack U2F devices using the newly implemented WebUSB feature in modern browsers (which allows websites to talk with USB connected devices). It is also important to mention that Yubico, the creator of popular U2F devices YubiKeys, tried to steal credit for their research, which they later apologized for.

You can find the list of all websites supporting U2F authentication here.

Coinciding with the release of Evilginx 2, WebAuthn is coming out in all major web browsers. It will introduce the new FIDO2 password-less authentication standard to every browser. Chrome, Firefox and Edge are about to receive full support for it.

To wrap up - if you often need to log into various services, make your life easier and **get a U2F device!** This will greatly improve your accounts' security.

Under the hood

Interception of HTTP packets is possible since Evilginx acts as an **HTTP server talking to the victim's browser** and, at the same time, acts as an **HTTP client for the website where the data is being relayed to**. To make it

possible, the victim has to be contacting Evilginx server through a custom phishing URL that will point to Evilginx server. Simply forwarding packets from victim to destination website would not work well and that's why Evilginx has to do some on-the-fly modifications.

In order for the phishing experience to be seamless, the proxy overcomes the following obstacles:

1. Making sure that the victim is not redirected to phished website's true domain.

Since the phishing domain will differ from the legitimate domain, used by phished website, relayed scripts and HTML data have to be carefully modified to **prevent unwanted redirection of victim's web browser**. There will be HTML submit forms pointing to legitimate URLs, scripts making AJAX requests or JSON objects containing URLs.

Ideally the most reliable way to solve it would be to perform regular expression string substitution for any occurrence of `https://legit-site.com` and replacing it with `https://our-phishing-site.com`. Unfortunately this is not always the case and it requires some trial and error kung-fu, working with web inspector to track down all strings the proxy needs to replace to not break website's functionality. If target website uses multiple options for 2FA, each route has to be inspected and analyzed.

For example, there are JSON objects transporting escaped URLs like `https:\/\/legit-site.com`. You can see that this will definitely not trigger the regexp mentioned above. If you replaced all occurrences of `legit-site.com` you may break something by accident.

2. Responding to DNS requests for multiple subdomains.

Websites will often make requests to multiple subdomains under their official domain or even use a totally different domain. In order to proxy these transmissions, Evilginx has to **map each of the custom subdomains to its own IP address**.

Previous version of Evilginx required the user to set up their own DNS server (e.g. `bind`) and set up DNS zones to properly handle DNS A requests. This generated a lot of headache on the user part and was only easier if the hosting provider (like Digital Ocean) provided an easy-to-use admin panel for setting up DNS zones.

With Evilginx 2 **this issue is gone**. Evilginx now runs its own in-built DNS server, listening on port 53, which acts as a nameserver for your domain. All you need to do is set up the nameserver addresses for your domain (`ns1.yourdomain.com` and `ns2.yourdomain.com`) to point to your Evilginx server IP, in the admin panel of your domain hosting provider. Evilginx will handle the rest on its own.

3. Modification of various HTTP headers.

Evilginx modifies HTTP headers sent to and received from the destination website. In particular the `Origin` header, in AJAX requests, will always hold the URL of the requesting site in order to comply with CORS. **Phishing sites will hold a phishing URL as an origin**. When request is forwarded, the destination website will receive an invalid origin and will not respond to such request. Not replacing the phishing hostname with the legitimate one in the request would make it also easy for the website to notice suspicious behavior. Evilginx automatically changes `Origin` and `Referer` fields on-the-fly to their legitimate counterparts.

Same way, to avoid any conflicts with CORS from the other side, Evilginx makes sure to set the `Access-Control-Allow-Origin` header value to `*` (if it exists in the response) and removes any occurrences of `Content-Security-Policy` headers. This guarantees that **no request will be restricted by the browser** when AJAX requests are made.

Other header to modify is `Location`, which is set in HTTP `302` and `301` responses to redirect the browser to different location. Naturally the value will come with legitimate website URL and Evilginx makes sure this location is properly switched to corresponding phishing hostname.

4. Cookies filtering.

It is common for websites to manage cookies for various purposes. Each cookie is assigned to a specific domain. Web browser's task is to automatically send the stored cookie, with every request to the domain, the cookie was assigned to. Cookies are also sent as HTTP headers, but I decided to make a separate mention of them here, due to their importance. Example cookie sent from the website to client's web browser would look like this:

```
Set-Cookie: qwerty=219ffwef9w0f; Domain=legit-site.com; Path=/; Expires=Wed, 30 Aug 2019 00:00:00 GMT
```

As you can see the cookie will be set in client's web browser for `legit-site.com` domain. Since the phishing victim is only talking to the phishing website with domain `our-phishing-site.com`, such cookie will never be saved in the browser, because of the fact the **cookie domain differs from the one the browser is communicating with**. Evilginx will parse every occurrence of `Set-Cookie` in HTTP response headers and modify the domain, replacing it with the phishing one, as follows:

```
Set-Cookie: qwerty=219ffwef9w0f; Domain=our-phishing-site.com; Path=/;
```

Evilginx will also remove expiration date from cookies, if the expiration date does not indicate that the cookie should be deleted from browser's cache.

Evilginx also sends its own cookies to manage the victim's session. These cookies are filtered out from every HTTP request, to prevent them from being sent to the destination website.

5. SSL splitting.

As the whole world of world-wide-web migrates to serving pages over secure HTTPS connections, phishing pages can't be any worse. Whenever you pick a hostname for your phishing page (e.g. `totally.not.fake.linkedin.our-phishing-domain.com`), Evilginx will **automatically obtain a valid SSL/TLS certificate from LetsEncrypt** and provide responses to ACME challenges, using the in-built HTTP server.

This makes sure that **victims will always see a green lock icon next to the URL address bar**, when visiting the phishing page, comforting them that everything is secured using *"military-grade"* encryption!

6. Anti-phishing tricks

There are rare cases where websites would employ defenses against being proxied. One of such defenses I uncovered during testing is using javascript to check if `window.location` contains the legitimate domain. These detections may be easy or hard to spot and much harder to remove, if additional code obfuscation is involved.

Improvements

The greatest advantage of Evilginx 2 is that it is now a standalone console application. There is no need to compile and install custom version of nginx, which I admit was not a simple feat. I am sure that using nginx site configs to utilize `proxy_pass` feature for phishing purposes was not what HTTP server's developers had in mind, when developing the software.

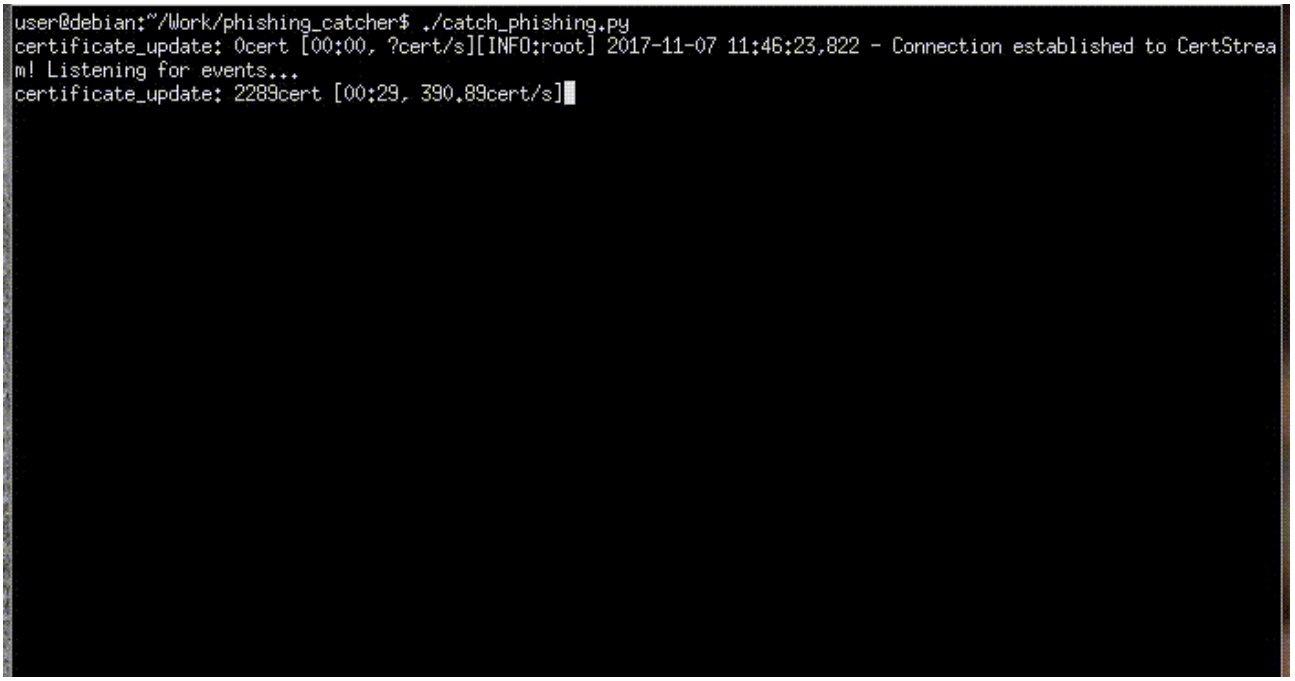
Evilginx 1 was pretty much a combination of several dirty hacks, duct taped together. Nonetheless it somehow worked!

Additionally to fully responsive console UI, here are the greatest improvements:

Tokenized phishing URLs

In previous version of Evilginx, entering just the hostname of your phishing URL address in the browser, with root path (e.g. `https://totally.not.fake.linkedin.our-phishing-domain.com/`), would still proxy the connection to the legitimate website. This turned out to be an issue, as I found out during development of Evilginx 2. Apparently once you obtain SSL/TLS certificates for the domain/hostname of your choice, **external scanners start scanning your domain**. Scanners gonna scan.

The scanners use public certificate transparency logs to scan, in real-time, all domains which have obtained valid SSL/TLS certifcates. With public libraries like CertStream, you can easily create your own scanner.



For some phishing pages, **it took usually one hour for the hostname to become banned and blacklisted by popular anti-spam filters** like Spamhaus. After I had three hostnames blacklisted for one domain, the whole domain got blocked. Three strikes and you're out!

I began thinking how such detection can be evaded. Easiest solution was to reply with faked response to every request for path `/`, but that would not work if scanners probed for any other path.

Then I decided that each phishing URL, generated by Evilginx, should come with a **unique token** in the URL as a GET parameter.

For example, Evilginx responds with redirection response when scanner makes a request to URL:

```
https://totally.not.fake.linkedin.our-phishing-domain.com/auth/signin
```

But it **responds with proxied phishing page**, instead, when the URL is properly tokenized, with a valid token:

```
https://totally.not.fake.linkedin.our-phishing-domain.com/auth/signin?tk=secret_l33t_token
```

When tokenized URL is opened, Evilginx **sets a validation cookie in victim's browser, whitelisting all subsequent requests**, even for the non-tokenized ones.

This works very well, but there is still risk that scanners will eventually scan tokenized phishing URLs when these get out into the interwebz.

Hiding your phishlets

This thought provoked me to find a solution that allows manual control over when the phishing proxy should respond with proxied website and when it should not. As a result, you can `hide` and `unhide` the phishign page

whenever you want. **Hidden phishing page will respond with a redirection 302 HTTP code**, redirecting the requester to predefined URL (Rick Astley's famous clip on Youtube is the default).

Temporarily hiding your phishlet may be useful when you want to use a URL shortener, to shorten your phishing URL (like `goo.gl` or `bit.ly`) or when you are sending the phishing URL via email and you don't want to trigger any email scanners, on the way.

Phishlets

Phishlets are new site configs. They are plain-text ruleset files, in YAML format, which are fed into the Evilginx engine. Phishlets define which subdomains are needed to properly proxy a specific website, what strings should be replaced in relayed packets and which cookies should be captured, to properly take over the victim's account. There is one phishlet for each phished website. You can deploy as many phishlets as you want, with each phishlet set up for a different website. Phishlets can be enabled and disabled as you please and at any point Evilginx can be running and managing any number of them.

I will do a better job than I did last time, when I released Evilginx 1, and I will try to explain the structure of a phishlet and give you brief insight into how phishlets are created (I promise to release a separate blog post about it later!).

I will dissect the LinkedIn phishlet for the purpose of this short guide:

```
 name: 'linkedin'
author: '@mrgretzky'
min_ver: '2.0.0'
proxy_hosts:
  - {
    phish_sub: 'www',
    orig_sub: 'www',
         domain: 'linkedin.com',
         session: true,
         is_landing: true
         }
sub_filters:
  - {
    hostname: 'www.linkedin.com',
         sub: 'www',
         domain: 'linkedin.com',
         search: 'action="https://{hostname}',
         replace: 'action="https://{hostname}',
         mimes: ['text/html', 'application/json']
         }
  - {
    hostname: 'www.linkedin.com',
         sub: 'www',
         domain: 'linkedin.com',
```

```
        search: 'href="https://{hostname}',
        replace: 'href="https://{hostname}',
        mimes: ['text/html', 'application/json']
        }
  - {
    hostname: 'www.linkedin.com',
        sub: 'www',
        domain: 'linkedin.com',
        search: '//{hostname}/nhome/',
        replace: '//{hostname}/nhome/',
        mimes: ['text/html', 'application/json']
        }
auth_tokens:
  - domain: 'www.linkedin.com'
    keys: ['li_at']
user_regex:
  key: 'session_key'
  re: '(.*)'
pass_regex:
  key: 'session_password'
  re: '(.*)'
landing_path:
  - '/uas/login'
```

First things first. I advise you to get familiar with YAML syntax to avoid any errors when editing or creating your own phishlets.

Starting off with simple and rather self-explanatory variables. `name` is the name of the phishlet, which would usually be the name of the phished website. `author` is where you can do some self promotion - this will be visible in Evilginx's UI when the phishlet is loaded. `version` is currently not supported, but will be very likely used when phishlet format changes in future releases of Evilginx, to provide some way of checking phishlet's compatibility with current tool's version.

Following that, we have `proxy_hosts`. This array holds an array of sub-domains that Evilginx will manage. This provides an array of all hostnames for which you want to intercept the transmission and gives you the capability to make on-the-fly packet modifications.

- `phish_sub` : subdomain name that will be prefixed in the phishlet's hostname. I advise to leave it the same as the original subdomain name, due to issues that may arise later when doing string replacements properly, as it often requires additional work to support custom subdomain names.
- `orig_sub` : the original subdomain name as used on the legitimate website.
- `domain` : website's domain that we are targeting.
- `session` : set this to `true` **ONLY** for subdomains that will return authentication cookies. This indicates which subdomain Evilginx should recognize as the one that will initiate the creation of Evilginx session and sets Evilginx session cookie for the domain name of this entry.

- `is_landing` : set this to `true` if you want this subdomain to be used in generation of phishing URLs later.

In the LinkedIn example, we only have one subdomain that we need to support, which is `www`. The phishing hostname for this subdomain will then be: `www.totally.not.fake.linkedin.our-phishing-domain.com`.

Next are `sub_filters`, which tell Evilginx all about string substitution magics.

- `hostname` : original hostname of the website, for which the substitution will take place.
- `sub` : subdomain name from the original hostname. This is will be only used as a helper string in substitutions that I will explain below.
- `domain` : domain name of the original hostname. Same as `sub` - used as a helper string in substitutions.
- `search` : the regular expression of what to search for in HTTP packet's body. You can use some variables in `{...}` that Evilginx will prefill for you. I listed all supported variables below.
- `replace` : the string that will act as a replacement for all occurrences of `search` regular expression matches. `{...}` variables are also supported here.
- `mimes` : an array of MIME types that will only be considered before doing search and replace. Any of these defined MIME types must show up in `Content-Type` header of the HTTP response, before Evilginx considers to do any substitutions, for that packet. Most common MIME types to use here are: `text/html`, `application/json`, `application/javascript` or `text/javascript`.
- `redirect_only` : use this `sub_filter` only if redirection URL is set in generated phishing URL (`true` or `false`).

The following is a list of bracket variables that you can use in `search` and `replace` parameters:

- `{hostname}` : a combination of subdomain, defined by `sub` parameter, and a domain, defined by `domain` parameter. In `search` field it will be translated to the original website's hostname (e.g. `www.linkedin.com`). In the `replace` field, it will be translated to corresponding phishing hostname of matching `proxy_hosts` entry (e.g. `www.totally.not.fake.linkedin.our-phishing-domain.com`).
- `{subdomain}` : same as `{hostname}` but only for the subdomain.
- `{domain}` : same as `{hostname}` but only for the domain.
- `{domain_regexp}` : same as `{domain}` but translates to properly escaped regular expression string. This can sometimes be useful when replacing anti-phishing protections in javascript, that try to verify if `window.location` contains the legitimate domain.
- `{hostname_regexp}` : same as above, but for the hostname.
- `{subdomain_regexp}` : same as above, but for the subdomain.

In the example we have:

```
- {
 hostname: 'www.linkedin.com',
      sub: 'www',
      domain: 'linkedin.com',
      search: 'action="https://{hostname}',
      replace: 'action="https://{hostname}',
```

```
        mimes: ['text/html', 'application/json']
        }
```

This will make Evilginx search for packets with `Content-Type` of `text/html` or `application/json` and look for occurrences of `action="https://www\.linkedin\.com` (properly escaped regexp). If found, it will replace every occurrence with `action="https://www.totally.not.fake.linkedin.our-phishing-domain.com`.

As you can see this will replace the `action` URL of the login HTML form to have it point to Evilginx server, so that the victim does not stray off the phishing path.

That was the most complicated part. Now it should be pretty straight forward.

Next up are `auth_tokens`. This is where you define the **cookies that should be captured on successful login**, which combined together provide the full state of the website's captured session. The cookies defined here, when obtained, can later be imported to any browser (using this extension in Chrome) and allow to be immediately logged into the victim's account, bypassing any 2FA challenges.

- `domain` : original domain for which the cookies will be saved for.
- `keys` : array of cookie names that should be captured.

In the example, there is only one cookie that LinkedIn uses to verify the session's state. Only `li_at` cookie, saved for `www.linkedin.com` domain will be captured and stored.

Once Evilginx captures all of the defined cookies, it will display a message that authentication was successful and will store them in the database.

The two following parameters are similar `user_regex` and `pass_regex`. These define the POST request keys that should be searched for occurrences of usernames and passwords. Searching is defined by a regular expression that is ran against the contents of the POST request's key value.

- `key` : name of the POST request key.
- `re` : regular expression defining what data should be captured from the key's value (e.g. `(.*)` will capture the whole value)

Last parameter is `landing_path` array, which holds URL paths to login pages (usually one), of the phished website.

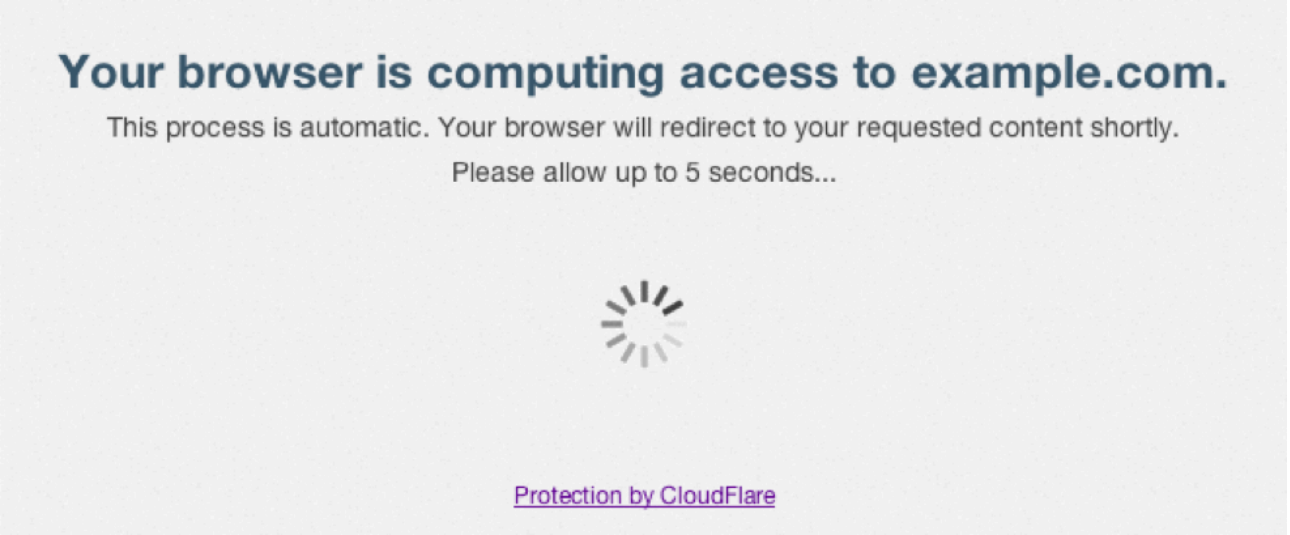
In our example, there is `/uas/login` which would translate to `https://www.totally.not.fake.linkedin.our-phishing-domain.com/uas/login` for the generated phishing URL.

Hope that sheds some light on how you can create your own phishlets and should help you understand the ones that are already shipped with Evilginx in the `./phishlets` directory.

Future development

I'd like to continue working on Evilginx 2 and there are some things I have in mind that I want to eventually implement.

One of such things is serving an HTML page instead of 302 redirect for hidden phishlets. This could be a page imitating CloudFlare's "checking your browser" that would wait in a loop and redirect, to the phishing page, as soon as you unhide your phishlet.



Another thing to have at some point is to have Evilginx launch as a daemon, without the UI.

Update: You can find out about version 2.1 release here

Business Inquiries

If you are a red teaming company interested in **development of custom phishing solutions**, drop me a line and I will be happy to assist in any way I can.

If you are giving presentations on flaws of 2FA and/or promoting the use of FIDO U2F/FIDO2 devices, I'd love to hear how Evilginx can help you raise awareness.

In any case, send me an email at: kuba@breakdev.org

I'll respond as soon as I can!

Credits

Since the release of Evilginx 1, in April last year, a lot has changed in my life for the better. I met a lot of wonderful, talented people, in front of whom I could exercise my impostor syndrome!

I'd like to thank few people without whom this release would not have been possible:

@evilsocket - for letting me know that Evilginx is awesome, inspiring me to learn GO and for developing so many incredible products that I could ~~steal~~ borrow code from!

@antisnatchor and **@h0wlu** - for organizing WarCon and for inviting me!

@juliocesarfort and **@Mario_Vilas** - for organizing AlligatorCon and for being great reptiles!

@x33fcon - for organizing x33fcon and letting me do all these lightning talks!

Vincent Yiu (@vysecurity) - for all the red tips and invitations to secret security gatherings!

Kevin Mitnick (@kevinmitnick) - for giving Evilginx a try and making me realize its importance!

@i_bo0om - for giving me an idea to play with nginx's `proxy_pass` feature in his post.

Cristofaro Mune (@pulsoid) & **Denis Laskov (@it4sec)** - for spending their precious time to hear out my concerns about releasing such tool to the public.

Giuseppe "Ohpe" Trotta (@Giutro) - for a heads up that there may be other similar tools lurking around in the darkness ;)

#apt - everyone I met there, for sharing amazing contributions.

Thank you!

That's it! Thanks for being able to read this overly long post!

Enjoy the tool and I'm waiting for your feedback!

[>> Follow me on Twitter <<](https://twitter.com/mrgretzky)

[>> Download Evilginx 2 from GitHub <<](https://github.com/kgretzky/evilginx2)
